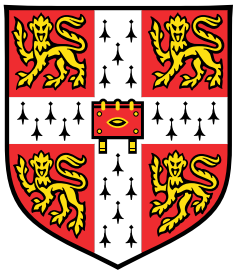
Improving Abstractive Summarization and Information Consistency Assessment



Potsawee Manakul

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

St John's College

February 2024

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Potsawee Manakul
February 2024

Abstract

Improving Abstractive Summarization and Information Consistency Assessment
Potsawee Manakul

Summarization is the process of compressing a document into a shorter form that contains all the relevant information. It is useful in many applications, for example, a brief summary of the content of an article or a podcast can help decide whether it should be read or listened to. This summarization process is often split into two distinct research areas: extractive summarization, which pulls out key phrases directly from the text; and abstractive summarization, which rephrases and condenses ideas from the text. This thesis examines abstractive summarization as it can provide a more effective and compact summary of the information than extractive summarization. In common with other research areas in natural language processing, deep learning has become the dominant technology in abstractive summarization demonstrating significant performance gains over more traditional approaches. Initially, these neural-based models were trained from scratch, with randomly initialized parameters, relying solely on supervised training. As this meant there was usually limited training data, the generated summaries often had limited diversity and fluency. The recent shift to using foundation models, trained on vast quantities of text exploiting self-supervised training schemes, addressed some of these issues such as fluency. However, there still remain issues such as how to handle long documents. A key challenge to developing these abstractive summarization systems is assessment. In common with many natural language generation tasks, it is not possible to generate an exhaustive set of reference summarises, so simple lexicon-matching approaches have limited accuracy. This thesis investigates both the generation and assessment of abstractive summaries. In addition, approaches motivated by summary assessment are applied to hallucination detection in large language models.

The first area examined in this thesis includes two aspects of summary generation. The first aspect is associated with the diversity issue when training a model from scratch with limited data. The second aspect is associated with applying foundation models to long-document summarization. Initially, recurrent neural networks (RNNs) were the dominant form of abstractive summarization models. However, these models can often produce

summaries with low diversity. This lack of diversity means that the models may not fully capture or maximize the available information. To address this problem, this work proposes metrics based on hierarchical representations to explicitly maximize information usage in the source and encourage diversity in the summary. The second aspect addressed is to improve foundation model based systems to efficiently handle long documents. The contributions in this part are three-fold. First, sentence filtering methods for content selection are examined for both the training and inference stages. Second, this work presents design considerations for local attention aimed at improving the efficiency of transformer models. By combining local attention and sentence filtering, high-performance long-document summarization is achieved. Third, this work investigates encoder-decoder attention and demonstrates that its cost can be critical at inference. Based on this finding, sentence structure and sparsity in encoder-decoder attention are exploited, resulting in an improvement in the complexity of the attention with minimal performance degradation.

The second area investigated is summary assessment. Ideally, summaries would be assessed manually. However, this is highly expensive, motivating the need to develop automatic assessment methods. There are many attributes associated with good summaries such as fluency and consistency. Since generation systems based on foundation models are highly fluent, this work focuses on information consistency between the source document and the summary. This work proposes a question answering (QA) based approach to measure the consistency. In contrast to previous QA approaches, a multiple-choice QA framework is proposed. This allows information consistency to be approximated by computing the expected statistical distance between summary and source answer distributions. Additionally, current large language models (LLMs) have been shown to have strong zero-shot abilities in various NLP tasks. Hence, this thesis investigates whether LLMs can be used for summary assessment in a zero-shot manner, and proposes a comparative assessment framework.

Information consistency can be useful in a wide range of tasks in addition to summary assessment. Another application of information consistency explored in this thesis is hallucination detection in LLMs. A hallucination, in the context of LLMs, is a response which contains non-factual information with respect to actual knowledge. This work applies information consistency methods, initially developed for summarization, to wider LLM generation to enhance the reliability of LLMs. Specifically, this work proposes SelfCheckGPT that measures the consistency between stochastically generated responses from an LLM. SelfCheckGPT does not require an external database and it can be applied in a black-box manner, making it applicable in a variety of situations. This work demonstrates the effectiveness of SelfCheckGPT through experiments based on GPT-3's hallucinated contents.

Acknowledgements

I would like to express my deepest gratitude to my PhD supervisor, Prof. Mark Gales, for his invaluable support and guidance. I am very thankful for all the insightful discussions that I have had with Mark. Also, I am thankful for all the opportunities to work in AI that Mark provided. In addition, I would like to thank other academic staff, Dr. Kate Knill, Prof. Philip Woodland (my advisor), Prof. Ben Simon (college tutor), and Dr. Georgina Evans (college tutor) for their support. I would also like to thank my lab members: Yiting (Edie) Lu, Qingyun Dou, Adian Liusie, Vatsal Raina, Vyas Raina, Yassir Fathullah, Rao Ma, Charles McGhee, Yu Wang, Linlin Wang, Xixin Wu, Mengjie Qian, Stefano Banno, Erfan Loweimi, and Simon McKnight for their helpful discussion and collaboration. I would also like to thank my family and many other people I met during my PhD for supporting me during this journey. Finally, I would like to thank the Cambridge Commonwealth, European and International Trust, St John's College, and Cambridge University Press & Assessment (CUP&A) for sponsoring my PhD.

I would like to dedicate this thesis to my loving parents ...

Table of contents

List of figures

List of tables

Notation

Abbreviations

AI	Artificial Intelligence
ASR	Automatic Speech Recognition
CPU	Central Processing Unit
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
LLM	Large Language Model
LSTM	Long Short-Term Memory
MBR	Minimum Bayes Risk
NER	Named Entity Recognition
NLG	Natural Language Generation
NLP	Natural Language Processing
QA	Question Answering
QG	Question Generation
RL	Reinforcement Learning
RMSE	Root Mean Squre Error
RNN	Recurrent Neural Network
WER	Word Error Rate

Symbols

$[;]$	Concatenation
$\boldsymbol{\theta}$	Model parameters
$\Delta\boldsymbol{\theta}$	Gradient of model parameters
$\mathbb{E}$	Expectation
$\mathbf{x}$	Vector
$\mathbf{x}_{1:N}$	Sequence of vector value of length N
∇f	Differential operator on the function f
X	Entity (e.g., sentence or document)
x	Scalar
$x_{1:N}$	Sequence of scalar of length N
$\sigma(.)$	Activation function
$f(.;\boldsymbol{\theta})$	Generic function parameterized by $\boldsymbol{\theta}$
$f(.)$	Generic function
$P(.)$	Probability density function or probability mass function
$P(.;\boldsymbol{\theta})$	Probability density function or probability mass function parameterized by $\boldsymbol{\theta}$

Superscripts

(i)	Instance (i) from a batch of data instances
T	Transpose
$\mathtt{s}$	Sentence-level
$\mathtt{w}$	Word-level

Subscripts

i	General index (e.g., position, time)
$i:j$	Indices from $i, i+1, ..., j$

Chapter 1

Introduction

1.1 Context and Motivation

Automatic summarization is the process of creating a shorter version of a text that still maintains all the important details. Automatic summarization can typically be achieved through two primary methods. The first method, known as extractive summarization, involves selecting salient segments directly from the text. The second method, called abstractive summarization, focuses on generating novel text that captures the main concepts from the original document. The primary goal of automatic summarization systems is to create concise summaries that capture the key points of a document, using few words and minimizing redundancy [239]. These systems enable users to understand the main ideas of the original document without having to read it in its entirety [211].

Before the era of deep learning, summarization methods were predominantly unsupervised extractive methods such as graph-based ranking methods [198, 64] or maximum marginal relevance [18]. Pre-deep-learning abstractive summarization methods were also developed such as deletion-based [138] or statistical models [8]. In the era of deep learning, abstractive summarization is more widely used than extractive summarization [271]. This is because abstractive summarization allows for more flexible and coherent summaries that can integrate information from various parts of the source text. This approach can produce summaries that are more readable and engaging for humans, as they often resemble the natural way humans summarize information. Thus, this thesis focuses on abstractive summarization.

Early work in neural abstractive summarization adopted recurrent neural network (RNN) [29, 208, 263] or convolutional neural network [252, 29]. To handle long documents such as transcripts of spoken documents or long articles, hierarchical RNN architectures were developed [39, 160, 349]. However, there remain challenges such as the diversity in the

generation from training these neural models from scratch on limited data [288, 263, 98]. This work examines this diversity problem in detail on spoken document summarization. This is because existing spoken document data is limited in size and has extra challenges such as long contexts, disfluencies in speech, and errors in transcription, which could lead to diversity issues in the generated summaries.

Given the advances in self supervised learning, abstractive summarization approaches have recently shifted to using pre-trained transformer-based [290] foundation models [169, 159, 340]. In this paradigm, large foundation models are first pre-trained on a large amount of raw text before being fine-tuned on an abstractive summarization task. This paradigm has shown impressive results on standard summarization tasks such as news summarization [159, 340]. However, there is a challenge in applying a large foundation model to long-document summarization such as meeting summarization, podcast summarization, or scientific paper summarization. This is because the time and memory requirements of standard self-attention in the transformer scale quadratically with sequence length. Thus, this work examines methods to improve transformer-based systems to efficiently handle long documents.

A further important challenge is how to evaluate summarization systems. Although human evaluation is considered the gold standard, it can be expensive, time-consuming, and difficult to reproduce. This necessitates automatic summary assessment. Current generation systems based on large pre-trained models [159, 340] are highly fluent, but these systems have been shown to generate false or unsupported information [143]. This phenomenon is also commonly referred to as *hallucination* [348, 193]. Hence, another area of investigation is on developing automatic methods for assessing *information consistency*, which is to evaluate whether the generated outputs are factually consistent with their source documents.

Lastly, hallucination is not limited to summarization tasks, and it is also a problem in broader generative tasks [123]. Recently, there has been mass adoption of large language models (LLMs) such as GPT [15] on a variety of tasks. However, despite the impressive generation ability of LLMs, they may produce unfaithful responses, similar to when summarization systems generate inconsistent summaries. Thus, this work extends information consistency methods, initially developed for information consistency assessment in summarization, to wider LLM generation.

Research Questions

To address the challenges discussed previously, this thesis considers these research questions:

1. How well do abstractive text summarization based on a hierarchical RNN perform on abstractive spoken document summarization and can a multi-task learning framework improve the performance?
2. Can sentence filtering based on the attention of hierarchical RNN improve long-input summarization?
3. How do we improve the encoder's attention for long-input summarization?
4. How do we improve the encoder-decoder's attention for abstractive summarization?
5. How do we measure information consistency between the summary and the source document?
6. Can we perform summary assessment in a zero-shot manner by prompting large language models, and is comparative assessment better than point-wise assessment?
7. Can we apply summary assessment methods to the hallucination problem in large language models?

Methodology

This section has described the research questions and challenges that will be investigated in this thesis. In the field of NLP, a general methodology is as follows:

1) Ideation: The first step is to identify a challenge or gap in the existing literature. This stage involves literature reviews to explore current approaches, trends, challenges and limitations in the domain of interest. In this work, literature reviews (and related fundamental methods) are presented in the chapters 2, 3, and 4.

2) Data Preparation: After the challenge or research problem is defined, the next step is data preparation. In NLP, research work could either utilize existing datasets when working on a standard task or benchmark, or collect a new dataset for a new task. To work on existing datasets, the process typically data includes cleaning (e.g., removing noise, normalizing text, etc) and converting raw text into a structured format. To collect a new dataset, this may involve web scrapping, or transforming publicly available datasets.

3) Model Development: Once data is clean and ready, the next step is model development which includes selecting appropriate model architectures, training frameworks, optimization algorithms, training criteria, or hyperparameter setting. Data is typically split into training, validation, and test sets, and the model is trained on the training set. During this development

step, the trained model is validated on the validation set, and the training process terminates when the performance on the validation set does not improve.

4) Automatic Evaluation: During the development stage, the model is trained on a criterion which may not align with the real-world metric (as the training criterion typically needs to be differentiable), so in this stage a metric that better aligns with the real-world use-case is adopted, for example, accuracy, precision, recall, or F1-score. In addition, further analysis can be performed to identify the current model's limitations and failed examples. Note that automatic metrics (e.g., ROUGE for summarization) were shown effective when evaluating those models in the early years of development. However, as models become better and more fluent, automatic metrics, especially for natural language generation tasks, often fail to capture semantic meaning, coherence, and contextual appropriateness.

5) Iterative Improvement: After evaluating and identifying current limitations, the next iteration of training, validation, and evaluating continues to refine the current approach until the method or the model achieves a satisfactory performance level.

6) Human Evaluation: Once the model achieves satisfactory performance (e.g., outperforming its baseline or achieving state-of-the-art performance in some benchmark), it might be used on a real application where humans evaluate its usefulness. We note that this step can be time-consuming or it is difficult to reproduce, and it is more common in the industry.

1.2 Thesis Outline

This thesis starts with background knowledge which is split into deep learning, foundation models, and summarization. Then, contributions in two topics of summary generation, summary assessment, and information consistency in generative language models are presented.

In the background knowledge sections in Chapter 2, the fundamentals of deep learning, including model architectures, training methods, inference methods, ensemble methods, and tokenization are established. Chapter 3 discusses foundation models in NLP and related topics such as self-supervised learning, transformer architectures and the variants, and the properties of transformer-based models. Chapter 4 introduces the main tasks in the field of summarization, including extractive summarization, abstractive summarization, and summary assessment, and existing work in each of these areas.

In terms of the contributions, Chapter 5 improves abstractive spoken document summarization, proposes a multi-task learning framework including dialogue act prediction, extractive labelling, and a novel diversity loss, and studies the impact of ASR systems on spoken

document summarization. Chapter 6 improves abstractive summarization with foundation models, and the focus is on long input spans and efficient modelling. Three main contributions in this chapter are (1) sentence filtering where training-time and inference-time filtering methods are proposed, (2) efficient encoder attention where local attention is investigated and applied to summarization as well as combined with sentence filtering, and (3) efficient encoder-decoder attention where the sparsity in this is observed and exploited. In addition to these three core contributions, the experimental work in this chapter also includes ensemble methods in summarization. Chapter 7 improves automatic summary assessment with a focus on information consistency in summarization. This chapter proposes a novel multi-choice question-answering framework for assessing summaries. Zero-shot assessment methods, which exploit large language models, are studied including a standard prompting and a novel comparative assessment framework. Motivated by the summary assessment methods, Chapter 8 extends these techniques to a broader information consistency problem in generative language models, and proposes SelfCheckGPT which is a family of hallucination detection methods. Finally, Chapter 9 provides conclusions of the work and discusses limitations and potential future directions.

1.3 Published Work

Part of the PhD research in this thesis has been published in peer-reviewed natural language processing and speech conferences and also released as an arXiv pre-print.

Peer-reviewed Publications

• [188] P. Manakul, M.J.F. Gales, L. Wang, "*Abstractive Spoken Document Summarization Using Hierarchical Model with Multi-Stage Attention Diversity Optimization*", In Proceedings of Interspeech 2020.
→ The work was done in collaboration with Linlin Wang who worked on producing ASR transcripts of the AMI dataset. My contribution was designing and executing the experiments, and writing the paper. This work is a part of Chapter 5.

• [184] P. Manakul and M.J.F. Gales, "*CUED_SPEECH at TREC 2020 Podcast Summarisation Track*", In Proceedings of Text REtrieval Conference (TREC) 2020.
→ Part of this work is included in Chapter 6.

• [185] P. Manakul and M.J.F. Gales, "*Long-Span Summarization via Local Attention and Content Selection*", In Proceedings of the 59th Annual Meeting of the Association for

Computational Linguistics (ACL) 2021.
→ This work is a part of Chapter 6.

• [186] P. Manakul and M.J.F. Gales, "*Sparsity and Sentence Structure in Encoder-Decoder Attention of Summarization Systems*", In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP) 2021.
→ This work is a part of Chapter 6.

• [183] P. Manakul, Y. Fathullah, A. Liusie, V. Raina, V. Raina, and M.J.F. Gales, "*CUED at ProbSum 2023: Hierarchical Ensemble of Summarization Models*", In the 22nd Workshop on Biomedical Natural Language Processing and BioNLP Shared Tasks (BioNLP) 2023.
→ This work was done in collaboration with Yassir Fathullah who worked on training summarization models; Adian Liusie who worked on the selection of abstractive summarization baselines and training models; Vyas Raina who worked on MBR decoding; Vatsal Raina who worked on data access and project coordination. My contribution was extractive summarization, zero-shot summarisation, token-level ensemble, and hierarchical ensemble combination. Part of this work is included in Chapter 6.

• [189] P. Manakul, A. Liusie and M.J.F. Gales, "*MQAG: Multiple-choice Question Answering and Generation for Assessing Information Consistency in Summarization*", In Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (IJCNLP-AACL) 2023.
→ The work was done in collaboration with Adian Liusie who helped discuss the concept and helped with paper writing. This work is a part of Chapter 7.

• [190] P. Manakul, A. Liusie and M.J.F. Gales, "*SelfCheckGPT: Zero-Resource Black-Box Hallucination Detection for Generative Large Language Models*", In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP) 2023.
→ The work was done in collaboration with Adian Liusie who helped discuss the concept and helped with paper writing (given our time constraint, the initial draft writing was possible with the help from Adian). This work is a part of Chapter 8.

• [173] A. Liusie, P. Manakul, and M.J.F. Gales, "*LLM Comparative Assessment: Zero-shot NLG Evaluation through Pairwise Comparisons using Large Language Models*", In Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (EACL) 2024.
→ The work was done in collaboration with Adian Liusie who led the project and set up the

framework for running the experiments. My contribution was in ideation and designing the experiments on summary assessment, and also on the results on (part of) SummEval and Podcast Assessment. This work is a part of Chapter 7.

arXiv Pre-print

• [187] P. Manakul and M.J.F. Gales, "*Podcast Summary Assessment: A Resource for Evaluating Summary Assessment Methods*", arXiv pre-print 2022.
→ This work is a part of Chapter 7 and Appendix A.

Chapter 2

Deep Learning Fundamentals

Deep learning, a subset of machine learning, has advanced the state of the art in Natural Language Processing (NLP), and it has revolutionized model architectures and representation learning [330]. Deep learning is centred on complex neural network architectures, usually with millions of connections and non-linearity, that are inspired by biological neurons [112]. Neural networks consist of multiple layers of interconnected nodes, or neurons, that process and transform data. Neural networks excel at automatically learning hierarchical features and representations from raw data [142], enabling it to tackle complex tasks such as image recognition [34, 276], natural language processing [279, 83], and speech processing [93, 94]. This chapter about deep learning fundamentals will delve into model architectures, training methods, inference methods, ensemble techniques, and tokenization.

There are a range of model architectures in deep learning such as feedforward networks, recurrent neural networks, and transformers. Each of these architectures is tailored to a specific task. Training methods involve optimization algorithms and loss functions that learn model parameters, while inference methods apply these trained models to unseen data for predictions. Ensemble techniques combine multiple models to improve performance and robustness. Lastly, as NLP tasks are typically sequential modelling tasks, the process of breaking down text into units is a crucial step, which is referred to as tokenization and will be described in this chapter.

2.1 Model Architectures

In deep learning, a neural network, parameterized by $\boldsymbol{\theta}$, is designed to learn a complex mapping function f from an input X to an output Y with minimal manual feature engineering,

$$Y = f(X; \boldsymbol{\theta}) \tag{2.1}$$

where the input X and output Y can take any form such as scalar x, vector $\mathbf{x}$, sequence of scalar $x_{1:N}$, or sequence of vectors $\mathbf{x}_{1:N}$ with examples being text, image, audio, video, time series, etc. The remaining of this section will provide the background of neural networks that will be used in this thesis.

2.1.1 Feedforward Neural Network

A feedforward neural network (FNN) is characterized by its uni-directional flow of information from the input nodes to the output nodes [259]. A feedforward neural network may comprise multiple linear layers with a non-linear activation function. Assume $\mathbf{x}^{(l)}$ be the input and $\mathbf{x}^{(l+1)}$ be the output of layer l, the linear layer $f^{(l)}(.)$ computes the following:

$$\mathbf{x}^{(l+1)} = f^{(l)}(\mathbf{x}^{(l)}) \tag{2.2}$$

$$\mathbf{x}^{(l+1)} = \sigma^{(l)}\left(\mathbf{W}^{(l)}\mathbf{x}^{(l)} + \mathbf{b}^{(l)}\right) \tag{2.3}$$

where $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are model parameters associated with the linear l and $\sigma^{(l)}$ is a non-linear activation function. A stack of linear layers $l = 1, 2, ..., L$ where each layer can have different input and output dimensions and activation functions make up a feedforward neural network.

Activation Functions

Activation function in a neural network calculates the outputs given the inputs and weights such as $\sigma^{(l)}$ in Equation 2.3. In deep learning, activation functions are non-linear in order to solve nontrivial problems using a finite number of nodes [158]. Non-linear activation functions ensure that the output cannot be reproduced from a linear combination of the inputs. Without a non-linear activation function, a neural network, regardless of the size, will act as a single-layer network. The forms that will be used are provided in Table 2.1.

Activation $\sigma(.)$	**Output**
sigmoid(x)	$\frac{1}{1+\exp(-x)}$
tanh(x)	$\frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$
ReLU(x)	$\max(0,x)$
GELU(x)	$0.5x\left(1+\text{erf}(\frac{x}{\sqrt{2}})\right)$
softmax($\mathbf{x}$)	$\frac{\exp(x_i)}{\sum_{j=1}^{J}\exp(x_j)}$ for $i=1,...,J$

Table 2.1 Commonly-used activation functions. Note that all of the activation functions in the table except softmax are applied elementwise, while softmax is a function of all elements.

2.1.2 Recurrent Neural Network

A recurrent neural network (RNN) [112] has hidden states for processing sequential data $\mathbf{x}_{1:T}$. At time step t, the hidden state $\mathbf{h}_t$ is calculated as a function of the current input $\mathbf{x}_t$ and its previous hidden state $\mathbf{h}_{t-1}$,

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}) \tag{2.4}$$

The vanilla RNN applies linear matrices with a non-linear activation as shown in Equation 2.5, and the output at this time step, $\mathbf{z}_t$, is then computed as a function of the hidden state:

$$\mathbf{h}_t = \sigma_h(\mathbf{W}_h\mathbf{x}_t + \mathbf{U}_h\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{2.5}$$

$$\mathbf{z}_t = \sigma_z(\mathbf{W}_z\mathbf{h}_t + \mathbf{b}_z) \tag{2.6}$$

where σ_h and σ_z are non-linear activation functions such as those described in Section 2.1.1, and $\mathbf{W}_h$, $\mathbf{U}_h$, $\mathbf{b}_h$, $\mathbf{W}_z$, and $\mathbf{b}_z$ are the model parameters (weights) to be trained. An unrolled recurrent unit shown in Figure 2.1 can be viewed as a fully connected neural network except that the weights (and biases) are shared. Training an RNN is done by backpropagation through time [312] rather than the standard error backpropagation for feed-forward neural networks. Multiple recurrent units can be stacked together to form a deep RNN architecture. Two RNNs, one operating forwards and the other operating backwards, can be combined to

incorporate both past and future information, and this yields a bidirectional RNN:

$$\mathbf{h}_t^{\mathrm{f}} = f(\mathbf{x}_t, \mathbf{h}_{t-1}^{\mathrm{f}}) \tag{2.7}$$

$$\mathbf{h}_t^{\mathrm{b}} = f(\mathbf{x}_t, \mathbf{h}_{t+1}^{\mathrm{b}}) \tag{2.8}$$

$$\mathbf{h}_t = \left[\mathbf{h}_t^{\mathrm{f}}; \mathbf{h}_t^{\mathrm{b}}\right] \tag{2.9}$$

The vanilla RNN suffers a vanishing gradient and exploding gradient due to computations involved in the backpropagation through time [107]. As a result, long short-term memory (LSTM) [109] and gated recurrent unit (GRU) [28] are widely used because they are designed to mitigate gradient problems by using gates to control information flow.

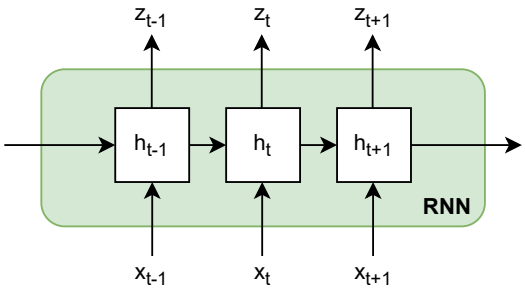


Fig. 2.1 RNN Architecture (unrolled in time)

Sequence Modelling Tasks

RNNs can be applied to different sequence modelling tasks as shown in Figure 2.2. In *sequence classification*, it is a many-to-one task, predicting the label y of the input sequence $\mathbf{x}_{1:T}$. This task is usually modelled by using the final time step output. Examples of sequence classification include sentiment classification and textual entailment. In *sequence labelling*, it is a many-to-many task where the input sequence and output sequence are of the same length. This task is to predict the label for each input time step. Examples of sequence labelling include token classification and grammatical error detection. In *sequence-to-sequence*, it is a many-to-many task where the input sequence and output sequence can be different lengths. Examples of sequence-to-sequence include machine translation and summarization. The next section will discuss sequence-to-sequence and encoder-decoder framework in more detail.

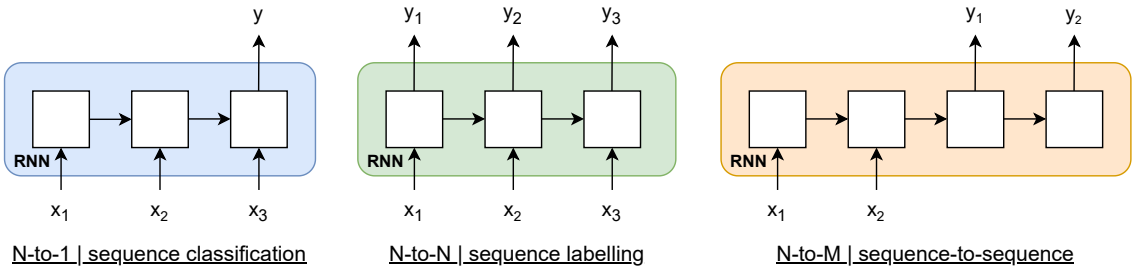


Fig. 2.2 The application of RNNs in sequence modelling tasks.

2.1.3 Encoder-Decoder Framework

In the previous section, sequence-to-sequence tasks were introduced, but Figure 2.2 (right) shows a simplified version of an RNN sequence-to-sequence model. In practice, a sequence-to-sequence model adopts an encoder-decoder architecture with two RNNs, which is described below in this section.

RNN Encoder-Decoder

In a sequence-to-sequence model, the input and the output can have different lengths. Given an input sequence $x_{1:N}$ and an output sequence $y_{1:M}$ (with their vector representations being $\mathbf{x}_{1:N}$ and $\mathbf{y}_{1:M}$, respectively), the goal of a sequence-to-sequence model is to estimate the conditional probability $P(y_{1:M}|x_{1:N})$. A sequence-to-sequence architecture is typically made up of *two* RNNs [279] as shown in Figure 2.3.

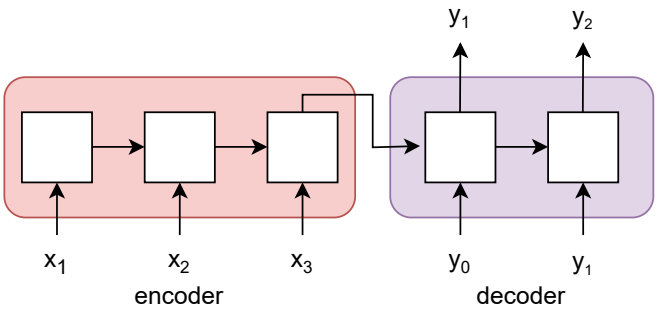


Fig. 2.3 RNN Encoder-Decoder Framework.

First, an RNN encoder $\boldsymbol{\theta}_{\text{enc}}$ maps the input sequence into a fixed-size context vector $\mathbf{c}$:

$$\mathbf{c} = f(x_{1:M}; \boldsymbol{\theta}_{\text{enc}}) \tag{2.10}$$

The context vector $\mathbf{c}$ is usually the output of the final timestep of the RNN encoder. Second, a decoder uses the context vector from the encoder and generates an output sequence. This RNN decoder $\boldsymbol{\theta}_{\text{dec}}$ estimates the conditional probability as follows,

$$P(y_{1:M}|x_{1:N}) = \prod_{m=1}^{M} P(y_m|y_{1:m-1}, x_{1:N}) \tag{2.11}$$

$$P(y_m|y_{1:m-1}, x_{1:N}) \approx P(y_m|y_{1:m-1}, \mathbf{c}; \boldsymbol{\theta}_{\text{dec}}) \tag{2.12}$$

Assume $\mathbf{d}_m$ is the hidden state of the decoder at time step m, the RNN computation is the same as in Equation 2.4, which is then followed by a softmax.

$$\mathbf{d}_m = f(\mathbf{y}_m, \mathbf{d}_{m-1}; \boldsymbol{\theta}_{\text{dec}}) \tag{2.13}$$

The decoder state is initialized using the context vector (in Equation 2.10), i.e, $\mathbf{d}_0 = \mathbf{c}$. To obtain the probability over the vocabulary, the distribution $P(y_m|y_{1:m-1}, \mathbf{c}; \boldsymbol{\theta}_{\text{dec}})$ in Equation 2.12 is represented using a softmax over all possible words:

$$P(y_m|y_{1:m-1}, x_{1:N}) = \text{softmax}(\mathbf{W}_d \mathbf{d}_m + \mathbf{b}_d) \tag{2.14}$$

where the model parameter $\mathbf{W}_d$ has the output dimension of the same size as the vocabulary.

Encoder-Decoder with Attention Mechanism

In a standard sequence-to-sequence model, it can be difficult to learn a long source sequence, as the source sequence is encoded into a single vector to initialize the decoder [5]. An attention mechanism is proposed to allow the decoder to *attend* to all the encoder states, and decide which part to attend at each time instance [5]. An attention mechanism can be considered as computing the context vector *dynamically* for each decoder time step as opposed to a fixed-size vector in the vanilla encoder-decoder architecture (Equation 2.10). Given the output vectors of the encoder $\{\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_N\}$, an encoder-decoder model with an attention mechanism computes the following:

$$e_{m,n} = \text{score}(\mathbf{d}_m, \mathbf{h}_n) \tag{2.15}$$

$$\alpha_{m,n} = \frac{\exp(e_{m,n})}{\sum_{n=1}^{N} \exp(e_{m,n})} \tag{2.16}$$

$$\mathbf{c}_m = \sum_{n=1}^{N} \alpha_{m,n} \mathbf{h}_n \tag{2.17}$$

The context vector $\mathbf{c}_m$ is concatenated with the decoder state $\mathbf{d}_m$ and the output distribution in Equation 2.14 is modified to,

$$P(y_m|y_{1:m-1}, x_{1:N}) = \text{softmax}(\mathbf{W}_d[\mathbf{d}_m; \mathbf{c}_m] + \mathbf{b}_d) \tag{2.18}$$

Lastly, it should be noted that there are other forms of attention mechanisms. Table 2.2 (adapted from Weng [311]) illustrates other forms of attention mechanism, which differ by how the attention scores $e_{m,n}$ are computed.

Attention Mechanism	$e_{m,n} = \text{score}(\mathbf{d}_m, \mathbf{h}_n)$
Content-based	$\frac{\mathbf{d}_m \cdot \mathbf{h}_n}{\|\mathbf{d}_m\| \, \|\mathbf{h}_n\|}$
Dot-product	$\mathbf{d}_m \cdot \mathbf{h}_n$
Scaled Dot-product	$\frac{\mathbf{d}_m \cdot \mathbf{h}_n}{\sqrt{D}}$
Additive	$\mathbf{v}_a^T \tanh(\mathbf{W}_a[\mathbf{d}_m; \mathbf{h}_n])$
General	$\mathbf{d}_m^T \mathbf{W}_a \mathbf{h}_n$

Table 2.2 Form of attention mechanisms. $\mathbf{W}_a$ and $\mathbf{v}_a$ are attention mechanism parameters. $\mathbf{d}_m$ and $\mathbf{h}_n$ are vectors of dimension D.

2.1.4 Transformer

The transformer architecture was introduced by Vaswani et al. [290] to replace RNNs entirely with *self-attention* layers. The Transformer model, firstly proposed Vaswani et al. [290], adopted an encoder-decoder architecture. Instead of using RNNs, both the encoder and decoder are a stack of self-attention layers. Each encoder layer comprises a self-attention layer and a feedforward layer, and each decoder layer comprises a self-attention layer, an encoder-decoder attention layer and a feedforward layer. A casual mask is applied on the decoder side to prevent it from using information from future time steps, and this allows the model to perform autoregressive decoding at inference. During training, the Transformer allows better parallelism compared to RNNs because each time step computation can be done independently of previous time steps, and this results in faster training [290]. Given that the Transformer can be trained efficiently (i.e., faster training), this architecture has become the default option in large-scale self-supervised learning, which will be discussed in Chapter 3.

This section will describe the building blocks of transformers. It should be noted the architecture is not limited to only the encoder-decoder architecture, and there are transformer-based encoder-only and decoder-only models. In addition, each component could have a

different implementation. This section will describe the realization of each component as adopted in the original Transformer [290] shown in Figure 2.4.

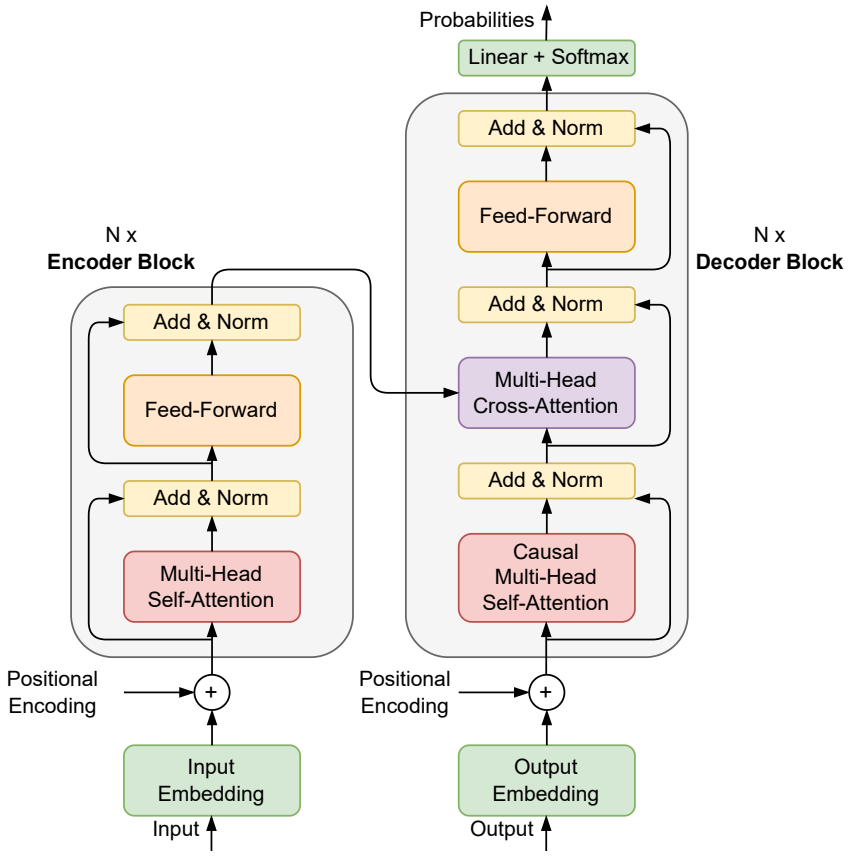


Fig. 2.4 Architecture of the original Transformer [290].

Multi-Head Self-Attention

Previously in Section 2.1.3, the attention mechanism between the encoder and decoder was described, and this attention mechanism allows the network to draw dependencies between the *input* and *output*. Instead, **self-attention** is designed to learn dependencies *within* the same sequence, similar to an RNN. Given a sequence of vectors $\{\mathbf{x}_1, ..., \mathbf{x}_N\}$, a self-attention layer computes the following to obtain the representation $\mathbf{h}_i$ for $i = 1, ..., N$:

- query, key, value:

$$\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i \tag{2.19}$$

$$\mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i \tag{2.20}$$

$$\mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i \tag{2.21}$$

- self-attention weights using scaled dot-product where D is the dimension:

$$\alpha_{i,j} = \text{softmax}\left(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{D}}\right) = \frac{\exp\left(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{D}}\right)}{\sum_{j'=1}^{N} \exp\left(\frac{\mathbf{q}_i \cdot \mathbf{k}_{j'}}{\sqrt{D}}\right)} \tag{2.22}$$

- state representation:

$$\mathbf{h}_i = \sum_{j=1}^{N} \alpha_{i,j} \mathbf{v}_j \tag{2.23}$$

The weights $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$ are specific to each layer and each head. In the Transformer, self-attention is multi-headed, meaning that there are multiple self-attention mechanisms running in parallel independently. Each of the heads operates on a subset of the total dimension. The outputs of all heads are then concatenated and linearly transformed into the final output.

For the encoder-decoder architecture, on the decoder side, there is also a multi-head cross-attention mechanism that allows the decoder to use information from the encoder. Given a sequence of encoder outputs $\{\mathbf{x}_1, ..., \mathbf{x}_N\}$ and a sequence of decoder input vectors $\{\mathbf{y}_1, ..., \mathbf{y}_M\}$, a cross-attention layer computes the following to obtain the representation $\mathbf{d}_j$ (corresponding to $\mathbf{y}_j$) for $j = 1, ..., M$:

- query (decoder side):

$$\mathbf{q}_j = \mathbf{W}_q \mathbf{y}_j \tag{2.24}$$

- key, value (encoder side):

$$\mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i \tag{2.25}$$

$$\mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i \tag{2.26}$$

- cross-attention weights:

$$\alpha_{j,i} = \text{softmax}\left(\frac{\mathbf{q}_j \cdot \mathbf{k}_i}{\sqrt{D}}\right) = \frac{\exp\left(\frac{\mathbf{q}_j \cdot \mathbf{k}_i}{\sqrt{D}}\right)}{\sum_{i'=1}^{N} \exp\left(\frac{\mathbf{q}_j \cdot \mathbf{k}_{i'}}{\sqrt{D}}\right)} \tag{2.27}$$

- state representation:

$$\mathbf{d}_j = \sum_{i=1}^{N} \alpha_{j,i} \mathbf{v}_i \tag{2.28}$$

The weight matrices are specific to each layer and each head in the same way as self-attention, and cross-attention is also multi-headed. Lastly, it should be noted that self-attention allows

each position to attend to all other positions. Since the decoder cannot attend to its future generated tokens at inference, causal masking is applied to the decoder self-attention layers.

Feedforward Layers

The outputs of each self-attention block are then passed into a feed-forward network, which typically consists of two linear layers with a non-linear activation function $\sigma(.)$, e.g. ReLU in the original Transformer [290], or GeLU in GPT models [242]. The feed-forward layers perform position-wise operations.

$$\text{Feedforward}(\mathbf{x}) = \mathbf{W}_{f_2}\left(\sigma(\mathbf{W}_{f_1}\mathbf{x} + \mathbf{b}_{f_1})\right) + \mathbf{b}_{f_2} \tag{2.29}$$

Positional Encoding

The self-attention mechanism is permutation-invariant. However, in natural language, the order of words in the text is important, and if omitted, semantics will change. Therefore, positional embedding is added to the token embedding before being passed to the first layer of attention blocks. Notable positional encoding methods are as follows:

• **Sinusoidal**: The original transformer [290] introduced sinusoidal positional embedding such that the embedding is fixed (i.e., not learnable) with each dimension having a different frequency. Assume that at position i, $\mathbf{x}_i$ is a token encoding vector, $\mathbf{p}_i = [p_{i,1}, p_{i,2}, ..., p_{i,D}]$ is a positional encoding vector of the same dimension where the elements of $\mathbf{p}_i$ are,

$$p_{i,2d} = \sin\left(\frac{i}{10000^{2d/D}}\right) \tag{2.30}$$

$$p_{i,2d+1} = \cos\left(\frac{i}{10000^{2d/D}}\right) \tag{2.31}$$

The motivation is this design may allow the model to extrapolate to longer sequences at the inference time. However, it has been shown that sinusoidal positional embeddings have a weak extrapolation ability [236].

• **Learned Embedding**: This method learns trainable positional encoding vectors $\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_N$ from scratch. This method is similar to the sinusoidal method in that $\mathbf{p}_i$ is added to the token encoding vector at the position i. This type of positional encoding is used in BERT [52]. It has a limitation that it cannot encode any position beyond the fixed number of learned positions. For example, BERT can only perform positional encoding up to 512 tokens.

• **T5 Bias**: The T5 model, proposed by Raffel et al. [244], uses a relative position method [270]. Instead of adding position information to token embeddings, this method injects positional information by modifying the way attention values are computed. The attention values are computed as before, but a learnable bias $b_{i,j}$ is added to each query-key score such that the self-attention weights in Equation 2.22 become,

$$\alpha_{i,j} = \text{softmax}\left(\frac{\mathbf{q}_i \cdot \mathbf{k}_j + b_{i,j}}{\sqrt{D}}\right) \tag{2.32}$$

This learnable bias depends on the distance between the query and the key, and the T5 model shares the position embedding parameters across all layers. The T5 model sets a maximum distance such that any larger distance will have the same learned bias – which allows T5 to have a sequence of any length[1] similar to the sinusoidal encoding method.

• **Rotary**: Rotary Position Embedding (RoPE) was introduced by Su et al. [278] and it has been adopted and popularized by GPT-J [299], Llama [286], Llama2 [286], PaLM [31]. Instead of adding sinusoidal embeddings to the input of the transformer, the rotary embedding multiplies the keys and queries at every attention layer by sinusoidal embeddings:

$$\alpha_{i,j} = \text{softmax}\left(\frac{(\mathbf{R}_i\mathbf{q}_i)^T\mathbf{R}_j\mathbf{k}_j}{\sqrt{D}}\right) = \text{softmax}\left(\frac{\mathbf{q}_i^T(\mathbf{R}_i^T\mathbf{R}_j)\mathbf{k}_j}{\sqrt{D}}\right) = \text{softmax}\left(\frac{\mathbf{q}_i^T\mathbf{R}_{j-i}\mathbf{k}_j}{\sqrt{D}}\right) \tag{2.33}$$

where $\mathbf{R}_i$ is a rotation matrix with rotation $\theta_d = 10000^{-2(d-1)/D}$,

$$\mathbf{R}_i = \begin{bmatrix} \cos(i\theta_1) & -\sin(i\theta_1) & 0 & 0 & \dots & 0 & 0 \\ \sin(i\theta_1) & \cos(i\theta_1) & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \cos(i\theta_2) & -\sin(i\theta_2) & \dots & 0 & 0 \\ 0 & 0 & \sin(i\theta_2) & \cos(i\theta_2) & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \cos(i\theta_{D/2}) & -\sin(i\theta_{D/2}) \\ 0 & 0 & 0 & 0 & \dots & \sin(i\theta_{D/2}) & \cos(i\theta_{D/2}) \end{bmatrix} \tag{2.34}$$

Essentially, the rotary encoding rotates different representation dimensions by θ_d. For two nearby positions, i.e. small distance $i - j$, the rotation $\mathbf{R}_{i-j}$ will be small. RoPE embedding injects position information into every layer instead of one time at the input, but unlike the T5 bias, RoPE embeddings are not constrained to a maximum distance.

[1] This only means that there will not be the same error observed in the learned embedding method of BERT; however, whether or not T5 bias can extrapolate beyond the maximum distance is a different aspect.

Layer Normalization

Layer normalization (LayerNorm) [2] is applied (after self-attention or feedforward layer as shown in Figure 2.4) to normalize across features for each individual sample in a (mini-)batch independently. Given a vector of D features, $\mathbf{x} = [x_1, x_2, ..., x_D]$, LayerNorm computes the following for $d = 1, ..., D$:

$$\mu = \frac{1}{D} \sum_{d=1}^{D} x_d \tag{2.35}$$

$$\sigma^2 = \frac{1}{D} \sum_{d=1}^{D} (x_d - \mu)^2 \tag{2.36}$$

$$\hat{x}_d = \frac{x_d - \mu}{\sqrt{\sigma^2 + \varepsilon}} \tag{2.37}$$

and $\hat{\mathbf{x}} = [\hat{x}_1, ..., \hat{x}_D]$, then the output of LayerNorm is:

$$\text{LayerNorm}(\mathbf{x}) = \gamma \hat{\mathbf{x}} + \beta \tag{2.38}$$

where γ and β are learnable parameters for each layer normalization layer. Layer normalization differs from batch normalization [120] which normalizes across the batch for each feature. LayerNorm does not depend on the batch size, which makes it especially useful for tasks where batch size might be dynamic or where using large batches is infeasible. Unlike batch normalization, which might have different behaviours during training and testing due to batch statistics, LayerNorm behaves consistently as it normalizes across features.

2.2 Sequence Training Methods

The training algorithms presented here are model-agnostic and applicable to all models that are parametric. As the optimization is performed using gradient ascend/descent (described in Section 2.2.3), the objective function must be differentiable, with the exception of sequence-level training objectives where the gradient is computed via an approximation (described in Section 2.2.2). In this section, the notation is as follows: $\boldsymbol{\theta}$ denotes the model parameters, $X = x_{1:N}$ the input sequence, and $Y = y_{1:M}$ the target sequence.

2.2.1 Token-level Loss

As mentioned in the previous section, a sequence-to-sequence model estimates the condition probability $P(Y|X)$. This section will describe a training method to maximize the likelihood of the conditional probability.

Maximum Likelihood Estimation

Although a model is trained to predict sentences or a sequence of tokens, the sentences are split into tokens[2] and the model is trained to maximize the likelihood of the ground-truth tokens given the input context. Maximum Likelihood Estimation (MLE) training is a method of estimating the parameters of a statistical model by maximising a likelihood function. Given a dataset of J source-target pairs, $\{(X^{(1)},Y^{(1)}),...,(X^{(J)},Y^{(J)})\}$ where $X^{(j)} = x^{(j)}_{1:N_j}$ and $Y^{(j)} = y^{(j)}_{1:M_j}$ denote the j-th instance, the likelihood function is defined below, and as training sequences $(X^{(1)},Y^{(1)}),...,(X^{(J)},Y^{(J)})$ are independent, the likelihood of the training sequences become the product of the likelihood of each sequence:

$$P\left(\{Y^{(1)},...,Y^{(J)}\}|\{X^{(1)},...,X^{(J)}\};\boldsymbol{\theta}\right) = \prod_{j=1}^{J} P\left(Y^{(j)}|X^{(j)};\boldsymbol{\theta}\right) \tag{2.39}$$

As the likelihood function can become very small resulting in the floating point precision issue, log-likelihood is usually maximized in practice. As the log function is monotonic, maximizing log-likelihood yields the same solution as maximizing the likelihood in Equation 2.39. The model parameters obtained by maximizing the likelihood is, therefore,

$$\boldsymbol{\theta}^*_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} \left(\frac{1}{J}\sum_{j=1}^{J} \log P\left(Y^{(j)}|X^{(j)};\boldsymbol{\theta}\right)\right) \tag{2.40}$$

Based on Equation 2.40, the MLE loss function (to be minimized) is defined as,

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{J}\sum_{j=1}^{J} \log P\left(Y^{(j)}|X^{(j)};\boldsymbol{\theta}\right) \tag{2.41}$$

As mentioned earlier, the sentence (or sentences) is split into a sequence of tokens, e.g. $Y = y_{1:M}$, for maximum likelihood training at the token level. The loss function is, therefore,

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{J}\sum_{j=1}^{J}\sum_{m=1}^{M_j} \log P\left(y^{(j)}_m|y^{(j)}_{1:m-1},x^{(j)}_{1:N};\boldsymbol{\theta}\right) \tag{2.42}$$

[2]Tokenization is discussed in Section 2.5.

In Equation 2.42, the loss is factorized across time. For gradient-based optimization, which will be described in Section 2.2.3, the gradient is:

$$\nabla \mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{J} \sum_{j=1}^{J} \sum_{m=1}^{M_j} \nabla \left(\log P \left(y_m^{(j)} | y_{1:m-1}^{(j)}, x_{1:N}^{(j)}; \boldsymbol{\theta} \right) \right) \tag{2.43}$$

For a differentiable neural model, the gradient in Equation 2.43 is simply the summation over time steps and independent training instances. In practice, the loss is optimized using *teacher forcing* (TF) [315] where $P(y_m^{(j)} | y_{1:m-1}^{(j)}, X^{(j)}; \boldsymbol{\theta})$ is computed using the reference output history $y_{1:m-1}^{(j)}$. Teacher forcing allows the training to be performed in parallel across time, and MLE and teacher forcing are the standard option in training sequence-to-sequence models. However, there are known issues due to the nature of MLE and teacher forcing:

- **Exposure Bias**: At the inference stage, a decoding algorithm is usually *free running* (FR), where the decoder has to use its own prediction from the previous time step. The model has not been trained to correct for errors in the history during training, and this mismatch may result in poor performance at inference [248, 300].
- **Metric Mismatch**: During training, the likelihood at the token level is maximized, while during inference, a sequence-level metric such as BLEU or ROUGE is used.
- **Diversity**: The generated outputs were observed to be generic and repetitive [263, 224]. One reason for this issue is that during training, the attention mechanism is more diverse as it is guided by ground-truth target tokens.

Learning Algorithms beyond MLE

To mitigate the exposure bias problem, scheduled sampling [11] gradually changes the training process from fully guided by ground-truth tokens towards generated tokens. As an improvement to scheduled sampling, professor forcing [153] and attention forcing [58] were developed to reduce to discrepancy in teacher forcing mode and free running mode. Note that Section 2.2.2 discusses sequence-level training that could reduce the metric mismatch problem, and Section 5.3 discusses and proposes methods to deal with the diversity problem in RNN approaches.

2.2.2 Sequence-level Loss

In practice, the generated sequences are typically evaluated at the sequence level (i.e., once generation is complete). Thus, this section discusses methods that directly optimize sequence-level metrics as these methods can be applied to achieve the highest evaluation score.

Minimum Bayes Risk (MBR) Training

Maximum likelihood is typically used for parameter optimization; however, once trained, the model performance is measured using non-differentiable and discrete metrics such BLEU in machine translation, ROUGE in summarization, or word error rate (WER) in automatic speech recognition. Due to this discrepancy, maximum likelihood trained models do not always yield the best performance on discrete evaluation metrics [224, 214, 235]. Minimum Bayes Risk (MBR) training optimizes the expected value of a discrete evaluation metric directly. The notation is X = the input sequence, Y = the reference sequence, $\tilde{Y}$ = a hypothesis sequence from all possible sequences $\mathcal{Y}$, and the discrete score = $R(Y,\tilde{Y})$. The expected discrete score of sequences generated by model $\boldsymbol{\theta}$ can as written as follows,

$$\bar{R}(X,Y;\boldsymbol{\theta}) = \mathbb{E}_{\tilde{Y}\sim P(\tilde{Y}|X;\boldsymbol{\theta})}\left[R(Y,\tilde{Y})\right] = \sum_{\tilde{Y}\in\mathcal{Y}} R(Y,\tilde{Y})P(\tilde{Y}|X;\boldsymbol{\theta}) \tag{2.44}$$

The MBR training (for $j = 1,...,J$ independent training instances), which is designed to directly maximize the expected reward, is defined as:

$$\boldsymbol{\theta}^*_{\text{MBR}} = \arg\max_{\boldsymbol{\theta}} \left(\frac{1}{J} \sum_{j=1}^{J} \bar{R}(X^{(j)},Y^{(j)};\boldsymbol{\theta}) \right) \tag{2.45}$$

which is usually done by computing the gradient of the expected reward defined above. For simplicity of notation, the summation over instances $1,...,J$ is omitted. For gradient-based learning methods, the gradient is, therefore,

$$\nabla\bar{R}(X,Y;\boldsymbol{\theta}) = \nabla\left(\sum_{\tilde{Y}\in\mathcal{Y}} R(Y,\tilde{Y})P(\tilde{Y}|X;\boldsymbol{\theta}) \right) \tag{2.46}$$

$$\nabla\bar{R}(X,Y;\boldsymbol{\theta}) = \sum_{\tilde{Y}\in\mathcal{Y}} R(Y,\tilde{Y})\nabla P(\tilde{Y}|X;\boldsymbol{\theta}) \tag{2.47}$$

Although Equation 2.47 is differentiable with respect to $\boldsymbol{\theta}$, the summation over all possible sequences Y is intractable. In the ASR community, the gradient in Equation 2.47 is approximated by sampling [235, 254, 268] or N-best sequences [235, 232] as follows:

• **Gradient Sampling**:

$$\nabla \bar{R}(X,Y;\boldsymbol{\theta}) = \sum_{\tilde{Y}\in\mathcal{Y}} R(Y,\tilde{Y}) \left[\nabla \log P(\tilde{Y}|X;\boldsymbol{\theta})\right] P(\tilde{Y}|X;\boldsymbol{\theta}) \tag{2.48}$$

$$\approx \frac{1}{K} \sum_{Y^{(k)}\sim P(\tilde{Y}|X;\boldsymbol{\theta})} R(Y,Y^{(k)}) \left[\nabla \log P(Y^{(k)}|X;\boldsymbol{\theta})\right] \tag{2.49}$$

where $Y^{(k)}$ is sampled from $P(\tilde{Y}|X;\boldsymbol{\theta})$. To stabilize training, Prabhavalkar et al. [235], Shannon [268] subtract the reward in Equation 2.49 by the mean of samples to reduce the variance of the estimate:

$$\nabla \bar{R}(X,Y;\boldsymbol{\theta}) \approx \frac{1}{K} \sum_{Y^{(k)}\sim P(\tilde{Y}|X;\boldsymbol{\theta})} \left[R(Y,Y^{(k)}) - \tilde{R}\right] \left[\nabla \log P(Y^{(k)}|X;\boldsymbol{\theta})\right] \tag{2.50}$$

where the mean of samples is,

$$\tilde{R} = \frac{1}{K} \sum_{k} R(Y,Y^{(k)}) \tag{2.51}$$

• N**-best List**: Although gradient sampling approximation yields an unbiased estimate, it requires a large number of samples to achieve a good estimate. N-best list approximation works on the assumption that top-N sequences (e.g., from beam search decoding) account for the majority of the probability $\sum_{\tilde{Y}} P(\tilde{Y}|X;\boldsymbol{\theta})$. Let $\hat{\mathcal{Y}} = \{\hat{Y}^{(1)},...,\hat{Y}^{(N)}\}$ be the set of N-best hypotheses. The gradient in Equation 2.47 can be approximated by the N-best list and becomes tractable as follows:

$$\nabla \bar{R}(X,Y;\boldsymbol{\theta}) = \sum_{n=1}^{N} R(\hat{Y},Y^{(n)}) \nabla \hat{P}(\hat{Y}^{(n)}|X;\boldsymbol{\theta}) \tag{2.52}$$

where

$$\hat{P}(\hat{Y}^{(n)}|X;\boldsymbol{\theta}) = \frac{P(\hat{Y}^{(n)}|X;\boldsymbol{\theta})}{\sum_{Y^{(m)}\in\hat{\mathcal{Y}}} P(Y^{(m)}|X;\boldsymbol{\theta})} \tag{2.53}$$

Note that a potential drawback of this method limits the exploration ability. The top-N hypotheses are likely to be close in the search space, especially for a long sequence.

An Interesting Interpretation: MBR Training and Reinforcement Learning

When casting a sequence training problem as a Reinforcement Learning (RL) problem, it can be shown that the policy gradient update is the same as the gradient sampling of the MBR criterion in Equation 2.50. RL algorithms seek to find a policy $P(\tilde{Y}|X;\boldsymbol{\theta})$, which is optimal such that it maximizes the expected reward. This expected reward is defined as the

MBR criterion in Equation 2.44. From a reinforcement learning (RL) perspective, Equation 2.48 and Equation 2.49 can be derived by using the REINFORCE algorithm [314] which is a Monte Carlo policy gradient algorithm. Instead of subtracting the mean of samples, as done in Equation 2.50, it can be shown that when computing the reward with respect to a baseline b that does not depend on Y, the expected gradient remains unchanged [251]:

$$\begin{aligned}
\nabla \bar{R}(X,Y;\boldsymbol{\theta}) &= \mathbb{E}_{\tilde{Y}}\left[\big(R(Y,\tilde{Y})-b\big)\nabla \log P(\tilde{Y}|X;\boldsymbol{\theta})\right] && (2.54)\\
&= \mathbb{E}_{\tilde{Y}}\left[R(Y,\tilde{Y})\nabla \log P(\tilde{Y}|X;\boldsymbol{\theta})\right] - \mathbb{E}_{\tilde{Y}}\left[b\nabla \log P(\tilde{Y}|X;\boldsymbol{\theta})\right] && (2.55)\\
&= \mathbb{E}_{\tilde{Y}}\left[R(Y,\tilde{Y})\nabla \log P(\tilde{Y}|X;\boldsymbol{\theta})\right] - b\sum_{\tilde{Y}\in\mathcal{Y}}\left[\nabla \log P(\tilde{Y}|X;\boldsymbol{\theta})\right]P(\tilde{Y}|X;\boldsymbol{\theta}) && (2.56)\\
&= \mathbb{E}_{\tilde{Y}}\left[R(Y,\tilde{Y})\nabla \log P(\tilde{Y}|X;\boldsymbol{\theta})\right] - b\nabla\sum_{\tilde{Y}\in\mathcal{Y}}P(\tilde{Y}|X;\boldsymbol{\theta}) && (2.57)\\
&= \mathbb{E}_{\tilde{Y}}\left[R(Y,\tilde{Y})\nabla \log P(\tilde{Y}|X;\boldsymbol{\theta})\right] && (2.58)
\end{aligned}$$

Hence, an approximation term b is added to reduce the variance of the gradient estimate. Examples of RL applications are Rennie et al. [251] which applied this RL training on image captioning tasks, and Paulus et al. [224] which applied it to abstractive summarization and computed the reward of the sequence generated using greedy search, $R(Y,\hat{Y})$, as b.

2.2.3 Optimization Algorithms

This section has described token-level and sequence-level losses for training sequence models. Training a neural model with parameters $\boldsymbol{\theta}$ is the process of finding $\boldsymbol{\theta}$ that minimizes a training criterion/loss, and this subsection will describe *gradient-based* optimization algorithms.

However, due to the large number of variables in a neural model, there is almost never any practical way to find a closed-form solution for the minimum. Hence, iterative methods such as gradient descent with a differentiable neural network and a loss function are often the only viable options for finding the minimum [253]. This iterative optimization process is referred to as *Gradient Descent* (GD) which takes the following form,

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\Delta\boldsymbol{\theta}_t \tag{2.59}$$

where $\boldsymbol{\theta}_t$ is the model parameter t at iteration t. The gradient is,

$$\Delta\boldsymbol{\theta}_t = \frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}|_{\boldsymbol{\theta}_t} \tag{2.60}$$

where η is the learning rate. In practice, after a pre-defined number of iterations, the loss is computed on a validation set. When the validation loss does not improve a certain number of times (e.g., once or a few times), the iterative optimization is stopped.

Stochastic Gradient Descent (SGD) is a popular extension of GD which takes the derivative on randomly drawn samples (i.e., a mini-batch) instead of the entire dataset. As the gradient computed from a mini-batch will be noisy, *momentum* term is used to reduce the variance in SGD, and accelerate the convergence towards the relevant direction. The SGD with momentum method can be written as follows,

$$\mathbf{m}_t = \gamma \mathbf{m}_{t-1} + \eta \Delta \boldsymbol{\theta}_t \tag{2.61}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{m}_t \tag{2.62}$$

where $\Delta \boldsymbol{\theta}_t$ is the gradient defined in Equation 2.60 and γ is a hyperparameter controlling the number of previous gradients (usually set to 0.9). As the learning rate η can influence the model performance significantly, it is important to tune η when using the SGD and SGD with momentum algorithms. To make learning algorithms less sensitive to η, improved optimizers have been proposed such that learning rates can be parameter-specific as well as adaptive. Notable adaptive learning algorithms are AdaGrad [60], RMSProp [92], AdaDelta [339], and Adam [136]. The Adam (adaptive momentum) optimizer has been widely adopted in the machine learning community, and it performs the gradient update as follows,

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\Delta \boldsymbol{\theta}_t \tag{2.63}$$

$$\mathbf{v}_t = \beta_2 \mathbf{b}_{t-1} + (1-\beta_2)(\Delta \boldsymbol{\theta}_t)^2 \tag{2.64}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \varepsilon} \quad \text{where} \quad \hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t}, \hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t} \tag{2.65}$$

where $\Delta \boldsymbol{\theta}_t$ is the gradient and $(\Delta \boldsymbol{\theta}_t)^2$ is the elementwise square of $\Delta \boldsymbol{\theta}_t$. The authors of the Adam optimizer [136] suggest the values of the hyperparameters to be $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-8}$, which will be used in this work.

Learning Rate Scheduling

Another important ingredient that exists in all gradient-based optimization methods is the learning rate η. When fine-tuning a pre-trained language model, the learning rate η is usually set to follow a schedule where: (1) the first stage has the learning rate starts from 0.0 and increases to the maximum value, and (2) the second stage has the learning rate decays from the maximum value [290, 169]. This schedule is to allow adaptive optimizers (e.g., Adam

optimizer) to gain enough statistics during the early stage (of training and not yet change parameters too dramatically (i.e., low learning rate). The first stage is called *warm-up*. Later, the learning rate decreases such as following an exponential decay pattern.

$$\eta_t = \begin{cases} \eta_0 t/\tau & t \leq \tau \\ \eta_0 \exp(-\alpha t) & t > \tau \end{cases} \tag{2.66}$$

where τ is the number of warm-up steps, α is the decay rate, and η_0 is the peak learning rate.

2.3 Inference Methods

The previous section discussed how the distribution $P(Y|X;\boldsymbol{\theta})$ is obtained (i.e., model training). This section is concerned with how to *generate* a target sequence given the distribution (i.e., the trained model).

2.3.1 Maximum Likelihood Decoding

At training (in Section 2.2), the distribution parameterized by model weights $P(Y|X;\boldsymbol{\theta})$ is optimized by maximizing the likelihood of the target sequence Y. At inference, the task is to find the most *likely* hypothesis sequence given the trained model,

$$Y_{\text{MLE}} = \arg\max_{Y} P(Y|X;\boldsymbol{\theta}) \tag{2.67}$$

The form in Equation 2.67 is a decoding method based on maximum likelihood estimation (MLE). However, considering all possible sequences Y is intractable. Given the common architectures are RNNs and transformers, traditional dynamic programming methods such as the Viterbi algorithm [293] are inefficient [195]. Generally, in a sequence-to-sequence task, autoregressive models are used to factorize the joint distribution,

$$P(Y|X;\boldsymbol{\theta}) = \prod_{m=1}^{M} P(y_m|y_{1:m-1},X;\boldsymbol{\theta}) \tag{2.68}$$

This factorization allows the model to generate one token at each time instance, and the following approximation methods for maximum likelihood decoding are used in practice:

- **Greedy search**: Greedy search decoding makes the locally optimal choice at each time instance by selecting,

$$\hat{y}_m = \arg\max_y P(y|\hat{y}_{1:m-1}, X; \boldsymbol{\theta}) \tag{2.69}$$

- **Beam search**: Beam search decoding extends greedy search decoding from keeping one candidate to keeping top-K candidates at each time where K is the beam width.

- **Sampling**: Greedy search yields a deterministic output, which may not be preferable in scenarios that value creativity. An alternative approach is to perform sampling at each time step. To control the amount of randomness, a temperature hyperparameter τ is incorporated before the final softmax layer to rescale the logits for each candidate token.

$$y_m^{\mathtt{s}} \sim P(y|y_{1:m-1}^{\mathtt{s}}, X; \boldsymbol{\theta}) \tag{2.70}$$

  Other variants of sampling-based decoding include: (1) top-K sampling which only considers the most probable K tokens (2) top-p sampling (i.e., nuclear sampling) [111] which samples from the top-p most probable tokens that collectively have a probability greater than or equal to p.

For greedy search, beam search, and sampling-based decoding methods, the back history of the conditional probability is based on the model's prediction. This type of decoding is referred to as free running (FR) mode. In contrast, when the back history is the ground truth (e.g., during training), it is referred to as teacher forcing (TF).

2.3.2 MBR Decoding

In Section 2.3.1, maximum likelihood based inference methods are described. These methods make locally optimal choices, which do not guarantee and generally do not yield the global optimal output. This subsection, on the other hand, considers a decoding method that maximizes a global criterion. Let $P(Y|X)$ be the true distribution of the target Y given the source X, e.g., all possible summaries $\mathcal{Y}$ are available. The quality of hypothesis sequence $\tilde{Y}$ as measured by the Bayes Risk [146] is

$$R_{\mathtt{B}}(X, \tilde{Y}) = \mathbb{E}_{Y \sim P(Y|X)}[R(Y, \tilde{Y})] = \sum_{Y \in \mathcal{Y}} R(Y, \tilde{Y}) P(Y|X) \tag{2.71}$$

where $R(Y, \tilde{Y})$ is the discrete score of hypothesis $\tilde{Y}$ with respect to reference Y. Note that in Section 2.2.2, Y was used to denote the reference sequence as it was assumed there is only

one gold standard sequence in that case. Whereas here it is assumed that the true distribution of the reference $P(Y|X)$ is known. The hypothesis that minimizes the Bayes risk is

$$Y_{\text{B}} = \arg\min_{\tilde{Y}\in\mathcal{Y}} \sum_{Y\in\mathcal{Y}} R(Y,\tilde{Y})P(Y|X) \tag{2.72}$$

This MBR decoding can be thought of as choosing a consensus output. However, in practice, it is not feasible to obtain the true distribution $P(Y|X)$, and the space $\mathcal{Y}$ of all possible sequences is too large. Therefore, approximation methods have been considered,

• **Single Models**: N-best list of hypotheses, $\hat{\mathcal{Y}} = \{\hat{Y}^{(1)},...,\hat{Y}^{(N)}\}$, can be generated by the model $P(Y|X;\boldsymbol{\theta})$, and the MBR decoding method can be implemented as:

$$Y_{\text{B}} = \arg\min_{\tilde{Y}\in\hat{\mathcal{Y}}} \sum_{n=1}^{N} R(\hat{Y}^{(n)},\tilde{Y})\hat{P}(\hat{Y}^{(n)}|X;\boldsymbol{\theta}) \tag{2.73}$$

where $\hat{P}(\hat{Y}^{(n)}|X;\boldsymbol{\theta})$ is defined the same as in Equation 2.53.

• **Multiple Models**: Based on M one-best outputs from M models, $\hat{\mathcal{Y}} = \{Y^{(1)},...,Y^{(M)}\}$, the MBR decoding method can be implemented as:

$$Y_{\text{B}} = \arg\min_{\tilde{Y}\in\hat{\mathcal{Y}}} \sum_{m=1}^{M} R(\hat{Y}^{(m)},\tilde{Y})\hat{P}(\hat{Y}^{(m)}|X;\{\boldsymbol{\theta}^{(1)},...,\boldsymbol{\theta}^{(M)}\}) \tag{2.74}$$

where

$$\hat{P}(\hat{Y}^{(m)}|X;\{\boldsymbol{\theta}^{(1)},...,\boldsymbol{\theta}^{(M)}\}) = \frac{\sum_{m'=1}^{M} P(\hat{Y}^{(m)}|X;\boldsymbol{\theta}^{(m')})}{\sum_{k=1}^{M}\sum_{m'=1}^{M} P(\hat{Y}^{(k)}|X;\boldsymbol{\theta}^{(m')})} \tag{2.75}$$

This MBR decoding using multiple models can be thought of as a system combination method (i.e., ensemble method). The next section will describe other ensemble methods.

2.4 Ensemble Methods

Ensemble methods combine multiple learning algorithms or generative models to obtain better performance than a single model. Training individual models could be prone to overfit specific aspects, i.e., local minima of the loss function, especially when working with a small dataset, so multiple models are trained so that each model can potentially capture different aspects of the data that are generalizable and not prone to overfitting. Combining diverse models together can often create a more robust and accurate prediction model [273]. Various approaches can be used to create diverse individual systems. A simple approach is to

use different weights' initialization for different seeds [152]. Alternatively for pre-trained systems, one can set different random seeds, which will influence training dropout and stochastic gradient descent batch creation, resulting in variability in the final models' weights. One can also use a form of data *bagging*, where a different subset of the data is used to train each model [77].

Given dataset $\mathcal{D}$, let $P(\boldsymbol{\theta}|\mathcal{D})$ be the model distribution such that an ensemble of models $\{\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \boldsymbol{\theta}^{(3)}, ...\}$ can be drawn from. A possible model combination method is **weight averaging**:

$$\boldsymbol{\theta}^{\texttt{Avg}} = \frac{1}{M} \sum_{m=1}^{M} \boldsymbol{\theta}^{(m)} \tag{2.76}$$

Although weight averaging across training runs has shown success in image classification [318], weight averaging across different training runs is expected to work only when individual runs operate in similar weight spaces. This limits the types of combinations for weight averaging to only models with the same architecture and the same input format. Alternatively, a Bayesian approach can be applied to system combination as follows,

$$P(Y|X,\mathcal{D}) = \int_{\boldsymbol{\theta}} P(Y|X,\boldsymbol{\theta}) P(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta} \tag{2.77}$$

$$\approx \frac{1}{M} \sum_{m=1}^{M} P(Y|X;\boldsymbol{\theta}^{(m)}), \quad \text{where } \boldsymbol{\theta}^{(m)} \sim P(\boldsymbol{\theta}|\mathcal{D}) \tag{2.78}$$

However, Equation 2.77 is intractable, so the Monte Carlo approximation is used in Equation 2.78. In practice, one can obtain a set of neural network models $\{\boldsymbol{\theta}^{(1)}, ..., \boldsymbol{\theta}^{(M)}\}$ from different initialization points, different data shuffles, or different model topologies. Due to the nature of the factorization property in Equation 2.68, combining the generative models can be done at either the token level or the sequence level as follows:

• **Token-level ensemble** (product-of-expectations):

$$P(Y|X) = \prod_{t=1}^{T} \left[\frac{1}{M} \sum_{m=1}^{M} P(y_t|y_{1:t-1}, X; \boldsymbol{\theta}^{(m)}) \right] \tag{2.79}$$

• **Sequence-level ensemble** (expectation-of-products):

$$P(Y|X) = \frac{1}{M} \sum_{m=1}^{M} \left[\prod_{t=1}^{T} P(y_t|y_{1:t-1}, X; \boldsymbol{\theta}^{(m)}) \right] \tag{2.80}$$

Maximum likelihood decoding methods in Section 2.3.1 can be used to decode both token-level combination (Eqn. 2.79) and sequence-level combination (Eqn. 2.80). In addition, based on the MBR decoding (Section 2.3.2), one could decode each model $P(Y|X;\boldsymbol{\theta}^{(m)})$ separately to obtain one (or more) sequence $\hat{Y}^{(m)}$, and use the MBR decoding to combine all sequences to obtain one output.

Hierarchical Ensemble

In addition to standard ensemble approaches, this work *introduces* Hierarchical Ensemble. The motivation is that in a task where many base models could be trained, instead of using only one type of combination, the hierarchical ensemble approach combines two levels of combination: token-level and sequence-level (through MBR decoding). An example of a hierarchical ensemble is depicted in Figure 2.5.

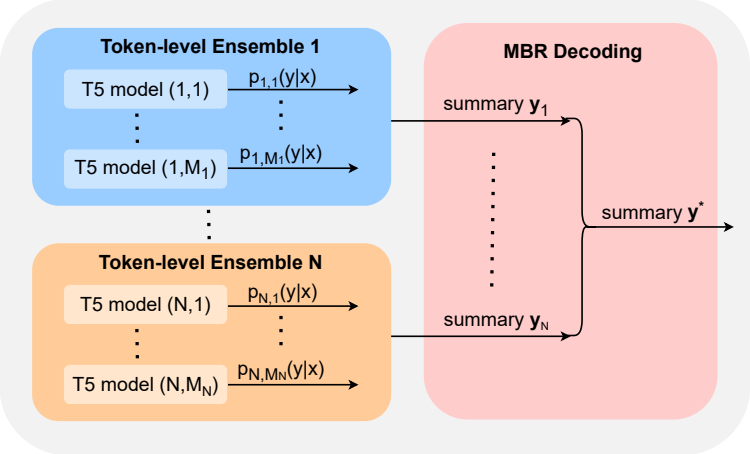


Fig. 2.5 Hierarchical ensemble of generative models (token-level ensemble and MBR decoding for sequence-level combination) where each individual model is fine-tuned independently. Although this figure uses T5 as an example of the base model, it can be replaced by any generative model.

Section 6.5.7 will examine the performance of ensemble techniques, including the proposed hierarchical ensemble on the Spotify podcast summarization challenge at TREC 2020, and the medical problem list summarization challenge at BioNLP 2023.

2.5 Tokenization

The previous sections described how models operate. It should be noted that these models work with numbers, not raw text. Thus, a process to convert raw text into numerical numbers is required. This process is called *tokenization* – transforming raw text into a format that

models can understand and process. Tokenization breaks down texts into smaller, manageable units (referred to as tokens [308]) such as words, subwords, or characters. By converting text into tokens, each token can be mapped to a numerical identifier, making it possible for the model to operate on the text data. For example, numerical identifiers of an input text are mapped into vector representations via an embedding layer. Also, at the generation stage, numerical identifiers are mapped back into tokens. Depending on the granularity, tokenization can be broadly categorized into word-level, character-level, and subword-level.

Word-level

The text is broken down into individual words. This is a common approach to tokenization, especially in languages (e.g., English) where words are separated by spaces. For example, the sentence "`This is Monday`" can be simply split by the spaces, yielding [`This`, `is`, `Monday`]. To handle a more complex text (e.g., with contractions, punctuation, commas), a standard word-tokenization toolkit NLTK[3] and spaCy[4] apply additional steps. For example, the default word tokenizer in NLTK is the Treebank tokenizer. It uses regular expressions to tokenize text as in Penn Treebank [192]. The additional steps for this tokenizer are: (1) split contractions (e.g., `don't` → `do n't`), (2) treat punctuations as separate words, (3) split commas and single quotes when followed by a space, (4) split periods at the end of line.

For languages without word boundaries (e.g., Thai, Chinese, Japanese), word tokenization is more challenging. For these languages, there are word segmentation algorithms such as longest matching [206] or maximal matching [317].

Character-level

The text is broken down into individual characters. It was first introduced to deep learning in the Character RNN model [130]. This approach treats each character as a token. For example, the sentence "`This is Monday`" can be split into [`T`, `h`, `i`, `s`, `_`, `i`, `s`, `_`, `M`, `o`, `n`, `d`, `a`, `y`]. This method offers the finest granularity, ensuring no out-of-vocabulary issues, and it can be useful for languages without clear word boundaries. However, its disadvantages are that it can lead to very long sequences and characters may not be able to capture semantic information as effectively as words.

[3] NLTK: https://www.nltk.org/
[4] spaCy: https://spacy.io/

Subword-level

Consider the word "tokenization". It might be considered a rare word and could be decomposed into "token" + "ization". Both of these *subwords* are more likely to appear standalone. Also, the meaning of "tokenization" is kept by the composite meaning of "token" and "ization". Typically, a 750-word document is about 1000 tokens. This process allows us to have good coverage (thus, a small percentage of unknown tokens) while keeping the vocabulary size small. Examples of widely used subword tokenization algorithms are summarized below.

• **Byte-Pair Encoding (BPE)** [266]: This subword tokenization method starts with a vocabulary of individual characters and it iteratively merges the most frequent pairs of symbols until a desired vocabulary size is reached. **Byte-Level BPE** is a variation of BPE that operates at the byte level. Byte-level BPE begins with a vocabulary of individual bytes instead of starting with a vocabulary of individual characters or words. This makes it language-agnostic and capable of handling any raw text, including special characters and even binary data. Models such as GPT-2 use byte-level BPE tokenization.

• **WordPiece** [52]: This subword tokenization method starts with a base vocabulary of individual characters from the dataset. This base vocabulary is often initialized with all the characters present in the training data. Instead of simply counting frequencies like in BPE, WordPiece tries to maximize the likelihood of the training data with respect to the model. During each iteration of vocabulary expansion, WordPiece considers adding a new token that will result in the biggest gain in the data's likelihood. New tokens are added until the desired vocabulary size is reached or until the likelihood gain falls below a certain threshold. When tokenizing a text, if a word is not in the vocabulary, the word is iteratively split into smaller subwords until the subwords are found in the vocabulary or reduced to individual characters. To differentiate between a subword that starts a word and a subword that is in the middle of a word, WordPiece typically uses a special prefix, often ##. For example, the word "unhappiness" might be tokenized as [un, ##happiness] or [un, ##happi, ##ness]. WordPiece tokenization was proposed and used in BERT [52].

• **SentencePiece** [144]: BPE and WordPiece have the same problem in that they assume the input text uses spaces to separate words. However, not all languages use spaces to separate words (e.g., Thai or Chinese). This problem can be addressed by using a language-specific pre-tokenizer as done in XLM [154]. SentencePiece handles this problem more generally by treating the input as a raw input stream, including the space in the set of characters to use. It then uses the BPE or WordPiece algorithm to construct the vocabulary. Note that decoding

with SentencePiece is easy since all tokens can simply be concatenated and `space_token` is replaced by a space. Models such as T5 use byte SentencePiece tokenization.

2.6 Chapter Summary

This chapter discussed deep learning fundamentals. Section 2.1 covered standard architectures, including feedforward neural network, recurrent neural network, encoder-decoder architecture, and transformer. Section 2.2 discussed training methods, including token-level training objective, sequence-level objective, and optimization algorithms. Section 2.3 discussed inference methods, including maximum likelihood decoding and MBR decoding. Section 2.4 discussed ensemble methods as well as introducing hierarchical ensemble. Lastly, Section 2.5 explained tokenization methods, which are an important pre-processing stage in text processing. The materials in this chapter are not specific to any particular experiments or investigations in the following parts of the thesis, and they are applicable to deep learning based methods broadly.

Chapter 3

Foundation Models in NLP

Foundation models[1] are large-scale neural network models that are (pre-)trained on a large quantity of data. Foundation models harness the increasingly larger computational resources and available large datasets to achieve state-of-the-art performance across a range of tasks, from natural language processing, computer vision, to speech processing. The term 'foundation' is used to describe these models because they provide a versatile basis upon which more specialized, application-specific models can be constructed [14].

Previously, the dominant pre-training method was transfer learning (TL), which uses labelled data to learn a good representation of neural network models from one task and transfer it to another task [358]. Recently, a new pre-training approach – Self-Supervised Learning (SSL) – has demonstrated improved results on a wide range of applications [6]. TL and SSL are both applied to learn a good representation of a model before applying the model to a target task. However, in contrast to TL, SSL does not require annotated labels as SSL creates auxiliary or pseudo tasks from unlabelled data.

3.1 Self-Supervised Learning (SSL)

Self-supervised learning (SSL) is an important approach that underpins advances in machine learning and generative AI. SSL is a methodology that can learn from unlabelled data. SSL is different from supervised learning which requires labelled data, which makes it difficult to scale up training data. SSL is similar to unsupervised learning in that it does not require labelled data; however, unlike unsupervised learning, SSL exploits inherent structure in unlabelled data, allowing SSL to obtain supervisory signals from the data itself.

[1] In the pre-training and fine-tuning paradigm, foundation models are also referred to as *pre-trained models*. In NLP, foundation models are also referred to as *large language models (LLMs)*.

3.1.1 General Approach in SSL

Self-supervised learning generates labels (i.e., the targets for supervised learning) from the unlabelled data without manual or weak label sources. The idea is that a part of the input (regardless of whether it is text, image, or speech unit) can be hidden or modified, such that the original data can be used as the ground-truth target. Self-supervised tasks are also referred to as *pretext* tasks. Common type pretext tasks include: recovering the input from corruption and predicting the future from the past. The general approach (illustrated in Figure 3.1) in any domain can be simplified into (1) transforming the unlabelled data into a pretext task, e.g., by masking or adding noise; (2) predicting the targets which are generally created from the original data; (3) computing the loss which can then be backpropagated to train the model. The exact realization of each component in SSL depends on the domain, and we provide some examples below.

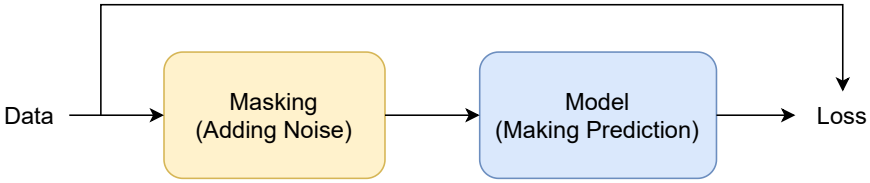


Fig. 3.1 General Approach in SSL. The data can be in any domain including texts, images, audio or time series. The model can be any architecture, but in modern deep learning, models are mostly transformer-based as it allows SSL on a large amount of data to be parallelized using GPUs. Widely-used architectures are discussed in Section 2.1.4.

In NLP, a common SSL approach is to predict hidden parts of the input or to reconstruct a perturbed input. In this example, SSL exploits unlabelled data (e.g., texts) by hiding or perturbing the texts, and the supervisory signals are the hidden or perturbed parts. This SSL task is also known as Mask Language Modelling (MLM), which has shown success in representation learning. In MLM, the model has to learn good representation in order to predict masked words from the surrounding contexts. Another common SSL task is to predict next the word conditioned on previous words, which has improved generative tasks. The applications of SSL in NLP will be discussed in more detail in Section 3.1.2.

In speech processing, Oord et al. [216] proposed a pretext task that maximizes the mutual information between the input features and the contextual representation. wav2vec 2.0 [4] is trained by predicting speech units for masked parts where the representation is quantized and negative examples are chosen from masked segments in the same utterance. Hidden Unit BERT (HuBERT) [115] follows the architecture of wav2vec2.0; HuBERT is trained on

a masked language modelling loss similar to BERT, but the targets are obtained by using k-means clustering to obtain a discrete set of outputs.

In computer vision, Gidaris et al. [85] were one of the first to apply SSL to computer vision tasks where the SSL task is to predict the amount of rotation applied to the original images. SimCLR [23] identifies augmented pairs as an SSL task. Similar to MLM in NLP, analogous tasks such as BYOL [95] learn to predict masked patches in an image. SSL has improved the results in object classification, object segmentation, object detection tasks, etc.

The supervised pre-training stage enables the model to learn general-purpose feature representation for downstream or target tasks. After the pre-training stage, the model can be applied to a target task in two standard ways: (1) fine-tuning the model either entirely or partially on the target task, and (2) applying the model in a zero-shot manner, which is more common for large foundation language models. Before delving into SSL in NLP, it should be noted that another important concept in SSL is contrastive learning, which is to distinguish correct (positive) samples from wrong (negative) ones.

Contrastive Learning

Contrastive learning aims to learn representation by pulling semantically close neighbours (e.g., similar sentences) together and pushing apart non-neighbours [100]. Contrastive learning assumes a set of paired examples $\{(X^{(j)}, \tilde{X}^{(j)})\}_{j=1,2,\ldots,J}$ such that $\tilde{X}^{(j)}$ is a positive example of $X^{(j)}$ while $\tilde{X}^{(k)}$ is a negative example when $k \neq j$. The conservative learning loss given the set of paired examples (or mini-batch of paired examples in practice) is:

$$\mathcal{L}_{\text{CT}}^{(j)}(\boldsymbol{\theta}) = -\log \left(\frac{\exp\left(\frac{\texttt{sim}(\mathbf{h}^{(j)}, \tilde{\mathbf{h}}^{(j)})}{\tau}\right)}{\sum_{j'=1}^{J} \exp\left(\frac{\texttt{sim}(\mathbf{h}^{(j)}, \tilde{\mathbf{h}}^{(j')})}{\tau}\right)} \right) \tag{3.1}$$

where $\mathbf{h}^{(j)} = f(X^{(j)}; \boldsymbol{\theta})$ is the model-based representation of $X^{(j)}$, the function $\texttt{sim}(.,.)$ measures similarity between two embeddings, and temperature τ is a hyperparameter. For example, in sentence representation learning, SimCSE [79] generates similar (i.e., *positive*) samples by applying dropout masks while diverse (i.e., *negative*) samples are other sentences in the same mini-batch. SimCSE achieved state-of-the-art performance in sentence representation learning. Researchers in different domains have proposed various contrastive learning based SSL objectives, which could be considered variants of Equation 3.1 [6].

3.1.2 Applications of SSL in NLP

In NLP, a large amount of raw (unlabelled or no target sequence) texts are readily available such as the Common Crawl data[2] or the RefinedWeb data [225] which has already been preprocessed. Thus, SSL is adopted in NLP to utilize the massive amount of unlabelled data, and this has led to significant performance improvement in modern NLP models such as BERT [52] or GPT [242]. The following will describe notable SSL techniques in NLP.

Word Embeddings

One of the first applications of SSL is word embedding [199, 227], which converts each word into a vector space representation. This vector space representation of words is expected to map semantically similar words closer to each other in the space. The most widely used word embedding models are word2vec [199] and GloVe [227] both of which are based on unsupervised learning. However, both word embeddings are context-independent, which means that they have a fixed embedding for each word regardless of the context (e.g., sentence). ELMo [228] is a context-dependent word embedding as it is trained to predict the next words. These early works in word embeddings use a recurrent neural network (RNN) as the backbone, and they were mostly invented before transformers. RNNs suffer from gradient vanishing [107, 108], capturing long-range dependencies [155], and non-parallelizability during training [290, 226]. These limitations prevent RNNs from performing SSL at scale. Subsequent works, therefore, have shifted to using transformer-based backbones, which can be trained more efficiently. This started a new trend in designing self-supervised tasks for pre-training transformer-based models, which we discuss below.

Masked Language Modelling (MLM)

In Masked Language Modelling (MLM), some words (or more precisely tokens and typically with a pre-defined percentage masking) in unlabelled texts are masked. The model is trained to predict the masked tokens using the remaining context. For example, Bidirectional Encoder Representations from Transformers (BERT) [52] is an encoder-only model pre-trained on a masked language modelling (MLM) objective. In this setup, the final hidden representations corresponding to masked tokens are passed to a softmax layer, and the training criterion is to maximize the probability of the masked tokens. For BERT pre-training, unlabelled texts are corrupted by random masking at around 15% of the tokens. BERT also has a second pre-training objective, which is to predict whether the two-sentence input is related (as the texts are the concatenation of two sentences either drawn randomly or from adjacent

[2] https://commoncrawl.org/

pairs). The pre-training data comprises a book corpus (800M words) and English Wikipedia (2,500M words).

Causal Language Modelling (CLM)

As opposed to the MLM objective, the causal language modelling (CLM) objective is to predict the next word. For example, GPT-2 [242] is a causal or unidirectional language model trained on 8 million web pages (40GB of text). A GPT model is causal in the sense that during training, a causal mask which prevents each token from using future information is applied, and hence, models trained on the CLM objective are suited for text generation tasks.

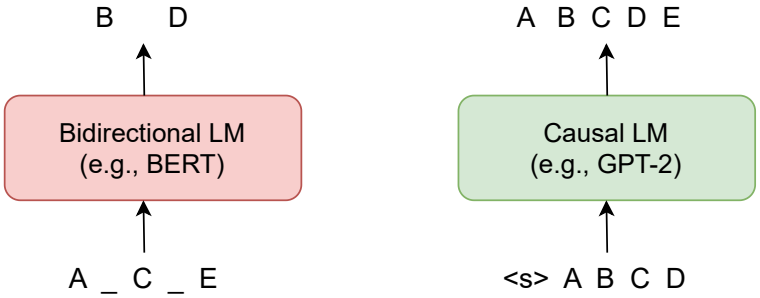


Fig. 3.2 BERT (left) is a bidirectional language model pre-trained with the MLM objective to predict masked tokens. BERT is good for extracting word or sentence representations, but it is not suitable for generation. GPT-2 (right) is a causal language model pre-trained with the CLM objective to predict the next words, and it is suitable for generation tasks.

3.1.3 Main Types of Transformers

In NLP, transformer architectures have paved the way for significant advances. Due to their performance, transformers have become the standard backbones of many sequential and NLP tasks such as text classification [329], machine translation [290], or grammatical error correction [127]. In summarization, Liu and Lapata [169] was one of the first to show that transformer models can be comparable with existing RNN models, and with pre-trained weights, transformer models set new state-of-the-art results. Given the impressive results of transformers, this subsection delves into the details of the architectures, which primarily manifest in three configurations: encoder-only, decoder-only, and encoder-decoder models.

• **Encoder-Only**: Encoder-only or autoencoding models are trained on the MLM objective. The dependencies between tokens are bi-directional such that each token attends to all other tokens in the input sequence. As a result, autoencoding models are suitable for natural language understanding tasks such as text classification, sentiment analysis, named entity

recognition, or token classification. Widely used autoencoding models include BERT [52], RoBERTa [170], XLM [154], ELECTRA [36], DeBERTa-v3 [102].

• **Decoder-Only**: Decoder-only or autoregressive models are trained on the CLM objective. The dependencies between tokens are uni-directional such that each token can only attend to the past tokens (e.g., through causal masking). As a result, autoregressive models are suitable for text generation. In addition to generation, autoregressive models have been shown to have emergent behaviour (i.e., capable of classification tasks) once trained at scale [15, 309, 31]. Widely used autoregressive models include the GPT family of models [241, 242, 15], BLOOM [257], OPT [342], LLaMA [286].

• **Encoder-Decoder**: Encoder-decoder or sequence-to-sequence models are trained on an objective to reconstruct corrupted text spans (e.g., masked tokens and swapped position). The encoder is an autoencoding model, so it has bi-directional dependencies and is suitable for understanding the input. The decoder is an autoregressive model, so it has uni-directional dependencies which allow it to perform generation. Because the encoder-decoder model combines the strengths of the previous two architectures, it facilitates both the understanding and generation of text, which is pivotal in complex undertakings like translation, summarization, and question-answering. Widely used sequence-to-sequence models include BART [159] and T5 [244].

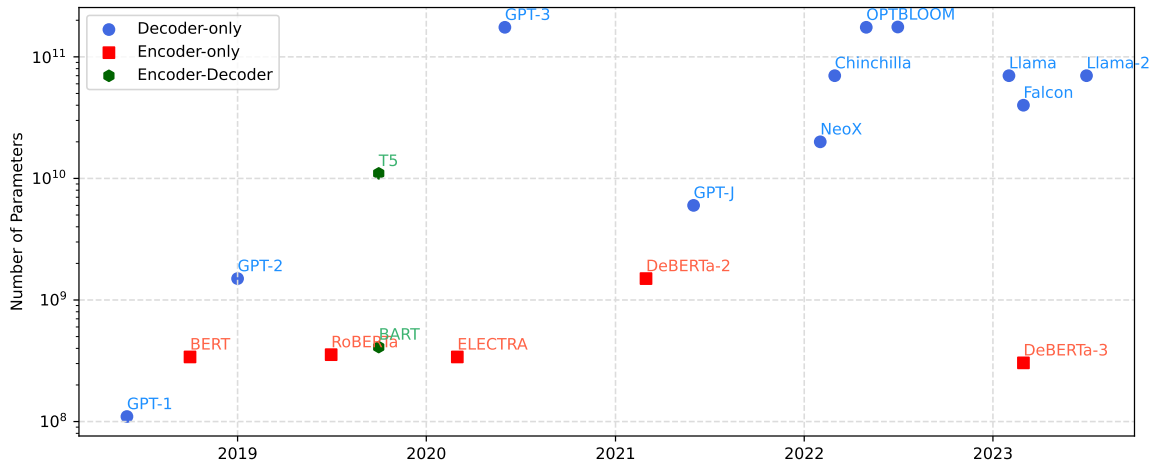


Fig. 3.3 Timeline of pre-trained transformers and the number of parameters.

Figure 3.3 illustrates the timeline of common pre-trained transformer architectures where the x-axis shows the time when each model is released based on either the timestamp of the corresponding arXiv pre-print if available or the published paper and the y-axis shows the log-scale of the number of parameters of the largest variant of each model. It is clear that

larger (and usually more capable) models have been released continuously throughout the time of conducting experiments in this thesis (late 2019 – late 2023). Figure 3.3 can be used as a rough guide about state-of-the-art backbones across the timespan.

3.1.4 Memory and Computation

Given that there are constraints in hardware (e.g., VRAM or memory of GPUs) and time, this subsection analyses the memory requirement and computational cost of transformers.

Memory Requirement

Since training is done on an accelerator (with GPU being the default option in practice), the memory requirement has to be within one GPU limit; otherwise, techniques such as model parallelism will be required, reducing training/fine-tuning speed. A standard GPU specification (during most of the time span of this research) includes:

- NVIDIA V100 with 32 GB
- NVIDIA P100 with 16 GB
- NVIDIA 2080Ti with 11 GB
- NVIDIA 1080Ti with 11 GB

$\mathcal{O}(N^2)$ complexity in memory requirement

As summarization tasks typically involve long input sequences, one of the important considerations in this thesis is the memory requirement. Following Section 2.1.4, we consider query, key, and value as matrices where

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}_1^T \\ \mathbf{q}_2^T \\ \vdots \\ \mathbf{q}_N^T \end{bmatrix} \quad \mathbf{K} = \begin{bmatrix} \mathbf{k}_1^T \\ \mathbf{k}_2^T \\ \vdots \\ \mathbf{k}_N^T \end{bmatrix} \quad \mathbf{V} = \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_N^T \end{bmatrix} \tag{3.2}$$

$\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathcal{R}^{N \times D}$ where N is the sequence length and D is the hidden dimension. The standard self-attention mechanism is,

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{Q}\mathbf{K}^T)\mathbf{V} \tag{3.3}$$

which involves the following three steps:[3]

1. 1st matrix-multiplication (Query, Key): $\mathbf{QK}^T$
2. Softmax: softmax($\mathbf{QK}^T$)
3. 2nd matrix-multiplication (Prob, Value): softmax($\mathbf{QK}^T$)$\mathbf{V}$

As the intermediate matrix $\mathbf{QK}^T \in \mathcal{R}^{N \times N}$, standard implementations (such as PyTorch implementation), which store intermediate results in the GPU high-bandwidth memory, will have a quadratic memory requirement in sequence length, (i.e., $\mathcal{O}(N^2)$). The memory requirement is more critical at training (due to gradient updates and optimizer states) than at inference (no gradient updates), which may prevent researchers without high-memory GPUs from training/fine-tuning models locally.

In recent work, FlashAttention [46] re-formulates the standard attention mechanism, leveraging the knowledge of the GPU memory hierarchy (high-memory bandwidth HRAM and on-chip SRAM) for efficiency. FlashAttention can reduce the memory requirement to linear with the sequence length $\mathcal{O}(N)$ by operating all intermediate steps on-chip without having to cache $\mathbf{QK}^T$. FlashAttention has since become the new de facto implementation of attention, but as it was released after this work, experiments in Chapter 6 do not cover FlashAttention.

Model Quantization

In standard Python, a model parameter is a 32-bit floating point (FP32) variable, which takes 4 bytes of memory. To store/load a model with 1 billion parameters, 4 GB of memory is therefore required. The 32 bits consist of 1 bit for sign, 8 bits for exponent, and 23 bits for fraction. Model quantization is to project higher precision variables into lower precision variables such that the memory requirement can be smaller but with the accuracy and stability trade-off [72, 333, 322, 164]. Table 3.1 illustrates examples of precision spaces. During the majority of the experiments in this thesis, a 32-bit floating point (full precision) is used as it is commonly adopted and is more stable; although we note that brain floating point (bfloat16) [302] has become more popular, especially with LLMs larger than 1 billion parameters.

[3] Masking, scaling, and dropout operations are omitted as they are element-wise operations with a linear complexity

Type	Bits	Exponent	Fraction	Memory
FP32	32	8	23	4 bytes
FP16	16	5	10	2 bytes
BFLOAT16	16	8	7	2 bytes
INT8	8	+/-	7	1 bytes

Table 3.1 Memory requirement for different values of precision.

Computation Cost

The computation cost of transformer-based models consists of several factors. First, as discussed in the memory requirement, computing standard self-attention is quadratic in the sequence length (i.e., $\mathcal{O}(N^2)$) due to $\mathbf{QK}^T$. In addition, a non-trivial amount of computation also comes from the two-layer feed-forward layers at every block (approximately half the compute time and/or FLOPs [283]). Although the complexity of the FFN is linear with respect to sequence length (i.e., $\mathcal{O}(N)$), it is generally still costly due to the size of hidden dimension D.

3.1.5 Efficient Transformers

Pre-trained transformer models have shown success [52, 159, 244] and become the starting point for various NLP problems such as BERT [52] in contextual representation, GPT-2 in text generation [242], or BART in sequence-to-sequence tasks [159]. However, memory and time requirements of standard implementations (in Section 3.1.4) for transformer models grow quadratically with the sequence length, which can hinder model scalability in many settings, especially for long-span tasks. The term 'efficient model', in this work, refers to the efficiency of the Transformer model both in terms of memory and computation. There have been a number of transformer variants proposed that address the quadratic complexity problem [283, 282], which can be categorized as follows:

• **Fixed Attention Patterns**: Blockwise attention [238] groups the sequence into fixed blocks (each with length W) such that the complexity reduces from N^2 to $W^2 \times \frac{N}{W} = WN$. Furthermore, Transformer-XL [42] extends the blockwise attention method by connecting blocks via recurrence. Similar to blockwise attention, local attention (shown in Figure 3.4b) [223, 338, 10] operates by limiting each token to attend to neighbouring tokens within a span size of W. Stride attention (shown in Figure 3.4c) [27, 10] employs strided or dilated windows such that each token can only attend to other tokens at a fixed interval. Global

attention (shown in Figure 3.4d) allows pre-defined tokens such as <bos> and <eos> to attend to over other positions as well as being attended by all other positions. A combination of fixed patterns has been considered. For example, Longformer [10] employs both local, strided, and global attention patterns. BigBird [338] employs random attention (a fixed number of random positions are attended to) in addition to local and global attention patterns.

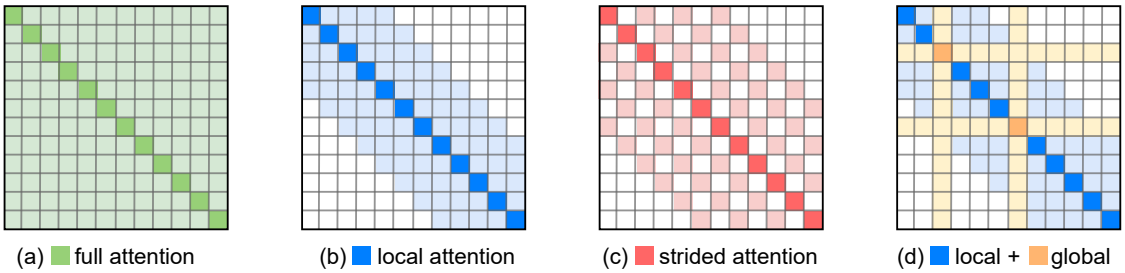


Fig. 3.4 Diagram of different forms of fixed attention patterns.

• **Learnable Patterns**: Instead of pre-defining fixed patterns, learnable patterns aim to find an effective pattern from the data, e.g., through token relevance and group tokens into clusters, such as Reformer [137]. Also, the Sinkhorn Sorting Network [281] exposes the sparsity in attention weights by learning to sort blocks of the input sequence.

• **Low-Rank Matrix Approximation**: This method assumes the low-rank structure in the $N \times N$ matrices. For example, Linformer [303] projects the length dimension of the key and value matrices to a lower-dimensional space ($N \rightarrow k$), reducing the complexity to $N \times k$. Recently, LoRA [116], a low-rank matrix approximation-based method, has made training LLMs faster and more memory-efficient. LoRA works by adding an adapter (e.g., a low-rank $N \times r$ matrices) and training only this adapter.

• **Kernel Methods**. These methods are approximations that use kernel tricks to avoid the explicit computation of the $N \times N$ matrix. These methods include Performers [30], Linear Transformers [131], etc.

A detailed survey on efficient transformers has recently been compiled by Tay et al. [283]. In addition to efficient architectures, there are alternative and complementary techniques which can be used to improve efficiency. For example, Voita et al. [294], Michel et al. [196] showed that some attention heads are redundant and can be pruned to reduce model size. Knowledge distillation [106, 255] compresses a large model (i.e., teacher) to a smaller model (i.e., student) while retaining most of the performance and reducing memory and computation.

Model quantization [121] has also been applied to compress the model from full precision (32-bit) to half-precision (16-bit) or lower.

3.2 Properties

To leverage foundation language models for a particular task, it is essential to understand their properties and limitations. First, this section will discuss the memory and computation requirements of the transformer architecture. Next, this section discusses scaling laws and emergent abilities which enable large language models to perform various NLP tasks in a zero-shot fashion. Lastly, this section discusses the methods to align LLMs to user intents, which has recently driven the mass adoption of LLMs.

3.2.1 Scaling Laws and Emergent Abilities

Language models (LMs) are probabilistic models that learn patterns in texts, and they can be used to compute the likelihood of words or to generate new words. More recently, GPT-3 has shown that as LMs become large (hence referred to as large language models or LLMs), they can be used in a few-shot learning manner [15]. Few-shot learning refers to when a model can make predictions or understand new tasks it has not seen before, using only a few examples for each task. This contrasts with traditional approaches that often require large amounts of data for accurate performance. In the context of LLMs, few-shot learning is usually performed through a prompting method such as in-context learning [201]. Furthermore, recent work such as Kojima et al. [141] shows that LLMs are zero-shot learners. This is when a model can infer information about new tasks it has never seen. LLMs can achieve impressive results *without* the need for task-specific fine-tuning or architectural modification. As LLMs are scaled up, new abilities are unlocked or *emerge*[4] without being directly trained on; for example, the emergent abilities include performing arithmetic, question answering, summarization, etc [15, 309, 31].

In-Context Learning

After pre-training and/or aligning LLMs to better follow human instructions (Section 3.2.2), a standard approach to using LLMs is to design suitable prompts for solving various tasks. A typical prompting method to utilize relevant examples, known as in-context learning (ICL), formulates the task description and/or demonstrations (examples) in the form of

[4]Note that recent work by Schaeffer et al. [258] argues that emergent abilities appear because of the choice of evaluation metrics that are non-linear.

natural language text [55]. It has been shown that LLMs can solve a series of complex tasks through in-context learning, such as solving mathematical reasoning problems [310]. In-context learning requires a few examples to form a demonstration context concatenated with a query in natural language as the input prompt to the LLM. Unlike supervised learning, in-context learning does not change the model parameters. This makes in-context learning interpretable, highly adaptable, and easier to incorporate human knowledge into LLMs. Since in-context learning is applied through natural language interface prompting, some more advanced prompting techniques have been proposed such as chain-of-thought (COT) [310], self-consistency COT [304], and three of thoughts [332].

Scaling Up LLMs

Scaling language model has shown consistent improvements in model performance (e.g., as measured by the cross-entropy loss). Kaplan et al. [128] showed that a linear relationship between the loss and log of the number of parameters holds from 10^3 to 10^9 parameters. In contrast, *emergent abilities* refer to the scenario when the LLM suddenly gains an ability to perform a task, it is on *not* trained on, once it reaches a certain scale. For example Srivastava et al. [275] showed that after reaching the size around 10^{10} to 10^{11} parameters, the LLM is suddenly capable of performing several tasks on BIG-bench [275].

In addition, recent works [110, 286] demonstrated that scaling up the model size solely may not be enough as training also depends on the dataset size. For example, Hoffmann et al. [110] showed that the number of tokens (in the training data) should be about 20 times the number of parameters to be data optimal for a fixed computational budget. Similarly, Touvron et al. [286] showed that previous training was not data optimal, and by training longer (e.g., larger training dataset), LLaMA achieves state-of-the-art performance and is comparable to an LLM with 10 times more parameters.

In conclusion, it has been found that scaling up LLMs results in emergent abilities which enable zero-shot or few-shot approaches. Another important technique to improve LLMs is instruction tuning, which is to further align LLMs with human instructions. Aligning LLMs is important as it improves the zero-shot ability of LLMs (e.g., without the need for providing examples as in few-shot learning). Both of these are discussed in more detail below.

3.2.2 Aligning Large Language Models

Despite being trained on a vast quantity of texts from the internet, LLMs do not necessarily understand human instructions, or generate unintended responses, limiting the usefulness of LLMs. Thus, aligning LLMs with human preferences has become an important component in

the success of modern LLMs in performing tasks in a zero-shot manner [306]. The following will outline two common techniques for aligning LLMs to human instructions.

Supervised Fine-Tuning

Supervised Fine-Tuning (SFT) is a method to instruction-tune LLMs using supervised instruction-response pairs. SFT is a technique to improve the capabilities and controllability of foundation models, especially large language models (LLMs) [32, 218, 341]. The goal is to make LLMs understand and respond to human instruction, which is crucial in zero-shot applications. SFT is performed by further training LLMs using pairs of instruction and desired output. Since LLMs are trained to predict the next words, they are capable of performing sentence completion but they may lack the ability to follow human instructions, which limits the helpfulness of LLMs. InstructGPT [218], for example, is an instruction fine-tuned variant of GPT-3 [15], allowing the LLM to give useful responses to user prompts.

An instruction-tuning dataset consists of multiple common supervised tasks such as question answering, summarization, and dialogue generation. Each example in an instruction-tuning dataset consists of three elements: (1) an instruction which specifies the task in human language, (2) an (optional) input which provides (optional) context for performing the task, (e.g., the source document for performing summarization), and (3) the desired output.

The dataset is generally constructed by methods as follows. First, existing NLP datasets are turned into instruction-tuning datasets using a template as done in, for example, T0 [256], Flan [176] and P3 [256]. T0 converts a large set of existing supervised datasets, each with multiple prompts and different wording. FLAN manually creates ten prompt templates, where each template contains natural language instructions to describe the tasks associated with the dataset. This includes 62 publicly available datasets from Tensorflow Datasets. Second, desired outputs are crafted by human annotators such as the dataset for InstructGPT [218] or by prompting (usually larger) LLMs such as the Alpaca dataset [280].

Reinforcement Learning from Human Feedback (RLHF)

Reinforcement Learning from Human Feedback (RLHF) [218] is a training methodology for training machine learning models, particularly deep reinforcement learning models, using feedback derived from human annotators. Traditional reinforcement learning (RL) involves a model (e.g., an LLM) interacting with an environment and learning through rewards based on its actions. However, designing a good reward function can be challenging, especially in complex environments such as natural language tasks. This is where human feedback can improve RL. The process of RLHF typically follows these steps:

1. **Human Annotation**: (1) desired responses are manually generated, which are then used to fine-tune an LLM (supervised training); (2) ranking responses are performed by human annotators, which are then used in the next step.

2. **Reward Model**: A reward model can be trained to rank responses using the dataset from the previous step. This reward model can then be used instead of a pre-defined hand-crafted reward function in the RL training.

3. **Training with Proximal Policy Optimization (PPO)**: Using the learned reward model as a surrogate reward function, the LLM is then fine-tuned using a reinforcement learning algorithm such as Proximal Policy Optimization (PPO) [260]. Recently, Direct Preference Optimization (DPO) [243] is proposed to implicitly optimize an objective that yields a similar result to RLHF without the need for reward model fitting, extensive sampling, and complex hyperparameter tuning in reinforcement learning.

The initial model can also be further improved by collecting more human feedback, and the RLHF process above can be iterated over again.

3.3 Chapter Summary

This chapter discussed foundation models, which are the backbone of many current deep learning systems. Section 3.1 discussed self-supervised learning (SSL) which exploits inherent structures in unlabelled data to pre-train foundation models for further fine-tuning to specific tasks. This section discussed the general pipeline in SSL, the application of SSL in NLP, major types of pre-trained transformers, memory and computation complexity, and efficient transformer architectures. Section 3.2 discussed the properties of transformers, including scaling laws, emergent abilities, and alignment processes. This chapter provides the fundamentals of foundation models, which will used for abstractive summarization in Chapter 6, for summary assessment in Chapter 7, and information consistency in generative AI in Chapter 8.

Chapter 4

Summarization Background

Summarization is the task of producing a shorter version of one or more source documents that preserve most of the source's information. It is one of the most researched areas in the Natural Language Processing (NLP) community with several benchmark approaches [212, 331] and tasks [208, 209]. A taxonomy for automatic text summarization can be done based various classifications; for example, the number of source documents (e.g., single-document, multiple-document), summarization method (e.g., extractive, abstractive), the nature of the summary (e.g., query-based, generic), language (e.g., monolingual, multilingual, cross-lingual), domain (e.g., news, podcasts, meeting transcripts, scientific articles, opinions/reviews, etc.).

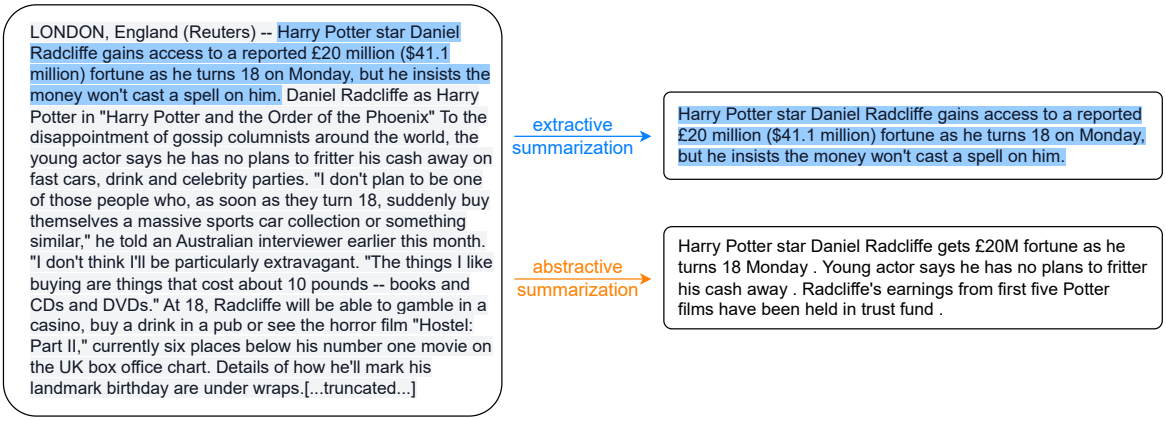


Fig. 4.1 An example of extractive summarization and abstractive summarization.

As the focus of this work is on the methods underlying summarization techniques, the main classification in this chapter will be *extractive* and *abstractive* summarization illustrated in Figure 4.1. Extractive summarization aims to select the most salient units from the source document and reorder the extracted units to form a summary. Extracted units can be sentences,

segments, or words. One way of achieving extractive summarization is to assign a score to each and every extractive unit, and then select those with high scores. Alternatively using a probabilistic view, extractive summarization can be seen as the probability of each extractive unit being included in the summary. Extractive summarization will be further described in Section 4.1. On the other hand, abstractive summarization aims to generate words and phrases that may or may not appear in the source document. From a probabilistic perspective, abstractive summarization is to learn the conditional probability of the summary given the source document. Abstractive summarization will be further described in Section 4.2.

A good summary is usually ill-defined and subjective because there are multiple aspects such as content and text quality. A good summary should contain salient information in the source as well as be readable. Despite the subjectivity, accurate and reliable summary evaluation methods are useful, as they provide a way to compare new summarization models to existing ones. Thus, various assessment/evaluation methods and metrics have been proposed from human evaluation to automatic assessment methods. Human evaluation is usually treated as the gold-standard approach. However, human evaluation can be expensive, time-consuming, and difficult to scale, therefore, limiting its usefulness. Automatic assessment methods aim to approximate gold-standard human evaluation. These automatic methods are evaluated by similar they are compared to gold-standard or human evaluation. The evaluation of automatic methods is referred to as *meta*-evaluation, or simply just evaluation in the context of developing automatic assessment methods. Given the importance, this chapter includes the background of automatic summary assessment methods in Section 4.3.

4.1 Extractive Summarization

Extractive summarization is to generate a summary by selecting salient units from the original document, with or without reordering them. Extractive units do not necessarily mean sentences in the linguistic sense. Extractive summarization units can be sequences of words such as word segments, sentences, or utterances in spoken language. For simplicity, in this chapter, we will refer to all types of extractive units as *sentences*. Extractive summarization methods are categorized into *unsupervised* methods and *supervised* methods, and we will discuss commonly used methods for each of the two types.

4.1.1 Unsupervised Extractive Summarization

Unsupervised extractive summarization methods are the first approaches applied to text summarization. Extractive summarization methods do not require document-summary pairs,

and they can be computationally cheap to run. Hence, they are often used as baselines. The notation for extractive summarization is as follows: X is the source document consisting of N extractive units, such as sentences, $\{S_1, S_2, ..., S_N\}$. The following will describe common extractive summarization methods.

TextRank and LexRank

TextRank [198] and LexRank [64] are similar, both algorithms are based on the PageRank algorithm. The first step is to split the source document X into sentences, $S_1, S_2, ..., S_N$. Similarity measures between sentences are then calculated and stored in the form of a matrix $\mathbf{M}$ where the element (i, j) is:

$$M_{i,j} = \frac{1}{Z_i}\texttt{sim}(S_i, S_j) \tag{4.1}$$

TextRank and LexRank differ in how the similarity score $\texttt{sim}(.)$ is computed. TextRank computes the number of words that the two sentences have in common divided by the sentence length, whereas LexRank computes the cosine similarity of TF-IDF vectors. The term Z_i is a normalization term of each column of $\mathbf{M}$ such that $\sum_i M_{i,j} = 1.0$, and $\mathbf{M}$ is a transition matrix where $M_{i,j}$ is the probability of $j \rightarrow i$. Both algorithms seek to find a vector $\mathbf{v}$ such that

$$\mathbf{Mv} = \mathbf{v} \tag{4.2}$$

In other words, if we consider the problem as a graph, each sentence as a node, and the similarity score as an edge, we have a process of information flow between sentences. Therefore, $\mathbf{v} = [v_1, v_2, ..., v_N]^T$ is the stationary distribution, or the eigenvector associated eigenvalue of 1. Finally, sentences can be ranked according to the scores $v_1, v_2, ..., v_N$.

Maximum Marginal Relevance (MMR)

MMR is an iterative method that can be used in extractive summarization [18]. Let Y^k be the hypothesis summary at the k-th iteration. The score of S_n in the k-th iteration is defined as:

$$\text{score}(S_n) = \lambda \times \texttt{sim}_1(S_n, X) + (1 - \lambda) \times \texttt{sim}_2(S_n, Y^{k-1}) \tag{4.3}$$

The first term can be thought of as a saliency measure, and the second term can be thought of as a redundancy measure. λ is a hyperparameter weighing the importance of the two measures. The sentence of the highest score in iteration k is selected and added to Y^{k-1} to yield Y^k. The similarity functions $\texttt{sim}_1(.)$ and $\texttt{sim}_2(.)$ can be defined differently as denoted by different subscripts. Traditionally, S_n, X, and Y^k are encoded using TF-IDF, and cosine similarity is used as the similarity functions.

Latent Semantic Analysis (LSA)

LSA is a method based on singular value decomposition (SVD). Let $t_1, t_2, ..., t_{|V|}$ represent frequencies where t_j is the frequency (count per total) of word j in a given text. A matrix $\mathbf{A}$ can be constructed such that $A_{i,j}$ is the frequency of word j in sentence i. Row i of matrix $\mathbf{A}$ can be thought of as a vector representing sentence i. We can compress $\mathbf{A}$ using SVD:

$$\mathbf{A}_{\{N,|V|\}} = \mathbf{U}_{\{N,K\}}\mathbf{S}_{\{K,K\}}\mathbf{V}^T_{\{K,|V|\}} \tag{4.4}$$

where K is the number of compressed dimensions (also called the number of topics). It can be seen that row i of the matrix $\mathbf{U}$ is a compressed vector representation of sentence i. This enables using the compressed representations to compute similarity. There are other variants of LSA such as probabilistic LSA (PLSA) and latent Dirichlet allocation (LDA). Once compressed representations are obtained, methods such as TextRank or MMR can be applied. Other unsupervised extractive summarization methods are, for example, Integer Linear Programming (ILP) based methods [194, 87].

4.1.2 Supervised Extractive Summarization

In Section 4.1.1, unsupervised methods rely on defining scores for estimating the saliency of each input sentence, respective to other input sentences. In contrast, supervised extractive summarization aims to identify salient sentences from ground-truth signals.

Early supervised methods construct input features from texts, and they adopt techniques such as support vector machine (SVM) and Naive Bayes Classier to make predictions about sentences [63]. For example, in Kupiec et al. [148] a binary classifier is trained using the Bayes' rule to calculate the probability of including each sentence, in Conroy and O'leary [41] Hidden Markov Model (HMM) is used to compute the likelihood of each sentence, in Fuentes et al. [75] SVMs are trained for query-based summarization to select relevant sentences. Given the success of deep learning, neural networks have been applied to extractive summarization and extractive summarization is formulated as a sequence tagging task.

Neural Sequence Classification

This section focuses on modern approaches which are based on neural sequence classification where extractive summarization is formulated as a sequence classification task. Various neural-based approaches have been proposed, for example, by Cheng and Lapata [25], Nallapati et al. [207], Xu and Durrett [326], Desai et al. [48].

The notation is the same as unsupervised extractive summarization where $X = \{S_1, S_2, ..., S_N\}$ is the source document with an addition of $z_i \in \{0, 1\}$ be a binary label indicating whether sentence S_i is salient or not. The task is predict to predict $\hat{z}_i$, i.e. assign how likely sentence i is salient or not,

$$\hat{z}_1, \hat{z}_2, ..., \hat{z}_N = f(\{S_1, S_2, ..., S_N\}) \tag{4.5}$$

where $\hat{z}_i$ is a continuous value bounded between $[0.0, 1.0]$. Using a neural model, Equation 4.5 can be modelled by a probabilistic model:

$$\hat{z}_i = P(z_i | \{S_1, S_2, ..., S_N\}; \boldsymbol{\theta}) \tag{4.6}$$

where early work adopted RNN, LSTM, and/or CNN architecture for $\boldsymbol{\theta}$ [25, 207] and the trend later shifted to using transformers [353, 3]. Nevertheless, these neural models similarly encode sentence S_i into a representation,

$$\mathbf{h}_i = f(\{S_1, S_2, ..., S_N\}; \boldsymbol{\theta}) \tag{4.7}$$

This is usually followed by a sigmoid activation at the output layer,

$$\hat{z}_i = \text{sigmoid}(\mathbf{w}^T \mathbf{h}_i + b) \tag{4.8}$$

where $\mathbf{w}$ and b are model parameters of the output layer. The objective function for supervised extractive summarization in this setup is binary cross entropy,

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{i=1}^{N} (z_i \log \hat{z}_i + (1 - z_i) \log(1 - \hat{z}_i)) \tag{4.9}$$

and the loss function can be optimized using a stochastic gradient descent approach. Neural approaches typically do not use hand-engineered features as the input texts are encoded into embeddings. The neural approaches have shown success compared to the previous approaches. Using a pre-trained backbone, the neural approach has achieved state-of-the-art performance in extractive summarization [169, 327].

Training extractive models require reference labels $\{z_1, ..., z_N\}$. However, the majority of reference summaries are often in the form of text-based summaries. To obtain reference labels, Cheng and Lapata [25], Nallapati et al. [207], Liu and Lapata [169] use greedy selection to create reference labels based on heuristics. For instance, one sentence at a time is selected incrementally to a set of already selected sentences such that it maximizes the ROUGE score [162], described in Section 4.3.1, between the current set of selected sentences and the

reference summary. To avoid creating pseudo extractive labels, Reinforced Neural Extractive Summarization (RNES) [320] directly trains the model using reinforcement learning loss with the reward comprising coherency and ROUGE score.

Alternatively, in Zhong et al. [352], they formulate the problem as a semantic text matching problem, in which a source document and candidate summaries (extracted from the original text) are matched in a semantic space.

4.2 Abstractive Summarization

Abstractive summarization generates words and phrases. These words and phrases capture the salient information in the source document, but the exact words may or may not appear in the source document. Abstractive summarization approaches encode the source document into feature representation and generate the summary based on the representation. As abstractive summarization requires more complex abilities such as paraphrasing, generalization, or incorporating domain knowledge, it was difficult to achieve these abilities without deep neural networks. With the success of sequence-to-sequence models [279], RNN-based abstractive summarization models using a sequence-to-sequence architecture were proposed [252, 29, 208].[1] Similarly to other natural language generation tasks, abstractive summarization models were traditionally trained from scratch and an inductive bias was carefully designed into the model architecture [252, 29]. Given the success of self-supervised learning, pre-trained language models have become the starting point for developing abstractive summarization models [159]. More recently, large language models with billions of parameters have enabled abstractive summarization in a zero-shot manner [15]. This section will delve into the details of each of these paradigms of abstractive summarization.

4.2.1 Training Models from Scratch

An abstractive summarization model consists of an encoder which processes the source document $X = x_{1:N}$, and a decoder that processes/generates the summary $Y = y_{1:M}$. The encoder can have bi-directional dependencies, while the decoder has uni-directional dependencies from left to right because, at the inference time, it has to perform autoregressive generation. Essentially, these models are a conditional language model $P(Y|X;\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$. The language model is then trained on the maximum likelihood criterion described in Section 2.2.

[1] The RNN-based sequence-to-sequence architecture is described in Section 2.1.3

Based on this conditional language modelling and sequence-to-sequence structure, early work focused on designing neural network architectures. For example, Rush et al. [252] experimented with a bag-of-words model, CNN, and attention network for the encoder while a feed-forward network was used for the decoder. Chopra et al. [29] adopted a CNN model or attention-based network for the encoder, while the decoder was RNN or LSTM. Nallapati et al. [208] improved the architecture by using a bidirectional GRU for the encoder with several input features including word embedding, part-of-speech tags, named-entity tags, and TF-IDF features. The encoder is also hierarchical with one GRU layer at the word level and one GRU layer at the sentence layer. The decoder uses a uni-directional GRU and a pointer-network [292, 96] which allows the decoder to copy words from the input.

Pointer Generator Network (PGN)

A notable abstractive summarization model is the **Pointer Generator Network (PGN)** [263] illustrated in Figure 4.2, and this model will be used as a baseline in Chapter 5. The PGN model extends the vanilla encoder-decoder model by adding a copying mechanism. The copying mechanism allows PGN to mitigate *factuality* and *out-of-vocabulary* problems. In addition, it employs a coverage mechanism [288] to mitigate repetition in generation.

Following the vanilla encoder-decoder with an attention mechanism described in Section 2.1.3, assume $\mathbf{c}_m$ is the context vector and $\mathbf{d}_m$ is the decode state at time instance m, the PGN model computes the probability of generating from the vocabulary as follows:

$$p_{\text{gen}} = \text{sigmoid}(\mathbf{w}_{\text{gen}}^T[\mathbf{d}_m; \mathbf{c}_m] + b_{\text{gen}}) \tag{4.10}$$

where $\mathbf{w}_{\text{gen}}$ and b_{gen} are pointing mechanism trainable parameters. The output probability distribution given *generation* is the standard output distribution of words in the vocabulary (i.e., softmax of the decoder output layer) in Equation 2.14. The output probability distribution given *copying* is the summation of the attention mechanism score for each word in the source text, $\alpha_{t,i}$. The overall output probability of PGN is:

$$P(y_m|y_{1:m-1}, X) = p_{\text{gen}} P(y_m|, y_{1:m-1}, X; \text{gen}) + p_{\text{copy}} P(y_m|y_{1:m-1}, X; \text{copy}) \tag{4.11}$$

$$= p_{\text{gen}} P_{\text{vocab}}(y_m) + (1 - p_{\text{gen}}) \left(\sum_{i:w_i = w_m} \alpha_{m,i} \right) \tag{4.12}$$

where $P_{\text{vocab}}(y_m)$ is the output distribution as defined in Equation 2.14.

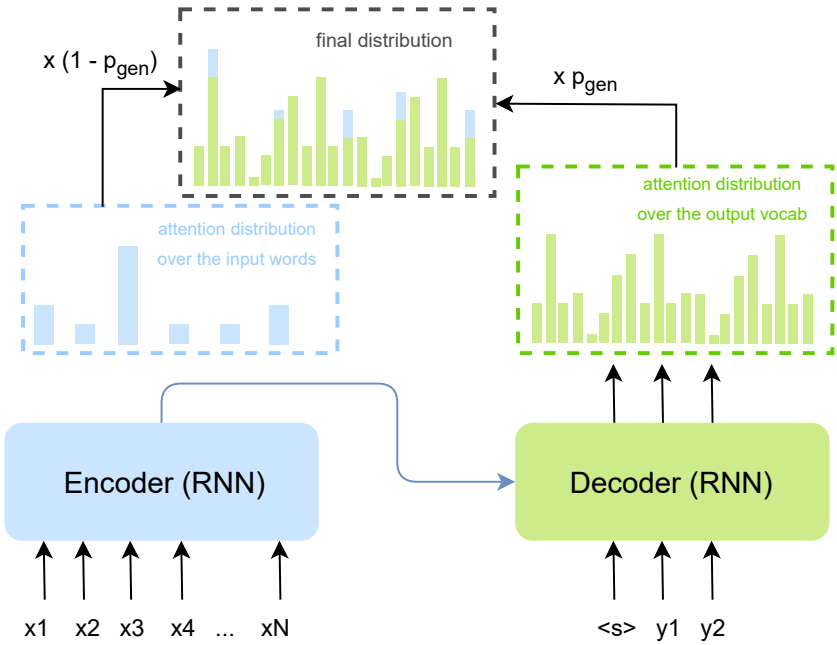


Fig. 4.2 Pointer Generator Network (PGN) Model Architecture. The output distribution comes from two parts: (1) generation from the vocabulary, and (2) copying from the source sequence.

Furthermore, Paulus et al. [224] augment the token-level MLE training loss with a sequence-level RL loss:

$$\mathcal{L}_{\text{RL}} = (R(Y^g) - R(Y^s)) \log P(Y^s|X;\boldsymbol{\theta}) \tag{4.13}$$

where $R(.)$ is a metric such as ROUGE, Y^g is a greedy-search output, and Y^s is a sampled output. More details about sequence-level criteria are discussed in Section 2.2.2. Furthermore, two-stage approaches are considered by Chen and Bansal [24], Gehrmann et al. [82], Hsu et al. [114] where they demonstrate that good content selection (i.e., via extractive summarization) helps abstractive news summarization systems. Gidiotis and Tsoumakas [86] divide the source and target into multiple smaller pairs before training abstractive summarizers.

Hierarchical Recurrent Neural Network

Hierarchical recurrent neural networks were designed to model long-term dependencies [105]. Given their effectiveness for long-term dependencies, they could be used as an encoder for long-input sequence-to-sequence tasks. In summarization tasks, Cohan et al. [39], for example, applied a hierarchical attention model to long-form documents (e.g., research papers) by having two RNN layers at the word level and the section level. In meeting summarization, Li et al. [160] proposed a multi-modal hierarchical attention encoder across three levels: segment, sentence/utterance and word where topic segmentation and

abstractive summarization are jointly modelled. Similarly, Zhao et al. [349] proposed a hierarchical attention encoder across three levels with a gating mechanism, which computes the representation of the current utterance as a non-linear combination of the representation of the previous utterance and the word-level representations of the current utterance. In this thesis, we follow the existing hierarchical model architecture, and we describe the architecture in detail in Section 5.2.1.

4.2.2 Fine-tuning Pre-trained Models

Pre-trained language models such as BERT [52] have been applied to summarization. Early work includes Liu and Lapata [169] where the transformer-based encoder was initialized with BERT's parameters while the transformer-based decoder was initialized from scratch. The encoder-decoder model was jointly trained on summarization datasets, achieving state-of-the-art at the time. Recent work has focused on designing pre-training self-supervised objectives to be closer to the target tasks, and notable pre-trained models are described below.

• **BART** [159] was proposed for sequence-to-sequence tasks. BART (illustrated in Figure 4.3) extends BERT and GPT to sequence-to-sequence tasks. BART has an encoder-decoder architecture, which allows it to perform denoising processes such as swapping word order and predicting masked words. Its decoder is to denoise, (i.e., reconstruct noise added inputs) and predict the next words. BART achieved state-of-the-art results on a range of sequence-to-sequence tasks such as abstractive dialogue, question answering, and summarization.

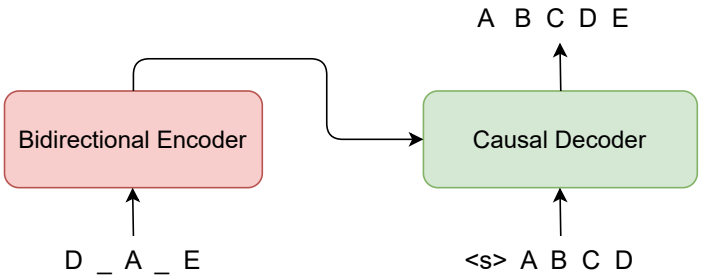


Fig. 4.3 BART combines the benefit of BERT (to encoder source sequence) and GPT-2 (to generate output sequence). Note that this also allows BART to be pre-trained using arbitrary noise transformation. This figure is adapted from the original BART paper.

• **UniLM** [54] combined unidirectional, bidirectional, and sequence-to-sequence pre-training objectives so that one unified transformer model can be used in any downstream task (with an appropriate mask).

• **T5** [244] is another sequence-to-sequence pre-trained where a large number of NLP datasets are formatted as a sequence-to-sequence task for a unified text-to-text framework. For instance, an example in a German-to-English translation dataset could be formatted as `Translate German to English: Das ist gut` as used as the input to T5. This unified text-to-text framework allows T5 to generalize and perform well. This process of formatting NLP datasets into a unified format is similar to supervised fine-tuning LLMs with formatted data described in Section 3.2.2.

• **PEGASUS** [340] was proposed to further make pre-training closer to summarization more than BART. PEGASUS (illustrated in Figure 4.4) masks entire sentences (referred to as Gap Sentence Generation, GSG) in addition to masking tokens. The model is trained to predict these sentences. Although the GSG objective is highly challenging, even for humans, this criterion was shown to elicit a higher sense of understanding for the generation of sentences that have an instance of the original document. The findings from PEGASUS also demonstrated that choosing the most important sentences for masking yields the best results, and this is done by finding sentences that are the most similar to the document according to the ROUGE score.

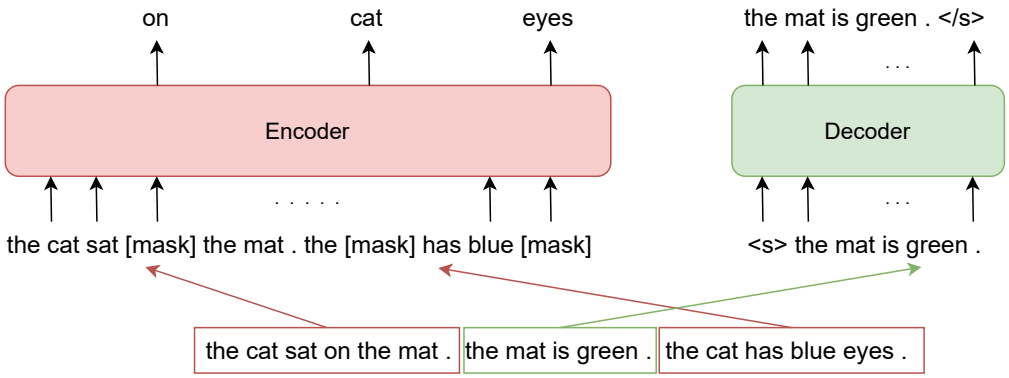


Fig. 4.4 PEGASUS performs sentence masking and word masking simultaneously. For example, the second sentence is masked and it is used in the text. The other two sentences remain in the input but have some tokens masked.

Furthermore, Liu and Liu [172] proposed a re-ranking approach where the first stage uses BART to generate a list of candidate summaries, and the second is to score the generated summaries to obtain the best summary. The scoring model (using RoBERTa as the backbone) is trained on a contrastive loss.

Long-input Summarization

As described in Section 3.2.2, the compute and memory costs of the standard transformer architecture grow quadratically with sequence length. In long-input summarization, this quadratic dependency leads to a high memory requirement that could make fine-tuning a large pre-trained model infeasible. Thus, applying a large pre-trained transformer model off-the-shelf to long-input summarization can be challenging.

• **Efficient Transformer for Summarization**: One method to address the long-span problem is to make the model architecture more *efficient* to handle a longer input span. For example, efficient transformer architectures, which have been applied to long-input summarization, are BigBird [338] and Longformer-Encoder-Decoder (LED) [10] which was developed concurrently with the work in Chapter 6. BigBird is an encoder-decoder model where its encoder employs local, global, and random attention patterns. Similarly, LED extends Longformer to an encoder-decoder model, and its encoder has local, strided, and global attention patterns. Although long-input summarization is a standard long-input task that efficient models are designed to address, such models are applicable to broader sequential tasks, and the discussion of these models is included in the section about efficient transformers (Section 3.1.5).

• **Hierarchical Architecture**: Hierarchical architectures (such as the one described in detail in Section 5.2.1) have also been applied to multi-document summarization [168], and extractive news and table-to-text summarization [345, 210]. A hierarchical RNN system has been applied to summarize long articles [39].

• **Two-stage: Extractive + Abstractive**: Existing work shows that good content selection helps abstractive news summarization systems [24, 82, 114]. Hybrid systems that select sentences and generate an abstractive summary have been proposed such as extractive system + TLM (Ext+TLM) for scientific articles [231], simple selection + BART for podcasts [274], and guided summarization by BERT-based keyword/sentence extraction + BART for news and scientific articles [101, 59]. CTRLsum [101] extends BART by conditioning the summary generation with extracted keywords V in addition to the document, e.g., $P(Y|X,V)$. The keyword extraction model is based on BERT, and they apply a sliding window allowing it to extract V in long sequences. However, their BART-based abstractive summarizer is still limited to the first 1,024 tokens. Other work includes dividing the source and target into multiple smaller pairs to train abstractive summarizers [86]. Extractive methods with and without redundancy reduction techniques for long-span summarization have been studied by Xiao and Carenini [323, 324].

4.2.3 Prompting Large Language Models

With the advances in large language models (LLMs), LLMs have become ubiquitous, powering many NLP applications including text summarization [91, 344]. LLMs have led to a shift in paradigm from fine-tuning language models to prompting language models to perform a new task in a few-shot or zero-shot manner [15, 218]. To make language models understand natural language prompts and perform tasks such as text summarization, two steps are required: (1) pre-training at scale to achieve emergent ability (outlined in Section 3.2.1), and (2) aligning the model with instruction-tuning datasets or with reinforcement learning with human feedback (outlined in Section 3.2.2). In the **zero-shot setup**, an instruction-tuned language model can be prompted with a user instruction as well as the desired output style defined in the prompt [218], for example,

```
Summarize the content you are provided with for a second-grade student.
Content: ...
```

Prompt-based methods have improved controllable text summarization through textual input in the form of a set of keywords or descriptive prompts [101, 124]. However, it should be noted that zero-shot LLMs have been shown to underperform supervised models and in-context learning on more challenging tasks such as machine translation. For example, Brown et al. [15] showed that the few-shot setup for machine translation increased the BLEU score for GPT-3 by almost 11 points from the zero-shot setup.

4.3 Summary Assessment

Summary assessment (i.e., summary evaluation) is useful for both machine-generated and human-produced summaries. Assessing the generated summary text enables, for example, summary generation system development and detection of inappropriate summaries to ensure the quality of training examples. Reliable summary assessment methods are crucial to benchmark new summarization models as they can show progress in summarization development. While human evaluations are seen as the gold standard, they are costly and hard to scale. Therefore, various automatic assessment methods have been proposed to emulate human standards. This section will describe existing summary assessment methods, which serve as the background for automatic summary assessment work in Chapter 7.

Notation

$X = x_{1:N}$ is the source document, $Y = y_{1:M}$ is a candidate summary (i.e., summary to be assessed), $Y^* = y^*_{1:M}$ is the reference/gold-standard summary which may or may not be available depending on the task, z is the quality of the summary Y.

Aspects of Summary Assessment

Assessment methods need to evaluate the quality of a summary on one or more aspects which are typically defined as follows [213, 45, 43, 125, 17]:

- **Text Quality**
  - **Fluency**: Assessing the quality of individual sentences (e.g., whether the summary is grammatical, or the summary appears natural).
  - **Coherency**: Assessing the collective quality of all sentences (e.g., how well sentences are connected).
- **Content**
  - **Informativeness** (or **Relevance**): Assessing how much salient information is presented in the summary, and how much redundancy the summary contains.
  - **Consistency** (or **Faithfulness**, **Factuality**): Assessing whether the information in the summary can be inferred by the source document. Hallucinated contents in an unfaithful summary can be categorized into (1) *intrinsic* hallucination = when information is manipulated inaccurately; and (2) *extrinsic* hallucination = when information is added. Maynez et al. [193] distinguished between faithfulness and factuality in whether world knowledge[2] is taken into account in the context.

A combination of the aspects has been used as the overall quality of a summary [45, 43, 125]. It should be noted that the list above should not be treated as complete as different categorizations and definitions have been adopted in the community [113]. Given that modern summarization systems (especially those which are based on pre-trained models) are capable of generating highly fluent summaries, the text quality aspects (e.g., fluency and coherence) are less important [229, 50, 49]. As a result, the focus of the majority of recent summary evaluation research has been on content-based aspects such as information consistency. Similarly, this thesis focuses on assessing content-based quality, and we will therefore

[2]World knowledge is defined as the knowledge from external sources.

discuss and present existing content-based assessment methods. Note that the details about text-quality methods can be found in survey papers by El-Kassas et al. [62], Mridha et al. [203], Cajueiro et al. [17].

Categorization of Summary Assessment Methods

A number of summary assessment methods have been proposed in the past decades and multiple ways of categorization exist [17]. For example, categorization can be based on whether the methods are *intrinsic* or *extrinsic* where intrinsic methods measure the quality of the summary against the source document, while extrinsic methods measure the performance on a target task when using the summary (e.g., document classification or reading comprehension). Other ways of categorization include the aspect that measures a summary, the algorithms (e.g., supervised or unsupervised), and model-based or model-free, etc.

In this thesis, as the focus is on content-based summary assessment, the methods are categorized into (1) reference-based where the candidate summary is compared against the gold-standard summary (or multiple gold-standard summaries), and (2) reference-free where the generated summary is compared against the source document as illustrated in Figure 4.5.

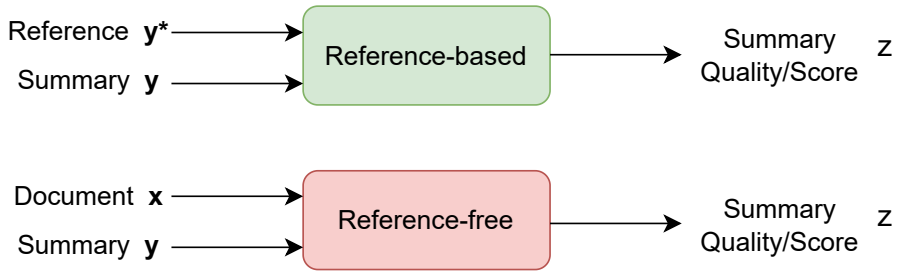


Fig. 4.5 Two types of content-based summary assessment methods: reference-based (top) and reference-free (bottom). Each assessment method can be either an unsupervised or supervised method.

Assessing extractive labelling tasks

While general summary assessment methods are applicable in both extractive and abstractive summarization, for extractive labelling tasks where ground-truth labels (i.e., binary label $z_i \in \{0, 1\}$ indicating whether sentence i is salient or not) are available, standard classification metrics such as accuracy, precision, recall, or F1 score can be used for evaluating this task. For example, extractive labels in the AMI dataset are available, so an F1-score evaluation is used in Section 5.5.5. However, in practice, it is common that only ground-truth summaries are available instead of ground-truth extractive labels. In this scenario, extractive summarization methods are evaluated in the same way as abstractive summarization methods where the top

sentences returned by an extractive method are concatenated as an output summary which is compared against the reference.

Human evaluation methods

Human evaluation of summaries is generally viewed as the gold-standard evaluation method since humans can reliably perform tasks that automatic methods often struggle with [88, 171]. For example, a human evaluator can be instructed to grade summaries based on certain criteria and guidelines, whereas doing this automatically is likely very challenging for most summarization applications. Hence, human evaluation is deemed more reliable than automatic methods, and it is considered the gold standard in most applications.

First, human evaluation can be in the form of direct rating (i.e., reference-free) based on specific criteria such as human evaluations performed in previous shared tasks, including DUC 2005 [44], TAC 2008 [45], and TREC 2020 [125]. This form allows human evaluators to judge summaries based on precise aspects one wishes to measure. However, it is essential to provide a clear scoring guide, and there could be inconsistency among the evaluators (even with training). This direct assessment requires human evaluators to understand the source documents, which can be highly labourious, especially when evaluating multiple documents or even just one in-depth document.

Second, human evaluation can be reference-based where human evaluators compare candidate summaries against gold-standard summaries. Common reference-based human evaluation methods are: Factoid [289], Pyramid [213], and Lightweight Pyramid [269]. In these methods, human evaluators assess whether the information in the gold-standard summary is in the candidate summary by comparing the content units of both texts. For example, the Pyramid method [213], illustrated in Figure 4.6, assesses the quality of a summary by comparing it against a set of possible content units extracted from a set of reference summaries. These content units, called Summary Content Units (SCUs), represent unique pieces of information. A pyramid is constructed by ranking SCUs based on how many reference summaries contain them: SCUs in more reference summaries are at the top, while less common SCUs are at the bottom. To evaluate a hypothesis summary, SCUs are identified within it and scored based on their level in the pyramid as follows,

$$\text{Pyramid score} = \frac{\sum_{i=1}^{N} i \times D_i}{\texttt{MaxScore}} \tag{4.14}$$

where D_i is the number of SCUs in level i, and `MaxScore` is the maximum value of $\sum_{i=1}^{N} D_i$ that has the same number of SCUs. The Pyramid method requires expert annotators, who are trained to identify and match SCUs. This makes calculating the Pyramid method expensive

and difficult to scale. The Lightweight Pyramid [269] was proposed to perform the Pyramid Method but with crowd-sourced workers.

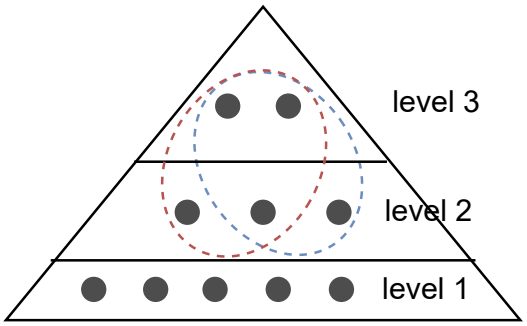


Fig. 4.6 Pyramid method. Each dot represents a summary content unit (SCU), e.g., "Daniel Radcliffe gets £20M". A level represents the importance of an SCU. Pyramid was originally proposed for multi-reference tasks and the importance was measured by how many times an SCU is present in ground-truth references. This figure shows two of three optimal (score = 1.0) summaries with 4 SCUs.

4.3.1 Reference-based Methods

In developing supervised summarization systems, the data consists of a set of documents $\mathcal{X} = \{X^{(1)}, X^{(2)}, ...\}$ and gold summaries $\mathcal{Y}^* = \{Y^{*(1)}, Y^{*(2)}, ...\}$, and the summarization systems are trained to maximize the likelihood of the gold summaries such that

$$\boldsymbol{\theta}_{\text{MLE}} = \text{argmax}_{\boldsymbol{\theta}}[P(\mathcal{Y}^*|\mathcal{X};\boldsymbol{\theta})] \tag{4.15}$$

As a result, traditional assessment methods, when gold summaries are available, take the form $f(Y, Y^*)$ where Y is the system-generated summary.

n-gram-based Methods

The most common reference-based assessment method is **ROUGE** (Recall Oriented Understudy for Gisting Evaluation) [162], which is based on the overlap between n-grams extracted from the summary and n-grams extracted from the reference. Let $n\text{-gram}(Y)$ denote the set of n-grams extracted from Y, the operator $\cap$ denote intersection, and $|.|$ is the number of items in the set. The definition of the ROUGE-n score as originally created is,

$$\text{ROUGE-}n\text{ (Recall)} = \frac{|n\text{-gram}(Y) \cap n\text{-gram}(Y^*)|}{|n\text{-gram}(Y^*)|} \tag{4.16}$$

which can be considered as the recall version of ROUGE. Existing ROUGE packages also include the precision version of ROUGE,[3]

$$\text{ROUGE-}n\text{ (Precision)} = \frac{|n\text{-gram}(Y) \cap n\text{-gram}(Y^*)|}{|n\text{-gram}(Y)|} \tag{4.17}$$

ROUGE-1 and ROUGE-2 are commonly adopted. An alternative ROUGE-based metric is ROUGE-L, which compares the longest common subsequence (LCS) [163],

$$\text{ROUGE-L (Recall)} = \frac{\text{LCS}(Y, Y^*)}{l_{Y^*}} \tag{4.18}$$

$$\text{ROUGE-L (Precision)} = \frac{\text{LCS}(Y, Y^*)}{l_Y} \tag{4.19}$$

where $\text{LCS}(Y^*, Y)$ is the length of the longest common subsequence of Y^* and Y, l_{Y^*} is the length of Y^*, and l_Y is the length of Y. ROUGE is usually reported as the harmonic mean,

$$\text{F1} = 2 \times \frac{\text{Prec} \times \text{Rec}}{\text{Prec} + \text{Rec}} \tag{4.20}$$

In practice, ROUGE packages such as the original Perl package or Python implementation `rouge-score`[4] often include text normalization and stopword removal.

Other n-gram based methods include BLEU [222] and METEOR [7]. Similar to the ROUGE-N *precision* (in Equation 4.17), **BLEU** counts matching n-grams in the system-generated summary to n-grams in the reference, but the difference is that BLEU score limits the matching count to the maximum n-gram count in the reference. **METEOR** computes precision and recall similar to ROUGE-1 (unigram), and it takes into account stemming and synonyms for matching. When computing the F-score, recall is weighed 9 times more than precision: $\text{F}_{\text{mean}} = 10 \times \frac{\text{Prec} \times \text{Rec}}{\text{Prec} + 9 * \text{Rec}}$. In addition, to account for longer n-gram matches, METEOR computes *penalty*, p, which is higher given more matches that are not adjacent, and finally METEOR score is computed as $M = \text{F}_{\text{mean}}(1 - p)$.

Representation-based Methods

Despite the robustness of n-gram-based methods, they do not take into account word semantics. Thus, researchers have replaced the surface-level comparison with semantic comparison. Using pre-trained language models such as ELMo [228] and BERT [52], representation-

[3] The precision version of ROUGE is the same as the BLEU score.
[4] https://pypi.org/project/rouge-score/

based methods have been proposed based on word (token) embeddings. Commonly used word-level representation-based methods are the following:

• **BERTScore** [52] computes the *semantic* similarity between tokens of reference and summary. Let $\{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, ..., \mathbf{h}_M\}$ be the token-level contextual BERT embeddings of the reference and $\{\hat{\mathbf{h}}_1, \hat{\mathbf{h}}_2, \hat{\mathbf{h}}_3, ..., \hat{\mathbf{h}}_N\}$ be the token-level contextual BERT embeddings of the summary. BERTScore calculates pairwise cosine similarity and maximum similarity as follows,

$$R_{\text{BERT}} = \frac{1}{M}\sum_i \max_j \left(\mathbf{h}_i \cdot \hat{\mathbf{h}}_j\right), \quad P_{\text{BERT}} = \frac{1}{N}\sum_j \max_i \left(\mathbf{h}_i \cdot \hat{\mathbf{h}}_j\right)$$

$$F_{\text{BERT}} = 2\frac{P_{\text{BERT}} \cdot R_{\text{BERT}}}{P_{\text{BERT}} + R_{\text{BERT}}} \tag{4.21}$$

• **MoverScore** [347] evaluates text generation quality similar to BERTScore, but it operates at a more macro, content-transformation level. It is based on Word Mover's Distance (WMD) [149], which is a measure in statistics that calculates the minimum amount of *work* required to transform one probability distribution into another. In this context, *work* corresponds to the semantic changes between sentences. MoverScore adapts this concept to measure the similarity between two sentences (e.g., the generated text versus the reference text). It uses pre-trained contextual embeddings (e.g., from BERT, ELMo, etc.) to represent the sentences in a high-dimensional space. Then, it computes the minimum amount of *movement* (in terms of semantic meaning) that would be required to transform the generated sentence into the reference sentence. The less movement required, the more similar the sentences are in terms of their overall semantic content, implying that the generated text is of higher quality.

Knowledge-based Methods

Knowledge-based methods involve converting texts into knowledge representations such as semantic triples or knowledge graphs. A **Semantic Triple** represents information using the "Subject-Relation-Object" pattern. (1) *Subject* is the entity from which the statement originates, which is often an individual entity described by some relationship; (2) *Relation* describes the relationship between the subject and the object, which is similar to the verb in a sentence; (3) *Object* is the entity that is linked by the relation to the subject, which is similar to the direct object in a sentence.

Multiple semantic triples can form a **Knowledge Graph**, which is a method to store information in a structured way. A knowledge graph is a collection of nodes and edges, where each node represents an entity, and each edge represents a relationship between those

entities. This structure is useful for handling complex interrelations. Figure 4.7, for example, shows an example of a text and its corresponding knowledge graph and semantic triples.

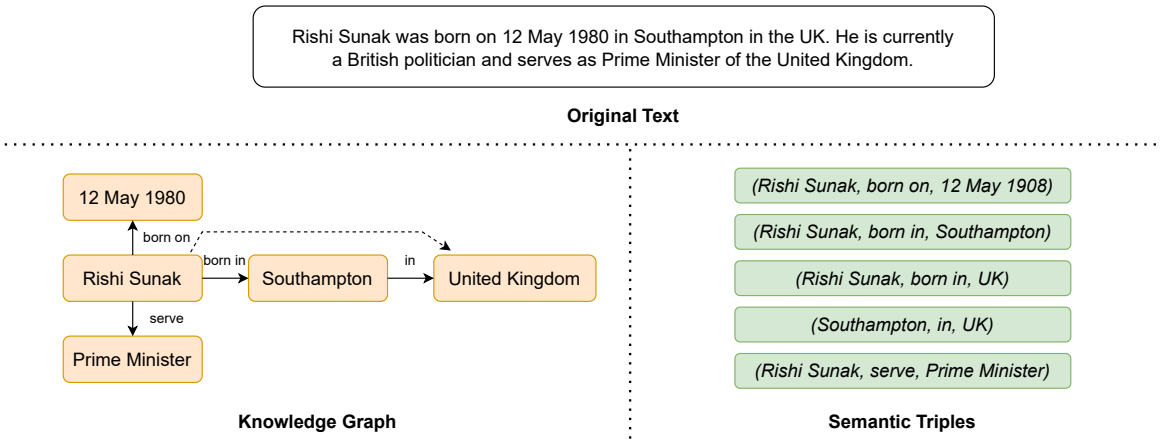


Fig. 4.7 An example of knowledge graph and semantic triples.

Existing work has utilized semantic triples for assessing summaries, and some examples are described below. Although knowledge graphs can capture more complex relations, their application in summary assessment is an open research question.

• **Triple-Matching**, proposed by Goodrich et al. [90], compares texts via the overlap of semantic triples to assess factual consistency. The semantic triples extracted from the summary are compared against the semantic triples extracted from the reference to compute the fact_{acc} score, which is defined as the precision of the generated summary,

$$\text{fact}_{\text{acc}}(Y, Y^*) = \frac{|\text{Triple}(Y) \cap \text{Triple}(Y^*)|}{|\text{Triple}(Y)|} \tag{4.22}$$

Goodrich et al. [90] considered various triple extraction methods. First, model-based triple extraction methods include (1) named entity recognition (NER) system (Stanford's CoreNLP or NLTK) + relation classifier (transformer encoder-only); (2) end-to-end model; (3) NER + binary relation classifier (i.e., selecting $\{0,1\}$ for every pair of entities). The models were trained using the WikiData knowledge base [296]. Second, the model-free triple extraction method is open-schema OpenIE [65]. In their experiments, fact_{acc} with the end-to-end triple extraction model achieved the best performance.

• **Fact-based Content Weighting**, proposed by Xu et al. [328], represent facts in a sentence by adapting Semantic Role Labelling (SRL) [220] which roughly captures "who did what to whom". The method creates the tree meaning representation from a list of facts. Automatic

content weighting computes argument and fact weights by the contextual similarity (e.g., BERT embeddings for content words) and their distance in the tree meaning representation.

Supervised Reference-based Methods

Supervised approaches require gold-standard summary scores (e.g., human judgements) for training. These methods are expected to have the best performance on the in-domain dataset; however, their weakness is on out-of-domain datasets. Most summary assessment datasets are small (and it is tedious to annotate summary scores), making these methods less practical. Examples of notatable supervised reference-based methods are: **S3** [230] train support vector regression (SVR) using the features (TF-IDF, KL-divergence, JS-divergence, and ROUGE) extracted from the source document, summary, and reference. **BLEURT** [264] and **COMET** [249] train neural models to predict summary score given the summary and reference.

Weakly Supervised Reference-based Methods

UniEval, proposed by Zhong et al. [354], converts NLG evaluation into a Boolean QA problem. This method uses a pre-defined scheme for each aspect (e.g., coherence) and generates synthetic data to fine-tune a T5 system assessing that aspect. References are required for some aspects such as relevancy,[5] and schemes/systems are bespoke to a particular attribute. The authors explored multi-task learning training and continual learning, and demonstrated that continual learning where a system is sequentially trained on multiple attributes performs the best.

Given the evaluation aspect A, context information C (which could be the source document and/or the reference), and the text to be evaluated Y, UniEval transforms (A,C,Y) into an input prompt $\mathcal{P} = \mathcal{P}(A,C,Y)$. For example, for the consistency aspect in summarization, the prompt template can be:

```
------------------------------------------------------------
Is this summary consistent with respect to the reference?
Summary: {summary}
Reference: {reference}
Answer: ...
------------------------------------------------------------
```

[5] Although for some aspects, references are required, for other scenarios, UniEval can be reference-free.

Then for each input prompt, the probabilities of predicting "Yes" and "No" are used to calculate the UniEval score:

$$\text{UniEval}(Y|A,C) = \frac{P(\text{`Yes'}|\mathcal{P}(A,C,Y);\boldsymbol{\theta})}{P(\text{`Yes'}|\mathcal{P}(A,C,Y);\boldsymbol{\theta}) + P(\text{`No'}|\mathcal{P}(A,C,Y);\boldsymbol{\theta})} \tag{4.23}$$

where in this particular example (assessing consistency with respect to the reference), the context C is the reference summary Y^*. The model $\boldsymbol{\theta}$ is trained on weakly supervised data such as NLI[6] datasets [334, 53, 307] and generic QA datasets [35, 135, 81, 84, 134].

4.3.2 Reference-free Methods

In practice, one cannot always assume that the reference or gold-standard summary is available for assessment of the generated summary, so in this scenario, assessment methods will have to rely on comparing the generated summary against the source document.[7] In addition, a prompt-based LLM (Section 4.2.3) generates summaries based on a prompt template and patterns that the LLM learned during pre-training, instruction-tuning, or reinforcement learning from feedback, so the applicability of comparing generated summaries against gold summaries is less clear compared to fine-tuned models. For example, Goyal et al. [91] showed that the GPT-3 prompting method cannot be evaluated using reference summaries, and they performed an A/B testing to compare GPT-3's outputs against the fine-tuned model's outputs. The following will categorize and describe the existing reference-free methods for summary assessment.

Textual Entailment Methods

Textual entailment, also known as natural language inference (NLI), is a relation between two natural language texts ('premise' and 'hypothesis'). The task is that: given a premise X and hypothesis Y, predict their relation from a set of possible options such as *entailment*, *neutral*, and *contradiction*. A widely-used dataset for NLI is the Multi-Genre Natural Language Inference (MNLI) dataset [313], which is a crowd-sourced collection of 433k sentence pairs. For example, an NLI score can be defined as,

$$\text{NLI-score}(X,Y) = P(\text{`entailment'}|X,Y;\boldsymbol{\theta}) \tag{4.24}$$

[6] The NLI dataset can be formatted into QA by, for example, 'Is this hypothesis entailed the premise'.

[7] In fact, widely used summarization datasets actually have pseudo summaries instead of real summaries. See Section 6.5.1 on *a note on reference summaries* for further discussion.

or the probability of the relation being entailment. Using textual entailment, Maynez et al. [193] showed that BERT fine-tuned to MNLI achieves the highest Spearman correlation with human judgements on faithfulness and factuality.

Similarly, FactCC, proposed by Kryscinski et al. [143], is a weakly-supervised method. Weakly supervised data is derived from a set of rule-based transformations, and BERT is adversarially trained to distinguish between real (faithful) and artificial (unfaithful) summaries.

Question Answering Methods

A question-answering method generally consists of a question-generation system and an answering model. This approach is designed to measure to what extent a summary provides sufficient information to answer relevant questions derived from the original context. As this approach is the most related to our proposed information consistency assessment method, a detailed background is provided in Section 7.2.

Language Model Scoring Methods

• **BARTScore** [337] evaluates texts using conditional language model scores,

$$\text{BARTScore} = \sum_{m=1}^{M} \omega_m \log P(y_m | y_{1:m-1}, X; \boldsymbol{\theta}) \tag{4.25}$$

where ω_m is a weight such as TF-IDF to put different emphasis on different tokens (note that the authors of BARTScore adopted equal weighting). Prompting can be applied to BARTScore such as appending text to the source text X and pretending text to the target text Y. Prompting has been shown to improve the performance of BARTScore.

• **GPTScore** (Generative Pre-training Score) [74] evaluates texts using conditional language model scores. By conditioning the language model $\boldsymbol{\theta}$ on instruction and context, GPTScore assumes that it will assign a higher probability to a high-quality generated text. The instruction (i.e., the input to the LLM $\boldsymbol{\theta}$) is composed of the task description and the evaluation aspect A, the context information (e.g., source document) X, and the text to be assessed (e.g., summary) $y_{1:M}$. GPTScore is defined similarly to BARTScore as,

$$\text{GPTScore}(y_{1:M} | X, A) = \sum_{m=1}^{M} \omega_m \log P(y_m | y_{1:m-1}, \mathcal{P}(X, A); \boldsymbol{\theta}) \tag{4.26}$$

where ω_m is the weight of the token at position m (note that the authors of GPTScore adopted equal weighting) and $\mathcal{P}(X, A)$ is the prompt given the instruction and context.

Supervised Reference-free Methods

Supervised approaches require ground-truth scores (e.g., human judgements) for the summaries $\mathcal{Z} = \{z^{(1)}, z^{(2)}, ..., z^{(J)}\}$ to train a regression model $\boldsymbol{\theta}$ that predicts

$$\hat{z} = f(X, Y; \boldsymbol{\theta}) \tag{4.27}$$

A common training objective is the mean squared error (MSE) that minimizes the L2 distance between the predicted score $\hat{z}$ and the ground-truth score z,

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{J} \sum_{j} \| z^{(j)} - \hat{z}^{(j)} \|_2^2 \tag{4.28}$$

For example, Xia et al. [321] collected English learners' summaries from a real examination, and have the summaries graded by professional examiners. Kernel Ridge Regression, LSTM, and CNN models were trained using this data. Bao et al. [9] trained fully connected, CNN, LSTM, and BERT-based models on *simulated* CNN/DailyMail, BillSum, arXiv, BigPatent data. They created simulated by negative sampling, e.g. random shuffling summaries or word-level summary corruption. Wu et al. [319] constructed negative samples with respect to linguistic qualities and informativeness, and they trained BERT-based models using contrasting learning.

4.3.3 Meta-Evaluation of Automatic Methods

As automatic assessment methods are designed to approximate gold-standard human judgements, automatic methods are evaluated by their similarity to human judgements, for example, through correlation measures. This evaluation of automatic assessment methods can be referred to as meta-evaluation.[8]

Summary assessment can operate at a range of levels such as (1) ranking summary generation systems, (2) ranking summaries of a particular document, and (3) estimating the quality of a document-summary pair on an absolute scale. Hence, meta-evaluation metrics can differ for different granularity levels.

Let u_i^j and v_i^j be two scores of metrics $f_U(.)$ and $f_V(.)$, for the candidate summary by system $i \in \{1, ..., N\}$ on the source document $j \in \{1, ..., M\}$. Correlations, which are the (meta)-evaluation metrics for summary assessment methods, are defined as follows.

[8] When working only on developing automatic evaluation/assessment methods, meta-evaluation will be referred to as evaluation for simplicity.

• **System-level Correlation** (i.e., Corpus-level) is used to measure how well an assessment method ranks summary generation systems. It computes the average of scores across documents for each system before computing a correlation, so it is generally less noisy compared to the summary-level correlation. The system-level correlation is defined as:

$$\rho = \text{Corr}\left(\left\{\frac{\sum_j u_i^j}{M}, \frac{\sum_j v_i^j}{M}\right\}_{i=1}^{N}\right) \tag{4.29}$$

• **Summary-level Correlation** (i.e., Sentence-level or Response-level) is used to measure how well an assessment method ranks summaries for each source document. It computes a correlation between predicted scores and ground-truths and then takes the average of individual correlation scores. The summary-level correlation is defined as:

$$\rho = \frac{1}{M}\sum_j \text{Corr}\left(\left\{u_i^j, v_i^j\right\}_{i=1}^{N}\right) \tag{4.30}$$

• **All-Example Correlation** computes a correlation between all predictions and ground-truth. It is used to evaluate all document-summary pairs together on an absolute scale. However, it is only applicable when the assessment method gives a score on an absolute scale. For example, the ROUGE score per one document is *not* comparable across documents as it is not in the same value range, so this metric is not applicable in this case.

$$\rho = \text{Corr}\left(\left\{u_i^j, v_i^j\right\}_{i=1,j=1}^{i=N,j=M}\right) \tag{4.31}$$

The exact form of correlation between $\{u_1, u_2, ..., u_n\}$ and $\{v_1, v_2, ..., v_n\}$ can be:

• *Pearson* correlation coefficient (PCC) assumes a linear relationship between two variables, and it calculates the correlation using this formula:

$$\rho = \frac{\sum_{i=1}^{n}(u_i - \bar{u})(v_i - \bar{v})}{\sqrt{\sum_{i=1}^{n}(u_i - \bar{u})^2}\sqrt{\sum_{i=1}^{n}(v_i - \bar{v})^2}} \tag{4.32}$$

• *Spearman*'s rank correlation coefficient (SCC) measures the monotonic relationship between two variables using their ranks instead of raw values. The formula is the same as the Pearson correlation coefficient (in Equation 4.32) with the difference in that ordinal ranks of U and V are used instead of raw values. For tied values, they receive the same average rank [71]. For instance, if two values of U are tied for the third smallest value, the ranks would be

3 and 4. The average of these two ranks is 3.5, and that is the value that is used as the rank of these two observations of U.

• *Kendall* rank correlation coefficient (KCC) [132] is similar to SCC in measuring the ranks, and KCC is preferred over SCC when the sample size is small.

4.4 Chapter Summary

This chapter provided the background of the summarization tasks. Section 4.1 covered unsupervised and supervised extractive summarization. Section 4.2 covered abstractive summarization approaches, which were categorized chronologically into three types: training neural models from scratch (which lays the foundation for Chapter 5), fine-tuning pre-trained models (which lays the foundation for Chapter 6), and prompting LLMs. Lastly, Section 4.3 covered summary assessment, including traditional reference-based methods and reference-free methods and meta-evaluation, which will be the focus of Chapter 7.

Chapter 5

Abstractive Summarization with Hierarchical RNNs

Earlier chapters have established the background for deep learning techniques and automatic summarization, setting the stage for the content in this chapter. Given recent advances in deep learning, modern summarization systems are abstractive summarization models based on neural networks [252, 29, 208]. Initially, these neural models adopt recurrent neural networks (RNNs) and they are trained from scratch. This often results in issues such as diversity in generated outputs when training data is limited. This chapter will study this problem in the context of spoken document summarization, which is less studied compared to standard summarization tasks such as news summarization. Spoken documents such as meeting recordings with gold-standard summaries are limited in size, which could lead to an issue about diversity. In addition, these spoken documents can be lengthy and have different characteristics from written documents such as disfluencies in speech or transcription errors. For existing attention-based sequence-to-sequence abstractive summarization systems, these challenges can yield a poor attention distribution over the spoken document words and utterances, impacting performance.

First, this chapter discusses the challenges of spoken document summarization in more detail in Section 5.1. To improve RNN-based approaches to abstractive spoken document summarization, Section 5.2 introduces a hierarchical encoder-decoder RNN model to explicitly model utterance-level attention distribution and exploit multi-task learning, including dialogue act classification and extractive summarization. Section 5.3 introduces modified hierarchical attention mechanisms at both training and inference stages to address the diversity problem. Given the flexibility of hierarchical attention mechanisms, Section 5.4 discusses sentence filtering to tackle redundant utterances. Section 5.5 provides experimental results as well as investigating the impact of ASR errors on the summarization performance.

5.1 Challenges in Spoken Document Summarization

Spoken documents can be more challenging, especially in situations involving human interactions, such as meetings, e.g., a snippet of a meeting transcript is illustrated in Figure 5.1. Compared to written texts, spoken language data can present additional challenges as follows:

• **Structure of Spoken Language**: In spoken documents such as meeting transcripts, the input source is typically longer, less grammatical, and contains less-structured utterances rather than well-constructed sentences. Consequently, Li et al. [160], Zhao et al. [349] demonstrated that standard sequence-to-sequence neural models with attention mechanisms that take a long sequence of tokens were less effective than hierarchical models for spoken document summarization. To exploit the hierarchical structure and incorporate additional information at the utterance level, this chapter will apply a multi-task learning framework, which is described in Section 5.2 with experimental results presented in Section 5.5.2.

• **Limited Data**: Text summarization systems are typically trained on news articles, scientific articles, or Wikipedia data. However, large-scale spoken language corpora are not readily available. Training a neural network from scratch usually requires a large dataset, which may be unavailable or expensive for spoken document systems. For example, the AMI corpus [19], which can be used for meeting summarization, only consists of hundreds of training documents – far fewer than news summarization datasets such as CNN/DailyMail [208] which has nearly 300 thousand documents. Furthermore, the summaries of these meetings are also similar to each other as they are mainly about a group of people discussing the design of a product. As a result, trained systems may produce commonly used words and repeated sentences, i.e., they may suffer the *diversity* problem. To address this challenge, this chapter proposes a sentence-level attention diversity criterion to be optimized in the multi-task learning framework at training, and a decoding method to improve diversity at inference. These methods are described in Section 5.3 with experimental results presented in Section 5.5.3.

• **Redundancy in Spoken Language**: Speech transcripts are not only less structured and less ungrammatical, but some of the sentences/utterances also convey little or no information. In addition, in the case of spontaneous speech, within an utterance, disfluencies and repetitions are present. Redundancies make source sequences unnecessarily long, leading to more complex dependencies that the model has to learn. To address redundancy, this chapter proposes using the sentence level (i.e., utterance level) attention scores as a measure of sentence importance to perform content selection. This is described in Section 5.4 with experimental results presented in Section 5.5.6.

• **Automatic Speech Recognition (ASR) derived transcripts**: To apply a summarization system on a spoken document, an ASR system (or manual transcription) is required to produce a transcript of the audio data. In sequence labelling tasks such as grammatical error detection, Knill et al. [139], Lu et al. [177] demonstrated that ASR-derived inputs result in the worst downstream system performance. This is because ASR makes errors such as wrongly derived, or segmentation errors. Similarly, we expect that transcription has an impact on summarization performance. Thus, this chapter examines the impact of ASR on summarization empirically in Section 5.5.4.

```
UTT_ID        transcription
-------------------------------------------------------------------------------------------------------------
utt0000.A     okay
utt0001.A     hi everybody and welcome to our kick-off meeting um for our new product that we're gonna
utt0002.A     um i'm mandy and i'm the project manager
utt0003.A     and i know all your names again , courtney , fenella and amber
utt0004.D     yep
utt0005.B     yep
utt0006.A     alright . okay ,
utt0007.A     so first let's go through this powerpoint
utt0008.A     i wonder what button i press ?
utt0009.C     just do it on the arrow
utt0010.A     yeah ,
utt0011.A     or how about i just click ?
utt0012.A     okay , here is our agenda for this meeting
...
...
...
utt0224.D     right
utt0225.A     i would think so . okay
utt0226.A     and you're gonna get more specific instructions emailed to you in just a little while
utt0227.A     okay , so does anybody have anything they wanna say before we close the meeting ?
utt0228.A     okay . this meeting is officially over
```

Fig. 5.1 Excerpt from AMI meeting (ES2010a). The first column shows utterance ID and speaker ID. The transcriptions and utterance boundaries were manually derived. Redundancies within sentences are, for example, `um`, `okay`, and redundant utterances/sentences are `utt0000.A`, `utt0004.D`, `utt0005.B`, `utt0006.A`, `utt0225.A`, etc.

5.2 Hierarchical Model

Although hierarchical RNN models are not commonly used in standard summarization tasks such as news summarization, previous work [160, 349] showed that the hierarchical models are more suitable for spoken document tasks such as meeting summarization. Thus, we focus on a hierarchical model as the base model. Additionally, Yuan and Yu [336] demonstrated semantic slot and dialogue domain can improve dialogue summarization. Utterances (or sentences)[1] may also serve different functions within a conversation, and interactive signals such as dialogue acts have been shown to be useful in predicting topic description [89]. As a

[1] Sentences and utterances are used interchangeably.

result, this chapter aims to utilize multi-task learning based on the sentence-level features of the hierarchical model to improve summarization performance.

5.2.1 Hierarchical Encoder-Decoder Architecture

In Section 2.1.3, the vanilla encoder-decoder architecture is described. This section will extend it to a hierarchical architecture, following previous work [160, 349] that also applied hierarchical models to spoken document summarization. The hierarchical architecture consists of (1) an encoder with word-level and sentence-level RNNs, and (2) a decoder with an RNN with an attention mechanism over the encoder states. This hierarchical encoder-decoder architecture is illustrated in Figure 5.2.

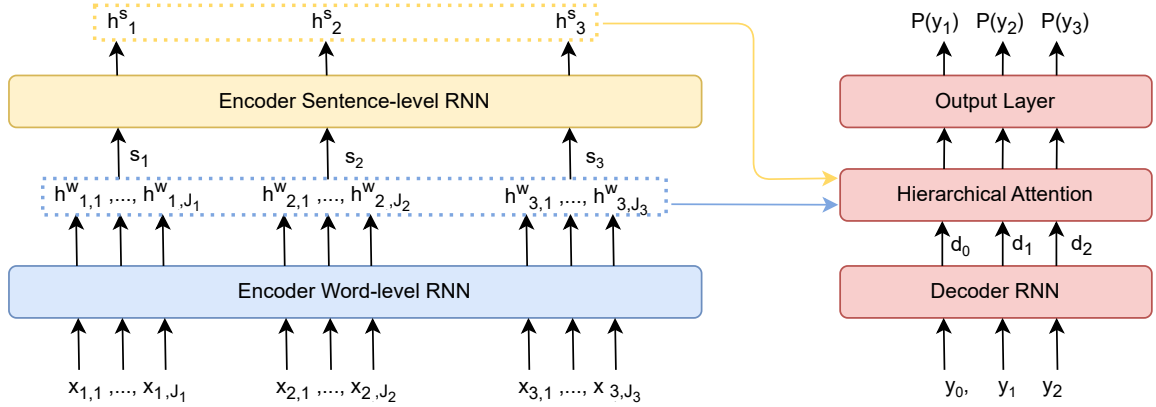


Fig. 5.2 Hierarchical encoder-decoder architecture. A variant of RNN (e.g., LSTM [109] and GRU [28]) or a self-attention layer can replace the RNN layer.

Notation: X is a source document with N sentences $\{S_1, S_2, ..., S_N\}$ and $Y = y_{1:M}$ is the corresponding summary. Each sentence S_i contains words $\{x_{i,1}, x_{i,2}, ..., x_{i,J_i}\}$, and the word $x_{i,j}$ is embedded into a vector $\mathbf{x}_{i,j}$. At the **word level**, a bidirectional word-level RNN is applied, and the forward and backward states are concatenated to obtain word-level representations:

$$\{\overrightarrow{\mathbf{h}}^{\mathrm{w}}_{i,1}, ..., \overrightarrow{\mathbf{h}}^{\mathrm{w}}_{i,J_i}\} = \text{Forward-WordRNN}(\{\mathbf{x}_{i,1}, \mathbf{x}_{i,2}, ..., \mathbf{x}_{i,J_i}\}) \tag{5.1}$$

$$\{\overleftarrow{\mathbf{h}}^{\mathrm{w}}_{i,1}, ..., \overleftarrow{\mathbf{h}}^{\mathrm{w}}_{i,J_i}\} = \text{Backward-WordRNN}(\{\mathbf{x}_{i,1}, \mathbf{x}_{i,2}, ..., \mathbf{x}_{i,J_i}\}) \tag{5.2}$$

$$\mathbf{h}^{\mathrm{w}}_{i,j} = [\overrightarrow{\mathbf{h}}^{\mathrm{w}}_{i,j}; \overleftarrow{\mathbf{h}}^{\mathrm{w}}_{i,j}] \tag{5.3}$$

At the **sentence level**, the input sentence representation is the concatenation of the word-level representations as follows,

$$\mathbf{s}_i = [\overrightarrow{\mathbf{h}}^{\mathrm{w}}_{i,J_i}; \overleftarrow{\mathbf{h}}^{\mathrm{w}}_{i,0}] \tag{5.4}$$

A bidirectional sentence-level RNN is then applied:

$$\{\mathbf{h}^{\mathtt{s}}_1, \mathbf{h}^{\mathtt{s}}_2, ..., \mathbf{h}^{\mathtt{s}}_N\} = \text{Bi-SentenceRNN}(\{\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_N\}) \tag{5.5}$$

As opposed to the vanilla encoder-decoder with an attention mechanism (described in Section 2.1.3), the hierarchical structure enables the decoder to *first* select salient input sentences *then* select salient words in each sentence. Given the decoder hidden state $\mathbf{d}_m$ (at decoder time step m), the hierarchical attention mechanism is defined as follows,

- Sentence-level attention:

$$\alpha^{\mathtt{s}}_{m,i} = \frac{\exp(\mathbf{d}^T_m \mathbf{W}^{\mathtt{s}} \mathbf{h}^{\mathtt{s}}_i)}{\sum_{i'} \exp(\mathbf{d}^T_m \mathbf{W}^{\mathtt{s}} \mathbf{h}^{\mathtt{s}}_{i'})} \tag{5.6}$$

- Word-level attention:

$$\alpha^{\mathtt{w}}_{m,i,j} = \alpha^{\mathtt{s}}_{m,i} \left(\frac{\exp(\mathbf{d}^T_m \mathbf{W}^{\mathtt{w}} \mathbf{h}^{\mathtt{w}}_{i,j})}{\sum_{j'} \exp(\mathbf{d}^T_m \mathbf{W}^{\mathtt{w}} \mathbf{h}^{\mathtt{w}}_{i,j'})} \right) \tag{5.7}$$

The word-level attention distribution is used to produce a context vector:

$$\mathbf{c}_m = \sum_i \sum_j \alpha^{\mathtt{w}}_{m,i,j} \mathbf{h}^{\mathtt{w}}_{i,j} \tag{5.8}$$

Lastly, similar to Equation 2.18 for the vanilla encoder-decoder model, the context vector is concatenated with the decoder state and fed through a linear layer, which has output units equal to the vocabulary size, to produce the output word distribution:

$$P(y_m | y_{1:m-1}, x_{1:N}) = \text{softmax}(\mathbf{W}^{\circ}[\mathbf{d}_m; \mathbf{c}_m] + \mathbf{b}^{\circ}) \tag{5.9}$$

5.2.2 Multi-Task Learning

The model is trained on the maximum likelihood estimation (MLE) criterion as described in Section 2.2,

$$\mathcal{L}_{\text{MLE}} = -\frac{1}{J} \sum_{j=1}^{J} \log P\left(Y^{(j)} | X^{(j)}; \boldsymbol{\theta}\right) \tag{5.10}$$

When a sentence-level annotation is available (e.g., dialogue act classes and extractive summarization labels), these auxiliary signals can be used for multi-task learning [89, 336]. In the following investigation, it is assumed that dialogue act classification and extractive summarization information are available. Given the output of the sentence-level RNN for

sentence i, $\mathbf{h}_i^{\mathrm{s}}$, we pass the vector to linear layers and softmax/sigmoid activation functions:

$$P_{\mathrm{DA}}(q_i|\mathbf{h}_i^{\mathrm{s}}) = \hat{q}_i = \mathrm{softmax}(\mathbf{W}^{\mathrm{DA}}\mathbf{h}_i^{\mathrm{s}} + \mathbf{b}^{\mathrm{DA}}) \tag{5.11}$$

$$P_{\mathrm{EX}}(z_i|\mathbf{h}_i^{\mathrm{s}}) = \hat{z}_i = \mathrm{sigmoid}(\mathbf{w}^{\mathrm{EX}} \cdot \mathbf{h}_i^{\mathrm{s}} + b^{\mathrm{EX}}) \tag{5.12}$$

where $\mathbf{W}^{\mathrm{DA}}$, $\mathbf{b}^{\mathrm{DA}}$, $\mathbf{w}^{\mathrm{EX}}$, b^{EX} are trainable parameters, $q \in \{1, ..., D\}$ is the dialogue act (one of D possible classes), and $z \in \{0, 1\}$ the binary extractive summarization label. Alternatively, z could be a continuous value representing how the saliency level of a sentence. The extractive summarization, $\mathcal{L}_{\mathrm{EX}}$, and dialogue act, $\mathcal{L}_{\mathrm{DA}}$, loss functions are maximum likelihood based:

$$\mathcal{L}_{\mathrm{DA}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{d=1}^{D} q_{i,d} \log \hat{q}_{i,d} \tag{5.13}$$

$$\mathcal{L}_{\mathrm{EX}} = -\frac{1}{N}\sum_{i=1}^{N} (z_i \log \hat{z}_i + (1 - z_i) \log (1 - \hat{z}_i)) \tag{5.14}$$

where $q_{i,d}$ is the one-hot encoding of q_i. Note that Section 5.3.2 will describe $\mathcal{L}_{\mathrm{DV}}$ diversity loss function, and the overall training loss function is given in Equation 5.28 in Section 5.5.2.

5.3 Exploiting Attention Diversity

For a sequence generation task such as summarization, See et al. [263], Tu et al. [288] observed that at the inference stage, neural models may repeat themselves even when using beam search decoding. We will refer to this problem as the *diversity* of the output. Typically, an encoder-decoder architecture is often used for sequence-to-sequence tasks, and the diversity problem is related to exposure bias [248]. At training time, the model is trained using teaching forcing [315], so the decoder sees a variety of input tokens from ground-truths. In contrast, at test time, the model has to rely on its own predictions, which are often less diverse than the ground truth tokens. This section focuses on approaches that aim to tackle the diversity issue directly, which are different from and can be used in addition to those designed for the broader exposure bias problem such as scheduled sampling as discussed in Section 2.2.1.

5.3.1 Existing Attention Diversity Methods

At *training* time, existing methods control the attention mechanism by incorporating additional training loss to improve diversity, including:

• **Coverage Mechanism** [288, 263] computes a coverage vector $\mathbf{c}_t$ to track what has been already attended over. It is computed by summing the attention score $\boldsymbol{\alpha}_t$ (of the vanilla encoder-decoder model):

$$\mathbf{c}_t = \sum_{\tau=1}^{t-1} \boldsymbol{\alpha}_\tau \tag{5.15}$$

The attention score at time step t, $\boldsymbol{\alpha}_t$, is modified to incorporate $\mathbf{c}_t$. A coverage loss to penalize repeatedly attending to the same places is defined as,

$$\mathcal{L}_{\text{COV}} = \sum_t \left(\sum_i \min(\alpha_{t,i}, c_{t,i}) \right) \tag{5.16}$$

where $\alpha_{t,i}$ represents the i-th component of $\boldsymbol{\alpha}_t$.

• **Global Variance Loss** [98] is defined as follows,

$$g_i = \sum_t (\alpha_{t,i}) - \max_t (\alpha_{t,i}) \tag{5.17}$$

$$\mathcal{L}_{\text{GV}} = \frac{1}{N} \sum_{i=1}^{N} (g_i - \bar{g})^2 \tag{5.18}$$

where $\bar{g}$ is the average of g_i. The global variance loss encourages the model to assign a high attention weight to the same positions once.

At the *inference* stage, Paulus et al. [224] observed that reference summaries are unlikely to contain the same trigram twice in a particular dataset, and they proposed a simple method to force the decoder to never output the same trigram twice by zeroing out the probability of repeated trigrams:

$$p(y_m | y_{1:m-1}, X; \boldsymbol{\theta}) = 0 \tag{5.19}$$

In general, this heuristic method should increase the diversity in the output, and we refer to it as **n-gram blocking**.

5.3.2 Attention Diversity at Training Stage

When summarizing, information from a different set of input sentences is typically used to generate each of the summary output sentences. For example, when humans produce words in one output sentence, we often use information from the same input sentences, whereas when we produce different output sentences information is gathered from different input utterances. This motivates us to model the diversity at the sentence level within the

sequence-to-sequence attention mechanism, allowing this diversity to be explicitly optimized during **training**. This diversity modelling approach is different to the coverage mechanism [263, 288] and global variance loss [98] methods (previously described in Section 5.3.1), which are also designed to mitigate repetitions as follows. The proposed approach operates at the sentence level instead of the word level. In addition, the proposed method encourages sentence-level attention to be similar when generating the same output sentences but varied when generating different output sentences.

Given the hierarchical attention mechanism (defined in Section 5.2.1), it is possible to define the diversity of input document attention both within an output sentence (*intra-sentence*) and between output sentences (*inter-sentence*). Let $\boldsymbol{\alpha}_t^{\mathrm{s}}$ be the vector representing all sentence-level attention scores at decoder step t. Inter-sentence and intra-sentence diversity scores can then be expressed as follows,

- **intra-sentence**:

$$D_{\text{intra},k} = \frac{1}{\binom{T_k}{2}} \sum_{t_1=1}^{T_k-1} \sum_{t_2=t_1+1}^{T_k} \| \boldsymbol{\alpha}_{t_1}^{\mathrm{s}} - \boldsymbol{\alpha}_{t_2}^{\mathrm{s}} \|_2 \tag{5.20}$$

$$D_{\text{intra}} = \frac{1}{K} \sum_{k=1}^{K} D_{\text{intra},k} \tag{5.21}$$

- **inter-sentence**:

$$\bar{\boldsymbol{\alpha}}_k^{\mathrm{s}} = \frac{1}{T_k} \sum_{t=1}^{T_k} \boldsymbol{\alpha}_t^{\mathrm{s}} \tag{5.22}$$

$$D_{\text{inter}} = \frac{1}{\binom{K}{2}} \sum_{k_1=1}^{K-1} \sum_{k_2=k_1+1}^{K} \| \bar{\boldsymbol{\alpha}}_{k_1}^{\mathrm{s}} - \bar{\boldsymbol{\alpha}}_{k_2}^{\mathrm{s}} \|_2 \tag{5.23}$$

where k denotes output sentence k, T_k is the number of tokens in sentence k, K is the total number of output sentences, and $\binom{n}{k} = \frac{n!}{k!(n-k)}$ is a binomial coefficient.

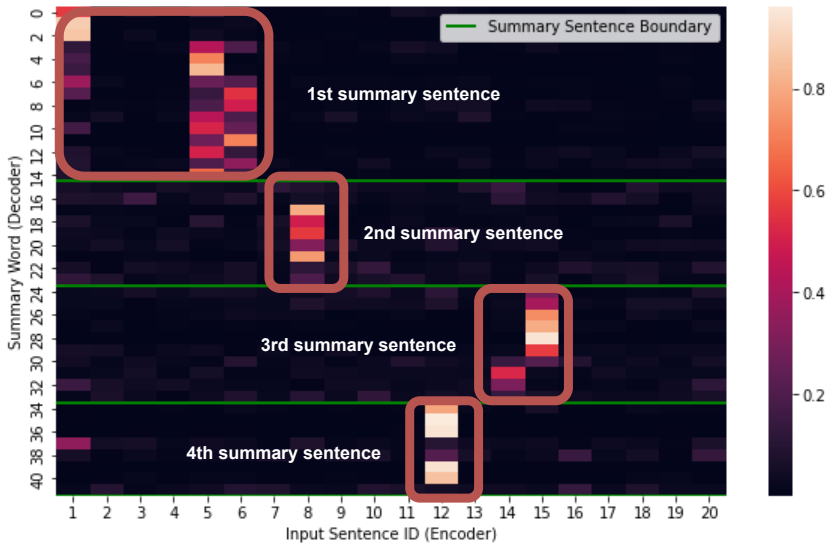


Fig. 5.3 Attention weights at the sentence level. Each box (in red) represents input sentences that a summary sentence focuses on the most.

As illustrated in Figure 5.3, the intra-sentence measure is the diversity of attention weights within each box, while the inter-sentence measure is the diversity across boxes. Motivated by the fact that as humans produce words in a particular output summary sentence, we are likely to attend over the same source sentence, whereas when we produce a new output sentence we will attend over different source sentences. This is equivalent to *low* intra-sentence (Equation 5.21) and *high* inter-sentence (Equation 5.23) diversity scores. Therefore, the **diversity loss** is defined to be directly minimized as follows,

$$\mathcal{L}_{\text{DV}} = \frac{D_{\text{intra}}}{D_{\text{inter}}} \tag{5.24}$$

The diversity loss, $\mathcal{L}_{\text{DV}}$, can then be optimized in addition to those losses in a multi-task learning framework introduced in Section 5.2.2.

5.3.3 Attention Diversity at Inference Stage

The previous section described a training loss to improve attention diversity. Another direction is to improve attention diversity at the inference stage.

An existing inference-time method includes a n-gram blocking method [224] described in Section 5.3.1. Instead of preventing repeated n-grams with a hard constraint, this work considers a soft criterion to penalize existing unigrams (without losing generality, it could be

applied to n-grams),

$$\hat{y}_t = \underset{y_t \in \mathcal{V}}{\arg\max} \left\{ \log P(y_t | y_{1:t-1}, X; \boldsymbol{\theta}) - \beta \left(\frac{\sum_{t'=1}^{t} \mathbb{1}_{y_t}(\hat{y}_{t'})}{t} \right) \right\} \tag{5.25}$$

where β is the unigram bias constant, $\mathcal{V}$ is the decoding vocabulary, and

$$\mathbb{1}_{y_t}(\hat{y}_{t'}) = \begin{cases} 1, & \text{for} \quad y_t = \hat{y}_{t'} \\ 0, & \text{for} \quad y_t \neq \hat{y}_{t'} \end{cases} \tag{5.26}$$

When this penalty method is applied to unigram at test time, this work will refer to it as **unigram bias decoding**. Note that a similar approach was also used to penalize n-grams in a concurrent work in controllable language generation by Keskar et al. [133].

5.4 Hierarchical Model for Sentence Filtering

To utilize summarization models trained on written texts, spoken documents are typically processed to be as close to written text as possible. One method is to detect and remove disfluencies, which has been shown to improve other downstream tasks [178]. We can extend word-level disfluency removal to the sentence level, for example, by removing sentences that contain little or no information. Although this process is similar to extractive summarization, the difference is that extractive summarization seeks to find a set of sentences that represents the key information in the original documents, whereas sentence filtering seeks to filter out sentences that convey little or no information and find an optimal set of sentences for downstream abstractive summarization. Additionally, the performance of a *sentence filtering system* is measured by the performance of the downstream abstractive summarization task.

A spoken document such as a meeting recording or podcast may contain 30 minutes of speech or longer, and the corresponding transcription could result in more than 5,000 tokens. This long input makes it inefficient and infeasible to train abstractive summarization systems. RNN-based systems are prone to vanish gradients and exploding gradients for long sequences, while self-attention in the transformer architecture has a quadratic dependency on the sequence length. Note that Section 6.2 discusses other sentence filtering approaches with a focus on handling quadratic dependency in the transformer's attention.

Sentence-level Attention Score as a Measure of Sentence Importance

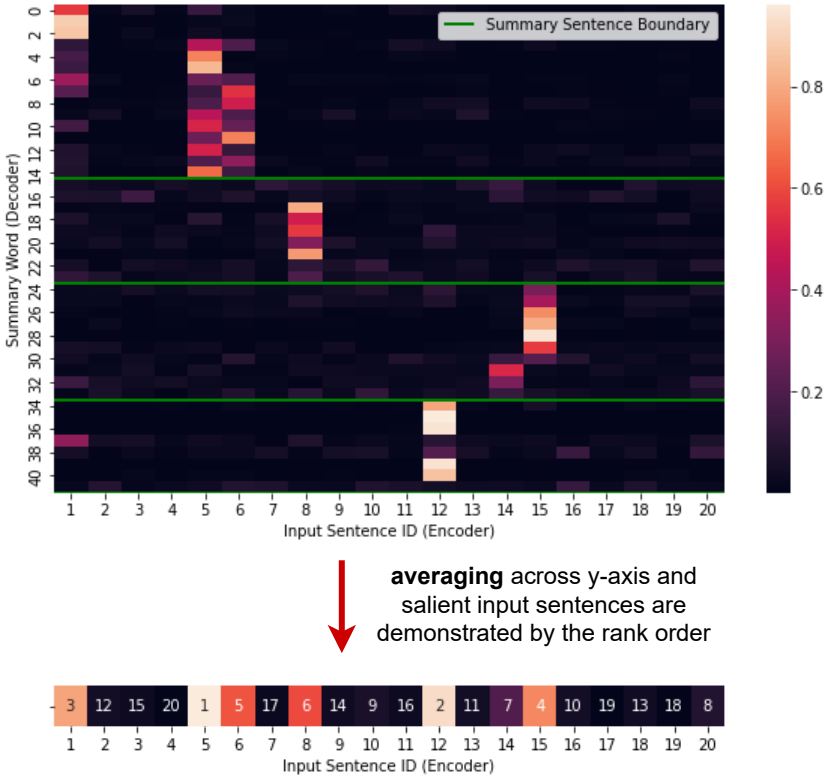


Fig. 5.4 Attention weights (same as Figure 5.3) and the corresponding sentence-level attention score as a measure of sentence importance.

The sentence-level attention score of a hierarchical RNN model (Section 5.2.1), $\alpha^{s}_{t,i}$, measures the importance sentence i in generating output at time t. We hypothesize that if we sum the sentence-level attention score over all time instances as illustrated in Figure 5.4, this should yield a measure of the importance of sentence i,

$$v_i = \frac{1}{T}\sum_{t=1}^{T} \alpha^{s}_{t,i} \tag{5.27}$$

where v_i is the importance measure of sentence i respective to the entire summary. It should be noted if the target sequence is available (e.g., in training data), we can use teacher-forcing decoding on the ground-truth target sequence to obtain the importance score. However, if the ground-truth target sequence is not available, we have to rely on a decoding method such as beam search to obtain a target sequence.

5.5 Experiments

The experiments in this chapter investigate the proposed hierarchical RNN model (Section 5.2) with the attention diversity methods (Section 5.3) on abstractive spoken document summarization. Furthermore, multi-task learning tasks including dialogue act classification and extractive summarization are investigated. The experiments also cover the impact of ASR performance on downstream abstractive summarization. This chapter also provides preliminary results on sentence filtering using the attention scores of the hierarchical model (while the next chapter provides more detail on sentence filtering in general). The evaluation metrics for the following experiments are the ROUGE scores [162] as commonly used in text summarization.

5.5.1 Datasets

AMI Meeting Corpus

The AMI corpus [19] contains meeting recordings of four people discussing a remote control design project. Each meeting is about 30 minutes, and there are 137 meetings (excluding those without the annotation required for multi-task learning). This work makes use of the dialogue acts, and extractive and abstractive summaries annotation in addition to the manual transcripts. The default data split is used: 97 training, 20 validation, and 20 test meetings, following the guidelines on the AMI website. The manually derived transcripts have 784 utterances, 6,200 words, on average per meeting, and each summary contains 10 sentences of 172 words on average (and up to 300 words). Two sets of automatically derived transcripts (test set), from ASR systems using manual segmentation, are used for testing our final summarization system:

- ASR1 - AMI official release v1.5[2] which is publicly available and also used in Shang et al. [267], Li et al. [160]. The word error rate (WER) is 36%.
- ASR2 - We generate this set of ASR transcripts using a CUED-trained ASR system: TDNN-F acoustic model [233] trained on AMI IHM training set using the Kaldi toolkit [234], with a 4-gram language model trained on the same set and the Fisher Corpus (LDC2004T19) [33]. 40-dim Mel-scaled filterbank features and 15 TDNN-F layers were used. The word error rate (WER) is 20%.

[2]http://groups.inf.ed.ac.uk/ami/download/

CNN/DailyMail

This dataset, introduced by Hermann et al. [104], contains news articles paired with multi-sentence summaries (highlights) and processed and processed by Nallapati et al. [208]. Currently, it is one of the most widely used text summarization benchmark corpus. The training, validation, and test split is 287,226, 13,368, and 11,490.

5.5.2 Hierarchical Model Baselines

This experiment aims to investigate the effectiveness of the hierarchical model, which explicitly models sentence-level attention distribution, for abstractive spoken document summarization. Another aspect of this experiment is the effectiveness of enforcing output diversity in training time and test time. Furthermore, multi-task learning tasks including dialogue act classification and extractive summarization are studied. Firstly, we investigate the meeting summarization using the AMI corpus. For initial contrasts with the hierarchical systems, the following baseline methods are used:

1. **Lead-Sentences**: This method simply selects the first K sentences in the input document as the summary. For the news article summarization task, the first few sentences are typically used as a baseline summary [207]. In this work, the first K sentences will be selected, and the value that yields the best results on the validation set will be used on the test set. Compared to news summarization, it is expected that in meeting summarization, the information is more evenly spread out in the source, so this work proposes DecoderLM as another baseline.

2. **DecoderLM**: This model is a decoder-only model with the same architecture as the decoder of the hierarchical model, but with no conditioning on the input document. This can be viewed as a baseline to assess the complexity and diversity of the summaries. The motivation of this baseline is that for a task where there is low diversity in the ground-truth outputs, DecoderLM (without any source document) should learn the generic output which could achieve a good score.

3. **Pointer Generator Network (PGN)** [263]: As described in detail in Section 4.2.1, PGN is an RNN encoder-decoder model with a copying mechanism. In this work, we use a publicly available pointer-generator network (PGN) implementation[3] with the hyperparameters set as in the original work by See et al. [263].

[3]https://github.com/atulkum/pointer_summarizer

For the hierarchical encoder-decoder model (in Section 5.2), a PyTorch implementation is developed and open-sourced at https://github.com/potsawee/spoken_summ_div. The model is trained on a combination of objectives (in Section 5.2.2 and Section 5.3.2), and the complete loss to *minimize* is a linear combination of all the losses:

$$\mathcal{L} = \mathcal{L}_{\text{MLE}} + \lambda_1 \mathcal{L}_{\text{DA}} + \lambda_2 \mathcal{L}_{\text{EX}} + \lambda_3 \mathcal{L}_{\text{DV}} \tag{5.28}$$

where $\mathcal{L}_{\text{MLE}}$ is the maximum likelihood loss on the sequence prediction task (Equation 5.10), $\mathcal{L}_{\text{DA}}$ the dialogue act prediction loss (Equation 5.11), $\mathcal{L}_{\text{EX}}$ the extractive summarization loss (Equation 5.12), and $\mathcal{L}_{\text{DV}}$ the diversity loss (Equation 5.24). When training without auxiliary tasks and diversity loss, $\lambda_1 = \lambda_2 = \lambda_3 = 0.0$.

Additionally, Section 5.5.4 will compare our best hierarchical encoder-decoder model against current state-of-the-art meeting summarization systems as follow:

1. **Unsupervised Abstractive Summarization with Multi-Sentence Compression and Budgeted Submodular Maximization (UMCB)** [267]: This method is multi-step. The first clusters sentences (i.e., utterances in spoken documents) into groups using LSA and k-means algorithm. Then, each group of sentences is summarized into a single sentence using the Multi-Sentence Compression Graph (MSCG) [69]. The final summary is then generated by selecting the best elements from the set of sentences from the second step using a budget constraint optimization.

2. **TopicSeg** [160]: This method uses an encoder-decoder architecture. The encoder is based on GRU [28], and the architecture is also hierarchical similar to our model. The attention mechanism of TopicSeg has a segment level on top of the sentence level. An extension of TopicSeg is **TopicSeg+VFOA** where VFOA is based on a neural network to extract visual information (*row*, *pitch*, *yaw*, *azimuth*, *elevation*) for each speaker and each transcribed utterance (i.e. sentence). The VFOA features are at the sentence level, which are then added to the sentence-level textual representation.

Summarization Performance

The performance of the system was first evaluated using the manual AMI transcripts, and the results are shown in Table 5.1. The baselines for this task are described above. First, the lead 10 sentences (utterances) are used as the Lead baseline, and the results show that it achieves poor performance since information is more evenly spread in the source compared to news summarization. Next, the DecoderLM baseline archives higher performance than the Lead baseline, demonstrating that the summaries in AMI are not diverse as the model

without any context can achieve reasonable performance. Nevertheless, the PGN with coverage, PGN+Cov, outperforms the DecoderLM baseline as this model can utilize the source document. Additional gains can be obtained with transfer learning (TL) where the model was initially trained on the CNN/DailyMail data. The hierarchical models achieve higher ROUGE scores, consistent with Li et al. [160]. This experiment confirms the effectiveness of the hierarchical model on the meeting dataset. Next, we investigate the output diversity of this hierarchical model.

		ROUGE F_1		
Model	**TL**	**R1**	**R2**	**RL**
Lead-10	✗	24.38	3.65	15.99
DecoderLM	✗	26.00±3.55	8.26±1.62	24.78±3.15
PGN+Cov	✗	29.90±0.88	10.41±0.99	28.16±1.12
PGN+Cov	✓	33.43±1.70	11.29±1.55	31.42±1.64
Hierarchical	✗	33.07±0.66	10.87±1.18	31.77±0.86
Hierarchical	✓	**39.12±2.45**	**13.03±1.58**	**36.77±2.28**

Table 5.1 Summarization Performance as measured by ROUGE F_1 on the AMI test set. TL denotes that the model was initially trained on CNN/DailyMail. The reported numbers are *mean* ± *standard deviation* from 3 runs with different data shuffling and different seeds.

5.5.3 Attention Diversity Results

Initially, the diversity of the test set summaries was evaluated for three generation modes with the hierarchical model: teacher forcing (TF); free-running (FR); and unigram bias decoding (UB). Teacher forcing and free running generation methods are described in Section 2.3.1, and unigram bias decoding is proposed in Section 5.3.3.

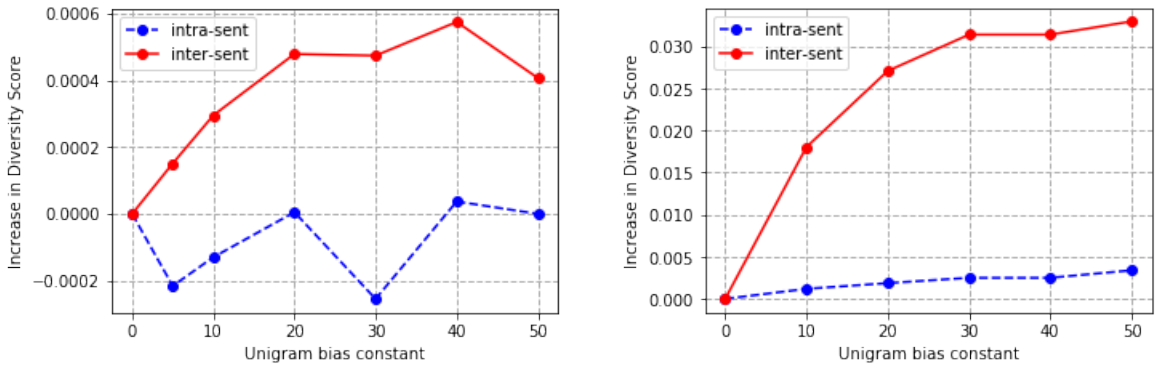
Experimental results in Table 5.2 show that the FR mode has lower inter-sentence diversity compared to the TF mode. This explains an observed higher number of repetitions since the diversity from the reference in TF has been lost. Next, the results show that using UB decoding increases the inter-sentence diversity. Figure 5.5 shows a detailed analysis of UB decoding with the value of β on both datasets. There is little variation in the intra-sentence variability, but clear gains in the inter-sentence diversity. This suggests that UB decoding, which penalizes repeated unigrams, forces the model to attend to different input sentences as it generates different summary sentences.

Additionally, we can see the impact of unigram bias decoding on diversity by computing coverage loss and global variance loss (defined in Section 5.3.1). Table 5.3 shows that both

losses increase when swapping from teacher forcing to free running, but the losses can be reduced using the unigram bias method, thus, increasing attention diversity.

	AMI		CNN/DailyMail	
Decoding	**intra** (↓)	**inter** (↑)	**intra** (↓)	**inter** (↑)
Teacher Forcing	0.0131	0.0066	0.0744	0.0837
Free Running	0.0133	0.0054	0.0755	0.0789
Unigram Bias	0.0133	0.0059	0.0774	0.1060

Table 5.2 Diversity scores on the test sets of AMI and CNN/DailyMain datasets where intra-sentence diversity score is defined in Equation 5.21 and inter-sentence diversity score is defined in Equation 5.23. Unigram bias is with $\beta = 20.0$.



(a) AMI **(b)** CNN/DailyMail

Fig. 5.5 Unigram bias constant β (x-axis) against Increase in Diversity Score (y-axis)

	AMI		CNN/DailyMail	
Decoding	$\mathcal{L}_{\text{COV}}$ (↓)	$\mathcal{L}_{\text{GV}}$ (↓)	$\mathcal{L}_{\text{COV}}$ (↓)	$\mathcal{L}_{\text{GV}}$ (↓)
Teacher Forcing	0.930	0.028	0.461	0.026
Free Running	0.958	0.062	0.642	0.158
Unigram Bias	0.949	0.052	0.611	0.121

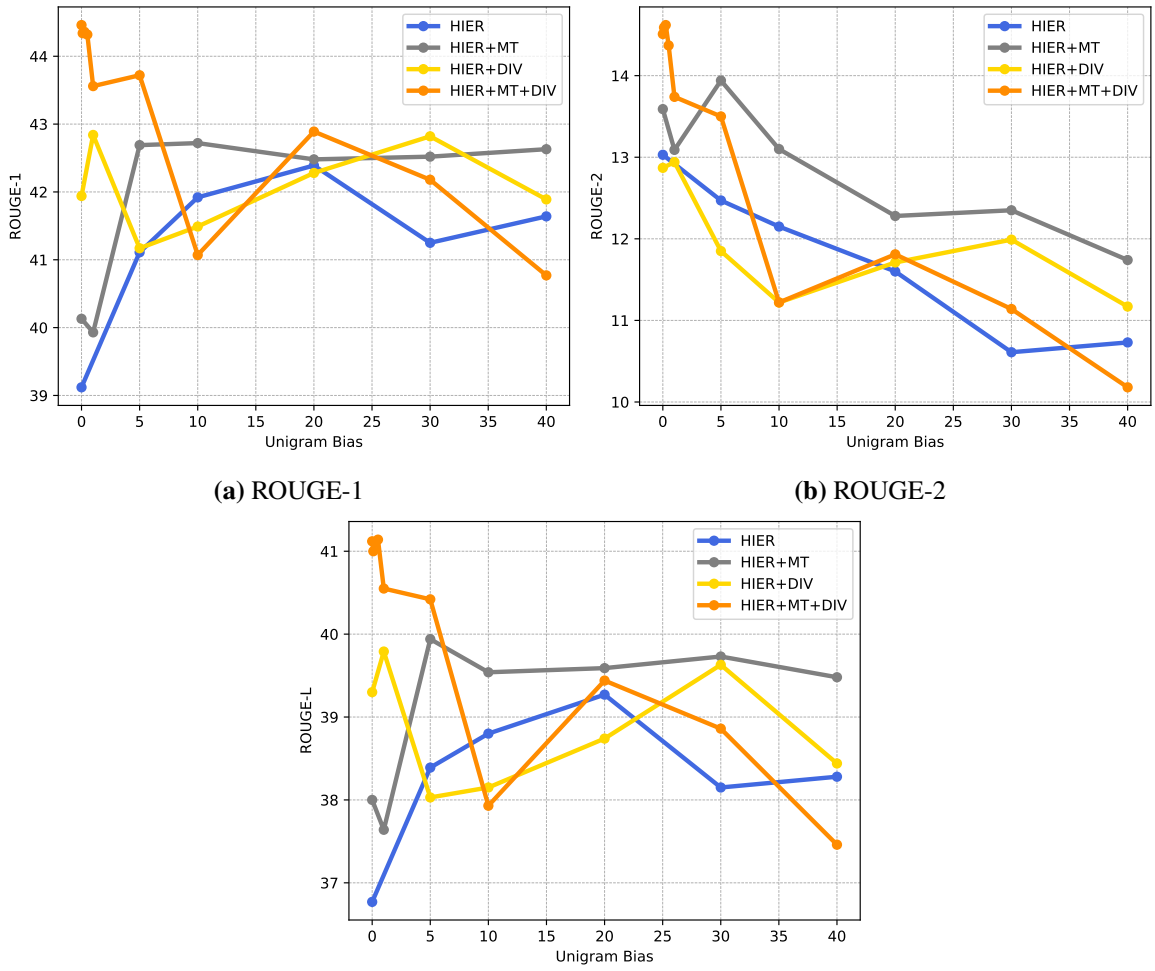
Table 5.3 Coverage loss and global variance loss (Section 5.3.1) on the test sets of AMI and CNN/DailyMain datasets. Note that *lower* loss means *higher* diversity.

The previous results show the effectiveness of unigram bias decoding on various diversity measures. Here, the summarization performance is investigated. First, in Figure 5.6, the

impact of unigram bias for $\beta \in [0.0, 40.0]$ is shown for 4 setups: the baseline hierarchical model (HIER), the inclusion of multi-task training (HIER+MT), the diversity training loss (HIER+DIV), and both (HIER+MT+DIV).

Since unigram bias decoding penalizes repeated unigrams, ROUGE-1 improves for all models as expected. Figure 5.5a shows that the inter-sentence diversity score flattens at $\beta = 20.0$, and it can be seen in Figure 5.6 that ROUGE-1 of the hierarchical setting also stops improving at around $\beta = 20.0$. This suggests a positive correlation between models being *diverse* and *higher* ROUGE-1. ROUGE-L, which measures the longest common sequence, also follows the same trend as ROUGE-1. However, we note that the increase in ROUGE-2 is less than the increase in ROUGE-1, and in one setting there is no gain from unigram bias.

The optimal β and summarization scores, from Figure 5.6, for each system are shown in Table 5.4. Additionally training the model with multi-task (HIER+MT) or diversity objectives yields performance gains. When explicitly optimizing the diversity objective during training (HIER+DIV), the model performance is similar to the baseline with unigram bias decoding (HIER with $\beta = 20.0$). When the diversity loss is optimized, a lower value of β is required to achieve optimal performance, suggesting that explicit diversity optimization during training and forcing diversity during decoding have a similar effect. The results also show that diversity optimization and unigram bias are complimentary, and the best performance is achieved in the HIER+MT+DIV with unigram bias decoding setting.



(a) ROUGE-1

(b) ROUGE-2

(c) ROUGE-L

Fig. 5.6 Variation of ROUGE F_1 scores against unigram bias for the hierarchical model (HIER), with multi-task training (HIER+MT), diversity loss (HIER+DIV) and both (HIER+MT+DIV).

Setting	β	ROUGE F_1 R-1	R-2	R-L
HIER	✗	39.12±2.45	13.03±1.58	36.77±2.28
	20.0	42.39±1.91	11.60±1.60	39.27±1.96
HIER+MT	✗	40.13±0.93	13.59±0.66	38.00±0.74
	5.0	42.69±1.03	13.94±0.86	39.94±0.94
HIER+DIV	✗	41.94±0.25	12.87±0.28	39.30±0.56
	1.00	42.84±1.10	12.94±0.50	39.79±1.83
HIER+MT+DIV	✗	44.46±0.11	14.51±0.12	41.12±0.13
	0.25	**44.36±0.58**	**14.62±0.21**	**41.10±0.25**

Table 5.4 Summarization Performance ROUGE F_1 on the AMI test set - Hierarchical Settings. All models are trained from scratch. The reported numbers are *mean* ± *standard deviation* from 3 runs with different data shuffling and different seeds.

As an ablation study, the comparison between the proposed diversity loss (+DIV) and the coverage mechanism (+COV) described in Section 5.3.1 is conducted. The results in Table 5.5 show that the proposed diversity loss, which is designed to operate at the sentence level, outperforms the coverage mechanism (+COV), which works at the word level. Note that both systems (+DIV and +COV) achieve lower ROUGE-2 scores, suggesting that the vanilla diversity loss could impact the fluency of the system.

Setting	ROUGE F_1 R-1	R-2	R-L
HIER	39.12	13.03	36.77
HIER+COV†	38.54	10.52	36.07
HIER+DIV	41.94	12.87	39.30

Table 5.5 Summarization Performance ROUGE F_1 on the AMI test set on the hierarchical model with a different diversity loss and without it. †The coverage mechanism [288] is applied at the word level after multiplying the sentence-level attention score to each word.

In addition, we investigate the effectiveness of the attention diversity method on CNN/DailyMail. As illustrated in Table 5.6, the HIER+DIV with unigram bias decoding system achieves higher ROUGE-1, ROUGE-2, ROUGE-L scores than the HIER system by 3.81%, 0.67%, and 3.29%, respectively. These results further confirm the effectiveness of diversity optimization.

Setting	β	ROUGE F_1 R-1	R-2	R-L
HIER	✗	31.45	12.60	29.54
	20.00	34.77	13.23	32.67
HIER+DIV	✗	32.54	12.75	30.57
	10.00	34.64	**13.36**	32.40
	20.00	**35.26**	13.27	**32.83**

Table 5.6 Summarization Performance ROUGE F_1 scores on the CNN/DailyMail test set. Note that the hierarchical training converged after 17 epochs, and we further train it with the diversity criterion and this additional training converged after 2 more epochs.

5.5.4 Impact of ASR on Summarization Performance

In Section 5.5.3, the experiments compared different setups using manual transcripts as the inputs. However, in practice, the first stage of spoken document summarization involves an ASR system, which may introduce transcription errors to input documents. Hence, this section investigates the impact of ASR on summarization performance. The best-performing summarization system (HIER+MT+DIV with β=0.25) is selected for this investigation.

Table 5.7 shows the performance of the summarization system when using ASR transcripts. When using the AMI ASR (ASR1) transcripts instead of the manual transcripts, the decrease in ROUGE is around 2-3%. This relatively small drop is likely because even at 30% WER, the sentence embedding similarity between a manual source and an ASR source is about 0.70-0.85% [359, 295]. When compared to current state-of-the-art meeting summarization methods (described Section 5.5.2), our system achieves higher ROUGE scores than UMCB [267], and when compared to TopicSeg (without visual signals) [160] our system achieves higher ROUGE-2 and ROUGE-L although lower ROUGE-1. Furthermore, when using transcripts with lower WER (ASR2), ROUGE scores are closer to those obtained from the manual transcripts, and yield higher ROUGE-2 and ROUGE-L scores than the state-of-the-art multi-modal TopicSeg+VFOA. [160].

Model	Input	ROUGE F_1 R-1	R-2	R-L
UMCB [267]	ASR1	37.86	7.84	13.72
TopicSeg [160]	ASR1	51.53	12.23	25.47
TopicSeg+VFOA [160]	ASR1	53.29	13.51	26.90
HIER+MT+DIV	ASR1	41.23±0.75	12.74±0.21	38.38±0.63
	ASR2	43.29±1.81	14.45±0.23	40.55±1.20
	MAN	44.36±0.58	14.62±0.21	41.10±0.25

Table 5.7 ASR1 = publicly available ASR transcripts (WER=36%), ASR2 = CUED generated transcripts (WER=20%), MAN =Manual transcripts. The reported numbers are *mean* ± *standard deviation* from 3 runs with different data shuffling and different seeds.

5.5.5 Auxiliary Task Performance

For the multi-task trained systems, it is possible to evaluate the performance of the system on the other training tasks, Dialogue Act classification (DialogueAct) and Extractive summarization (ExtractiveSum). Both tasks used the encoder of the hierarchical model. For the DialogueAct baseline $\mathcal{L}_{\text{DA}}$ was optimized, and for the ExtractiveSum baseline $\mathcal{L}_{\text{ext}}$ was optimized. For the HIER+MT setting, the weighting of the loss terms, λ_1 and λ_2 (in Equation 5.28) were both set to 10.0, and for the HIER+MT+DIV setting, λ_3 was set to 1.0. Table 5.8 shows that the summarization signal improves both dialogue act classification and extractive summarization labelling tasks. Our best dialogue act accuracy is comparable with the results achieved by Goo and Chen [89]. The diversity loss, in contrast, does not benefit these two tasks. This is expected as the diversity criterion aims to improve the diversity of the attention mechanism of the decoder, whereas for these two tasks, only the encoder of the summarization system is used.

DialogueAct	Accuracy	ExtractiveSum	F_1
HIER	63.42±0.82	HIER	54.47±3.75
HIER+MT	64.32±0.39	HIER+MT	56.05±1.61
HIER+MT+DIV	63.11±0.36	HIER+MT+DIV	56.06±2.39

Table 5.8 Performance on auxiliary tasks: Dialogue act classification (1-in-15) and Extractive summarization labelling (threshold of 0.5). The reported numbers are *mean* ± *standard deviation* from 3 runs with different data shuffling and different seeds.

5.5.6 Hierarchical Model for Sentence Filtering

Based on the sentence-level attention score of the hierarchical model (described in Section 5.4), this section investigates the impact of removing *redundancies* in source documents. The aim is to develop a method to filter redundancies at the sentence level to improve the overall summarization performance.

We hypothesize that an abstractive summarization system performs salient sentence selection implicitly as there exist redundant sentences that contain little or no information. Hence, simply filtering some sentences out may improve summarization performance. We use the following hierarchical models: (1) for AMI, the abstractive summarization model is HIER+MT+DIV with $\beta = 0.0$; (2) for CNN/DailyMail, the abstractive summarization model is HIER+DIV with $\beta = 20.0$. We compare three input scenarios as follows:

1. Full Input (baseline): All input sentences $\{S_1, ..., S_N\}$ are kept.

2. Keep_prob: Selection based on sentence score, v_i, defined in Equation 5.27 (using the teacher forced target) such that the remaining sentences have v_i above the threshold:

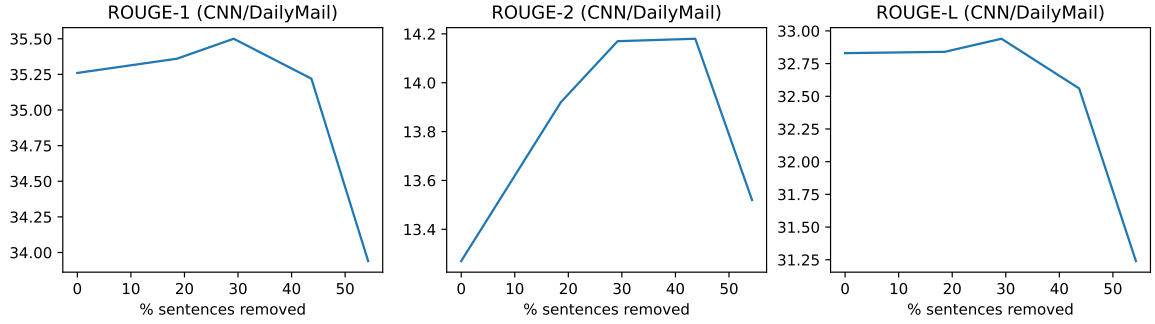
$$\hat{S} = \{S_1, ..., S_n\} = \arg\min_{\mathcal{S}} \sum_{i, S_i \in \mathcal{S}} v_i \quad \text{and} \quad \sum_{i, S_i \in \mathcal{S}} v_i > \texttt{keep\_prob} \tag{5.29}$$

3. First-K: Keep the same number of sentences as keep_prob, but the first K sentences.

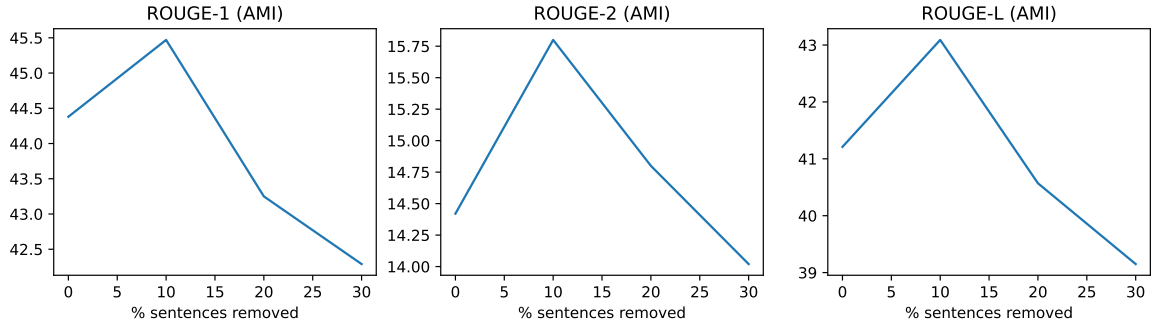
The results in Table 5.9 show that a small improvement in summarization performance can be obtained by simply performing sentence filtering using the score based on the sentence-level attention of the hierarchical model. In Figure 5.7a, the improvement is observed until around 30% of input sentences are removed. Lastly, we investigate how models trained using sentence-filtered data perform compared to models trained using full-input data. Figure 5.7b demonstrates that an improvement can be achieved by re-training the hierarchical models on filtered data. In conclusion, this section shows that an improvement in summarization performance can be obtained when sentence filtering is applied either at the inference stage only or at both stages. The next chapter will investigate sentence filtering in more detail, especially in the context of foundation models where the length of source documents becomes more critical.

Source Input	R1	R2	RL
Full Input	35.26	13.27	32.83
Keep_prob=0.90	**35.50**	**14.17**	**32.94**
First-K	35.17	14.03	32.59

Table 5.9 Summarization Performance on the CNN/DailyMail test set using different source inputs. The baseline system is the HIER+DIV system previously analyzed in Table 5.6.



(a) CNN/DailyMail



(b) AMI

Fig. 5.7 ROUGE score on CNN/DailyMail and AMI as more sentences are removed. Due to the size of the training data, sentence filtering is applied at both training and inference for AMI. For CNN/DailyMail, sentence filtering is applied at inference only.

5.6 Chapter Summary

This chapter investigated abstractive summarization in the context of training hierarchical RNN models from scratch. The proposed explicit sentence-level attention diversity at the training stage and the unigram bias decoding method were both shown to improve abstractive summarization using the hierarchical RNN model. It was demonstrated that a multi-task learning method, which incorporates dialogue act and extractive summarization

information, is useful for summarization. The hierarchical model with multi-task and attention diversity objectives (HIER+MT+DIV) was found to be the best configuration for the meeting summarization task. A further investigation demonstrated that for a real-world spoken document summarization using an ASR system with WER of around 20%, summarization performance close to that obtained from manual transcription can be achieved. Furthermore, this chapter applied the hierarchical model to sentence filtering, and it was shown that filtered inputs (i.e., a subset of input sentences) can yield better summarization results with and without model re-training.

Chapter 6

Abstractive Summarization with Foundation Models

Chapter 5 studied abstractive summarization with hierarchical RNNs where the models were either trained from scratch or trained on another summarization task before being fine-tuned to the target summarization task. However, RNNs have limitations such as vanishing gradients [107] and non-parallelizability during training [290, 226], limiting pre-training RNNs on large-scale data. Recently, there has been a shift in NLP approaches to using the transformer architecture, which can be parallelized during training and does not suffer vanish gradients. Transformer-based models with billions of parameters have then been pre-trained on billions of words. These pre-trained models are referred to as *foundation* models. After being fine-tuned on a target task, these foundation models achieve strong performance in a range of tasks [52, 159, 244]. Yet, there are challenges associated with the transformer architecture such as its quadratic dependencies with sequence length. This makes it challenging to apply foundation models to long-document summarization. This chapter will examine techniques for applying foundation models on long-document summarization tasks such as podcast transcripts and research articles.

First, Section 6.1 introduces the framework for applying foundation models to summarization, and it discusses the challenges, limitations, and general research questions being addressed. Section 6.2 discusses sentence filtering techniques for selecting salient information as the input to a foundation model at both training and inference stages. Section 6.3 focuses on the encoder of foundation models and investigates efficient attention mechanism. Section 6.4 focuses on the encoder-decoder attention and proposes a method to make it more efficient by exploiting sparsity in attention weights. Section 6.5 presents experiments, which include the comparison between RNN-based and foundation model based approaches, sentence filtering, efficient transformers, and ensemble of summarization models.

6.1 Challenges in Long-Span Summarization

Foundation models or pre-trained models are typically trained using self-supervised learning, e.g. masked language modelling or de-noising tasks [52, 159, 14], so they are generally unfinished and should not be used directly on a target task. The standard approach in using any foundation model on a sequence-to-sequence task is *fine-tuning* (i.e., adapting) the model to the specific task using the teacher-forced maximum likelihood criterion (Section 2.2).[1] With regard to the architecture, encoder-decoder and decoder-only models are used in abstractive summarization. This is because the generation process is *autoregressive*, so only these architectures are suitable. The encoder-decoder architecture, such as BART [159] and T5 [244], is typically used in abstractive summarization. This is because the encoder-decoder architecture adopts a bidirectional encoder, which improves the information flow from the source document to the summary.

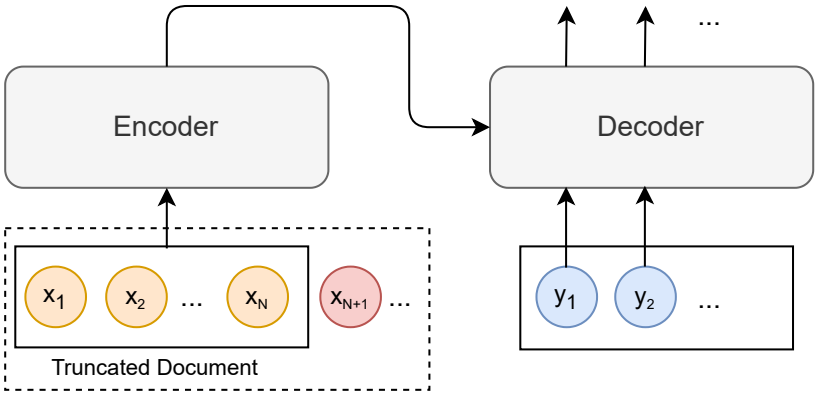


Fig. 6.1 Fine-tuning a pre-trained transformer on a downstream task such as summarization. The source document is truncated to the maximum input tokens of the pre-trained encoder model, or to the limit of the available VRAM (e.g., GPU memory).

One issue with transformer models is that the use of self-attention does not scale well in terms of memory and compute requirements as the input length grows (Section 3.1.4). Thus, for long-document summarization, it can be challenging to fine-tune a large foundation model. A vanilla approach could result in source documents being truncated as illustrated in Figure 6.1. Therefore, in the following two sections, we address long-span dependencies in summarization in two directions: explicit content selection to reduce the number of input tokens into the model (Section 6.2), and efficient modelling using local attention to increase the maximum length that the model can handle (Section 6.3). These two sections focus on

[1] Recently, large language models (LLMs) have been shown to emergent properties where the models can perform tasks in a zero- or few-shot setup without fine-tuning. Emergent abilities are discussed in Section 3.2.1.

handling long inputs and efficient encoder, while the subsequent section (Section 6.4) will focus on the efficiency of the encoder-decoder attention.

6.2 Sentence Filtering

In the previous chapter (in Section 5.4 and Section 5.5.6), we investigated RNN-based summarization models. With an RNN-based model, we could take any arbitrary length sequence because applying an RNN would yield a representation, and it has a linear complexity (time and compute) with the sequence length. With a transformer-based model, however, its fixed positional encoding could result in limitations in the maximum length in addition to the quadratic complexity with the sequence length of self-attention. Hence, sentence filtering can be important in using foundation models.[2]

A form of sentence filtering based on the attention of the hierarchical model was introduced and investigated in the previous chapter (in Section 5.4 and Section 5.5.6). This section aims to firstly extend sentence filtering to improve fine-tuning foundation models with a limited length input on long-document summarization tasks (e.g., to improve document truncation shown in Figure 6.1), and secondly study other forms of sentence filtering.

It has been shown that a better content selection improves in news summarization [24, 82, 114], multi-document summarization [168, 165], and scientific article summarization [231]. Based on this motivation, this section seeks to tackle the excess length by sentence filtering.

We will investigate the upper-bound performance by making use of the *oracle* information (i.e., using ground-truth summary). Oracle information also provides training signals about which source sentences are the most informative with respect to the ground-truth summary. However, oracle information is not available at the inference stage where sentence filtering has to be performed without the ground-truth summary. Following extractive summarization, a model-based approach is considered for sentence filtering. It should be noted that although at training, an oracle information approach is expected to yield the best result (lowest training loss), using oracle information could result in a mismatch between the training and inference stages. Therefore, this work will investigate the performance at inference time when the model is trained with and without oracle information. Table 6.1 summarizes when each sentence filtering method is applicable.

[2]Sentence filtering is also referred to as content selection.

Approach	Stage	
	Training	Inference
Truncation	✓	✓
Oracle (§6.2.1)	✓	✗
Model-based (§6.2.2)	✓	✓

Table 6.1 Stage when sentence filtering approach can be applied. During training, ground-truth summaries are available, so filtering methods can be either: ground-truth-based (model-free), which is also referred to as oracle and model-based. Ground-truth-based methods cannot be used at the inference time, while model-based methods can be applied at both phases.

Let the subscript (i, j) denote the position of the j-th word in the i-th input sentence, the full source document $X = \{S_1, ..., S_i, ..., S_{N_1}\} = [\underbrace{x_{1,1}, x_{1,2}, x_{1,J_1}}_{\text{sent } 1}, ..., \underbrace{x_{i,1}, x_{i,J_i}}_{\text{sent } i}, ..., \underbrace{x_{N_1,1}, x_{N_1,J_{N_1}}}_{\text{sent } N_1}]$ where N_1 is the number of sentences in X and J_i is the number of words in the i-th sentence,[3] and $Y = y_{1:M}$ is the summary. A sentence filtering method re-ranks, truncates, and sorts the input document X to get a filtered input $\tilde{X}$ as follows,

$$\bar{X} = \{S_{r_1}, S_{r_2}, S_{r_3}, ..., S_{r_R}\} \tag{6.1}$$

$$\tilde{X} = \texttt{SortOrig}(\texttt{TruncateN}(\bar{X})) \tag{6.2}$$

where r_i is the index of the sentence of rank i, the `TruncateN` operation filters $\bar{X}$ such that the total number of words (or tokens) is less than N, and `SortOrig` retains the original sentence order. The following ranking methods are considered,

- Truncation (TRC):

$$r_k = k \tag{6.3}$$

- Model-based: Given the score f of model parameterized by $\boldsymbol{\phi}$,

$$r_k = \{i \in N_1 : f_{\boldsymbol{\phi}}(i|X) \text{ is ranked } k\text{-th}\} \tag{6.4}$$

- Oracle (ORC): Given the ground-truth summary y and similarity measure `sim`,

$$r_k = \{i \in N_1 : \texttt{sim}(S_i, Y) \text{ is ranked } k\text{-th}\} \tag{6.5}$$

[3] This notation follows the notation in Section 5.2.1 with the only difference is that this chapter uses N_1 to denote the number of sentences, and N is instead used for the total number of words in all input sentences.

Note that the model-based method is similar to extractive summarization (Section 4.1). The slight difference is that extractive summarization seeks to find a subset of sentences that yields the best summary, while the method in this section seeks to rank sentences and pass as many sentences as the abstractive model could process. Hence, existing extractive summarization models are applicable as a model-based method where they can rank original sentences. Section 6.2.2 will describe model-based sentence filtering in detail.

6.2.1 Oracle Sentence Filtering

Oracle (or ground-truth-based) methods require ground-truth targets. Oracle methods have the advantage that they utilize ground-truth summaries in identifying salient sentences and these methods are applicable in training. However, the disadvantage is that they are *not* applicable during inference, and their performance during inference only represents an upper-bound performance level. Existing oracle methods include using the use of ROUGE-2 recall in Liu et al. [165] or the average of ROUGE-1,2,L recall in Pilault et al. [231].

In this work, we use ROUGE-2 recall as the similarity measure `sim` in oracle methods (in Equation 6.5). First, we retain only the input sentences with positive `sim`. We found that the number of sentences with positive `sim` is low at 21.3% of the total number of sentences on average on the Podcast dataset. This corresponds to 56% of training instances being shorter than the BART input span of 1024. This percentage represents the percentage of documents that are aggressively extracted by the oracle method. As the no-padding oracle method (Oracle-no-pad) is highly *aggressive*, potentially preventing the downstream summarizer from learning complex abstraction, we propose variants of oracle methods to extend the Oracle-no-pad selected input to the maximum input span N tokens. Two proposed variants shown in Figure 6.2 are:

- Oracle-pad-lead: Pad by leading unselected sentences and keep original order.
- Oracle-pad-rand: Pad by random unselected sentences and keep original order.

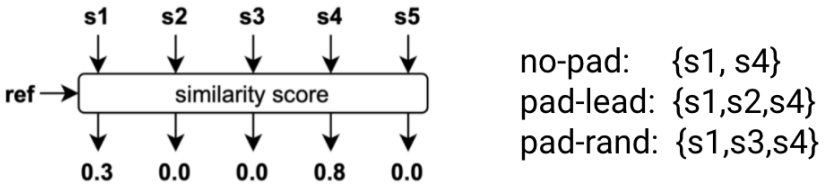


Fig. 6.2 Illustration of the oracle sentence filtering methods considered: no-pad, pad-lead, and pad-rand. $\{S_1, S_2, S_3, S_4, S_5\}$ are input sentences, Y (`ref`) is the ground-truth summary, and similarity score is ROUGE-2.

6.2.2 Model-based Sentence Filtering

To process long input sequences entirely, we consider RNN, whose memory requirement grows linearly with the sequence length, and hierarchical architectures which have been shown effective for long sequence-to-sequence tasks [39, 160] as well as Section 5.5.6.

Through a least-squares regression with 20 samples, the hierarchical RNN model (described in Section 5.2) requires VRAM during training given the target length of 144 as follows,

$$\texttt{memory} = 0.83 + B(3.96 \times 10^{-5} + 3.33 \times 10^{-5} N_2) N_1$$

where N_1 is the number of input sentences, and N_2 is the maximum number of words in a sentence, and B is batch size. By setting $N_1 = 1000$ and $N_2 = 50$, only 2% of podcast data exceeds this limit, while taking GPU memory to only 2.53 GB for a batch size of 1. Thus, this finding confirms that this model can cover long sequences.

Previous model-based methods treat content selection as extractive labelling and create labels heuristically [231], while Section 5.4 has used an encoder-decoder attention mechanism. To utilize both of these in one framework, we propose a Multi-task Content Selection (MCS) method where we train the hierarchical encoder-decoder with an attention mechanism and a classification layer on top of the encoder. First, the model is trained on sequence-to-sequence abstractive summarization objective as follows,

$$\mathcal{L}_{\text{seq2seq}} = -\sum_{m=1}^{M} \log P(y_m | y_{1:m-1}, X) \tag{6.6}$$

Second, we create binary labels as follows: for the i-th sentence S_i and reference summary Y, the label z_i is 1 if $\texttt{sim}(S_i, Y) > \texttt{threshold}$; else z_i is 0. The similarity measure $\texttt{sim}$ is ROUGE-2 recall and the threshold is set to 0.0 The extractive labelling task objective is defined as,

$$\mathcal{L}_{\text{label}} = -\sum_{i=1}^{N_1} (z_i \log \hat{z}_i + (1 - z_i) \log(1 - \hat{z}_i)) \tag{6.7}$$

$$\hat{z}_i = \text{sigmoid}(\mathbf{w}_{\text{cls}}^T \mathbf{h}_i + b_{\text{cls}}) \tag{6.8}$$

where $\mathbf{h}_i$ is the sentence-level encoder output associated with sentence i, and $\mathbf{w}_{\text{cls}}, b_{\text{cls}}$ are the parameters of the classification layer. Thus, the MCS training loss is defined as follows:

$$\mathcal{L}_{\text{MCS}} = \lambda \mathcal{L}_{\text{label}} + (1 - \lambda) \mathcal{L}_{\text{seq2seq}} \tag{6.9}$$

At inference, there are two modes: (i) standard abstractive summarization, e.g., via beam search; (ii) ranking input sentences via labelling score and sequence-to-sequence attention

score. The latter is how MCS is used during inference.[4] For sentence i, the scores are:

$$\text{score}_{i,(\text{label})} = \hat{z}_i \quad \text{and} \quad \text{score}_{i,(\text{seq2seq})} = \sum_{m=1}^{M} \alpha^{\text{s}}_{m,i} \tag{6.10}$$

where $\alpha^{\text{s}}_{m,i}$ is the sentence-level attention weight at decoder step m over input sentence i. Since the scores are on different scales, rather than using the scores defined in Equation 6.10, we simply rank the scores, and then normalize the score ranks into the range $[0.0, 1.0]$. Let nscore denote the normalized ranking score, the MCS inference score is:

$$f_{\boldsymbol{\phi}}(i|X) = \text{nscore}_{i,(\text{label})} + \text{nscore}_{i,(\text{seq2seq})} \tag{6.11}$$

6.3 Efficient Encoder Attention

To fine-tune pre-trained transformer models on summarization tasks with potentially long input sequences, one method is to modify the architecture. In this section, we will delve into local attention design with practical memory profiling and computation complexity analysis. Moreover, **Local attention BART (LoBART)** will be introduced in Section 6.3.1.

6.3.1 Local Attention

BART [159] is used as an example of encoder-decoder foundation models. Following the local attention window attention in Sparse Transformer [27] and Longformer [10], *local attention* (illustrated in Figure 6.3) is applied to BART's encoder to tackle the quadratic dependency. Following the self-attention discussed in Section 2.1.4, the standard self-attention mechanism computes the representation for position i shown in Figure 6.3a:

$$\mathbf{h}_i = \sum_{j=1}^{N} \left(\mathbf{q}_i \cdot \mathbf{k}_j\right) \mathbf{v}_j \tag{6.12}$$

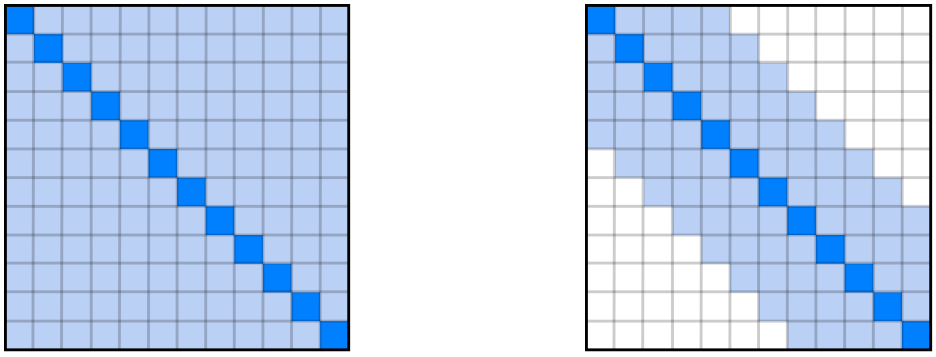
Local attention span limits the receptive field to a local neighbourhood [223], and modifies the attention mechanism as shown in Figure 6.3b to:

$$\mathbf{h}_i = \sum_{j=\max\left(1, i-\frac{W}{2}\right)}^{\min\left(i+\frac{W}{2}, N\right)} \left(\mathbf{q}_i \cdot \mathbf{k}_j\right) \mathbf{v}_j \tag{6.13}$$

[4] In practice, we run beam search decoding of width 4, and we obtain the attention score from the top beam.

where N is the input sequence length and W is the size of the local window. Given a self-attention mechanism, this is applied to every position from $i = 1, ..., N$, so it reduces the computational and memory costs from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \cdot W)$.

Local attention is motivated by that the information is likely localized. Also, a standard pre-trained model (e.g., BART) is used instead of other efficient encoder-decoder architectures because pre-trained weights of standard models are more readily available (which was especially true at the time of conducting the experiment), and they have also been fine-tuned on various target tasks, including text summarization.



(a) Full Attention **(b)** Local Attention (with $W = 9$)

Fig. 6.3 Self-attention patterns that are investigated in this work. Other forms of fixed attention patterns are provided in Section 3.1.5.

In addition to modifying the attention mechanism, other *complementary* techniques for reducing memory in training could be considered: (i) gradient-checkpoint where a subset of intermediate values in the computation graph are cached, and the rest are re-computed during backpropagation [22], but this requires changes to optimization and leads to longer training time; (ii) half/mixed-precision training [197] where the precision (discussed in Section 3.1.4) of model parameters is reduced such as from 32-bit floating point to 16-bit floating point. This reduction in the precision may result in lower performance; (iii) model parallelism with micro-batching [118], but this method requires multiple accelerators.

Understanding Local Attention

To understand whether the assumption that the information is localized (hence the suitability of local attention), this experiment seeks to empirically investigate the distance characteristic of the attention mechanism of BART's encoder. First, we define the mean attention distance in a particular layer and head to study the characteristics of self-attention in BART's encoder

as follows,

$$D_\alpha = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{j=1}^{N}\alpha_{i,j}\times|i-j|\right) \tag{6.14}$$

where $\alpha_{i,j}$ is the attention weight of position i attending to position j such that $\sum_{j=1}^{N}\alpha_{i,j}=1$. This measure corresponds to the average distance of self-attention. If the attention weight is assumed uniform, $D_\alpha = \frac{N^2-1}{3N}$. For example, if $N=1024$, $D_\alpha = 341$ for uniform attention.

Based on BART, Figure 6.4 shows that most layers have a shorter mean distance than uniform-attention D_α, supporting that the information is more localized. The mean distances of differently initialized BART models computed on the podcast data also show that the attention mechanism is learned during the pre-training stage as there is little variation after the pre-training stage. As illustrated in Figure 6.4, the average attention distance D_α of the BART model is around 250-350 tokens. This suggests the window size W should be designed to be above 700, allowing half local attention window $W/2$ to be greater than 250-350 to effectively match BART and to exploit transfer learning more efficiently.

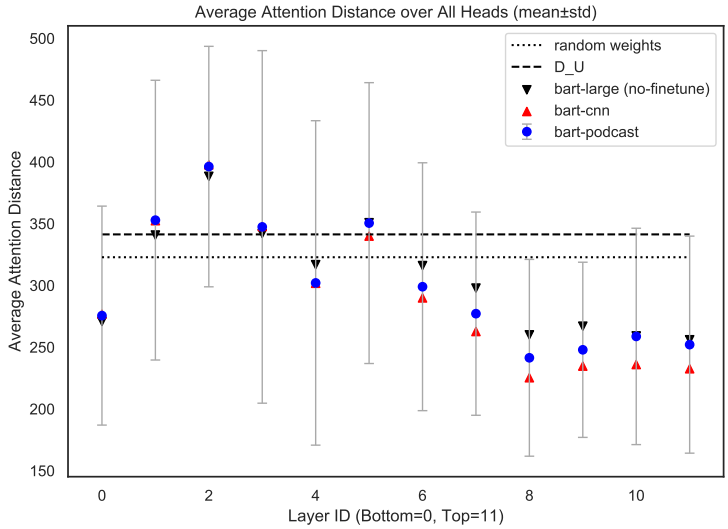


Fig. 6.4 The average mean distance across multi-heads for each layer. The average mean distance of the random weight model is slightly lower than uniform D_α as some inputs are shorter than 1,024.

Extending Maximum Input Span

As described in Section 2.1.4, transformers have a position encoding. For example, BART has a learned embedding as its positional encoding, so the number of input tokens is limited to the size of this positional encoding which is 1024 for BART. Previous work [169] handled this limitation by extending a learned embedding of BERT by adding a position embedding of extended positions. These new embeddings are initialized randomly and fine-tuned with other

parameters in the encoder. This work, on the other hand, extends the positional embedding of BART (and LoBART) beyond 1,024 by copying BART positional embedding with flipping to allow a smoother transition as shown in Figure 6.5 where the positional embedding is extended from length 1024 to 4096.

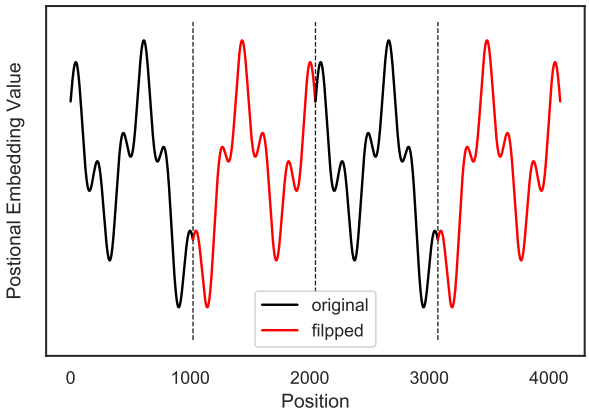


Fig. 6.5 Extending position embedding of length 1024 to 4096.

Local Attention BART (LoBART)

As an example of an efficient foundation model for long-document summarization, this work proposes local attention BART (LoBART) which is based on the BART-large system with local attention in the encoder. Its positional embedding is initialized as described in the previous subsection. In the subsequent sections, LoBART(nk) with local attention of width W refers to the variant where the positional embedding is extended to $n \times 1024$. The number of parameters in LoBART(nk) is, therefore, $406 \times 10^6 + 50264 \times (n-1) \times 1024$ where 50264 is the size of the vocabulary and 1024 is the size of the representation.

6.3.2 Memory and Time Analysis

It has been known that the memory and compute complexity of transformers is *quadratic* with the sequence length. However, in encoder-decoder architectures, the exact dependencies on input length N, target length M, and batch size B are less well understood. This is particularly important in long-span sequence-to-sequence tasks because large memory or compute requirements could make training or fine-tuning impractical. The memory requirement is also crucial in determining whether a design configuration is feasible given a hardware accelerator (e.g., VRAM of a GPU). Thus, the following part will investigate these dependencies and show the trade-off between the size of the input span and the efficient attention modification.

Memory Profiling through Regression Analysis

To investigate the memory profiling of BART and its local attention, let input length = N, target length = M, local attention width = W, and batch size = B. Through regression analysis for an encoder-decoder architecture such as BART, the memory required in training is:

$$\texttt{Memory} = c_1^b + B(c_2^b M + c_3^b N + c_4^b MN + c_5^b M^2 + c_6^b N^2) \tag{6.15}$$

The term c_1^b depends on only the model size and optimizer, and it is constant. The remaining terms are activation memory associated with the activation outputs cached for backpropagation, and they grow with N, M, and B.

To perform regression analysis, the memory requirement for each (B,N,M) setup is measured. In practice, this is done by selecting B, N and M and then BART is trained and its maximum memory usage on a GPU is measured. The process can be repeated for different sets of (B,N,M) values, and a regression analysis can then be performed to obtain the coefficients in Equation 6.15 using the MATLAB curve fitting tool.

Based on the **BART** system (BART-large), the experiments include $N \in$ [64, 256, 512, 1024, 2048, 3000] and $M \in$ [36, 72, 144, 288, 576] using a batch size of 1, resulting in a total of 30 samples. The least-squared regression of BART's VRAM (in Equation 6.15) yields $R^2 = 1, \text{RMSE} = 0.026$, and the coefficients $c_1^b = 6.054$, $c_2^b = 1.594 \times 10^{-3}$, $c_3^b = 8.192 \times 10^{-4}$, $c_4^b = 1.418 \times 10^{-6}$, $c_5^b = 1.077 \times 10^{-6}$, $c_6^b = 1.456 \times 10^{-6}$. As an example, Table 6.2 shows system-independent[5] regression results for the memory in training BART. It is apparent that as N grows the dominant term is $c_6^b N^2$, which is associated with the encoder self-attention. Thus, this finding confirms the motivation in modifying self-attention only on the encoder side.

Term	c_1^b	$c_2^b M$	$c_3^b N$	$c_4^b MN$	$c_5^b M^2$	$c_6^b N^2$
GB	6.05	0.23	0.84	0.21	0.02	1.53

Table 6.2 BART's Memory Profile (N=1024, M=144).

By introducing local self-attention of width W to BART (in short, we refer to this setup as **LoBART**), the memory in training becomes:

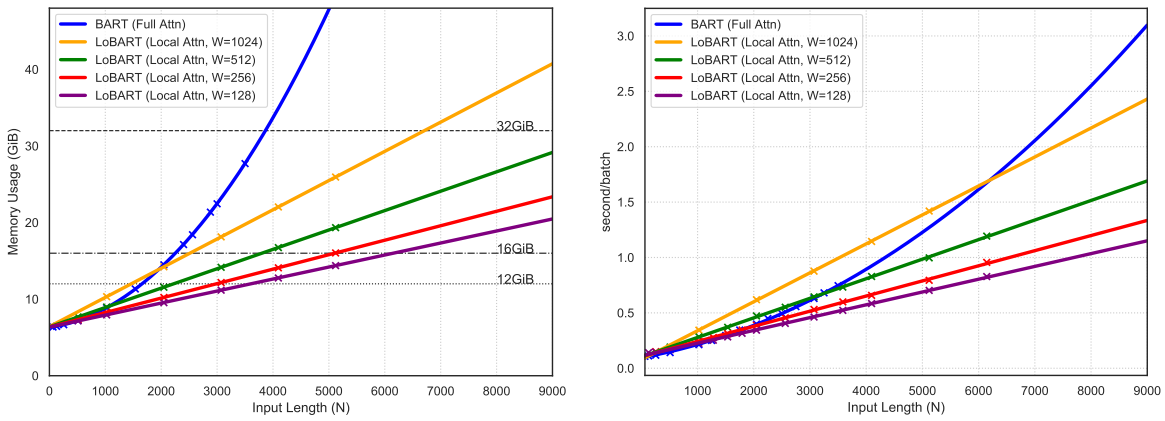
$$\texttt{Memory} = c_1^l + B(c_2^l M + c_3^l N + c_4^l MN + c_5^l M^2 + c_6^l NW) \tag{6.16}$$

[5] system-independent across hardware and machines; albeit implementation-dependent. This analysis is based on widely used PyTorch and Huggingface implementation.

Similarly, the experiments include $N \in [512, 1024, 2048, 4096]$, $M \in [100, 200, 400]$, and $W \in [32, 128, 512]$ using a batch size of 1, resulting in a total of 36 samples. The least-squared regression of LoBART's VRAM (in Equation 6.16) yields RMSE $= 0.010$, and the coefficients $c_1^l = 6.104$, $c_2^l = 1.443 \times 10^{-3}$, $c_3^l = 1.032 \times 10^{-3}$, $c_4^l = 1.487 \times 10^{-6}$, $c_5^l = 1.277 \times 10^{-6}$, $c_6^l = 2.503 \times 10^{-6}$.

The activation memory is now dominated by $c_6^l NW \times B$, where $c_6^l \approx 1.72 c_6^b$. Thus, we highlight that once $W > 0.58N$, LoBART likely no longer reduces memory. Note that we also tried incorporating the terms N^2 and W in the least-squared regression analysis, but their resulting coefficients are small, making both terms negligible. This is expected as quadratic self-attention is replaced by local attention of width W, and the width W only determines the receptive field of each and every position in N, resulting in the NW term.

The memory requirement for training BART and LoBART in Figure 6.6a enables us to choose an operating point. For example, the experimental setups shown in Table 6.10 are constrained by the maximum GPU memory of 32 GB illustrated in Figure 6.6a, e.g., for full attention, $N = 4096$ results in a GPU out of memory issue.



(a) Memory Complexity **(b)** Time Complexity

Fig. 6.6 Operating points for a batch size of 1 and summary length of 144 tokens. (i) Section 6.3 studies local attention to reduce the complexity from quadratic to linear. As the local attention width W decreases, the gradient of linear complexity decreases. (ii) Section 6.2 studies sentence filtering (i.e., content selection) to move an operating point to the left.

A further note on Model, Optimizer, and Activation

The *constant* term in Equation 6.2, $c_1^b = 6.054$ GB, is independent of batch size, system, or implementation (given the same floating-point precision). This term comprises model and optimizer memory as follows (in 32-bit floating point, 1 variable takes 4 bytes):

1. Model Parameter: BART has 406,290,432 parameters, yielding $406290432 \times 4 = 1.625 \times 10^9$ bytes $= 1.51$ GB.

2. Model Gradient: Each parameter has one corresponding gradient variable, e.g. `.grad` in PyTorch. Thus, this also occupies 1.51 GB.

3. Optimizer: Adam optimizer [136] stores first moment and second moment for each and every model parameter, hence, taking 3.02 GB.

4. Activation: The terms corresponding to $c_2^b, ..., c_6^b$ are associated with activation buffers cached for computing gradients in backpropagation. These terms grow linearly with batch size. The dominant term $c_6^b N^2 B$ grows quadratically with the input length N, motivating the encoder's local self-attention design. Chen et al. [22] proposes a method to save the activation memory by only caching buffers of a subset of layers, and re-computing the rest dynamically during backpropagation. This results in repeated computations and more training time.

Time Analysis BART & LoBART

Unlike memory, the time requirement is both system and implementation-dependent. In this analysis, we show the results using our infrastructure consisting of a 32 GB V100 GPU and 32-core Intel Xeon 4215R CPU (3.20GHz). We compute the time required for 50 forward and backward passes in 12 settings for each model configuration. Similar to the memory analysis, we perform least-squared regression where the results are shown in Figure 6.6b. It can be seen that although LoBART reduces memory requirement, when it comes to time requirement, LoBART is only comparable to BART. This is due to the implementation of local self-attention that involves additional processes such as chunking.

6.4 Efficient Encoder-Decoder Attention

The previous section (Section 6.3) showed that time and memory are dominated by the encoder's self-attention. Models such as LoBART adopt local attention in its encoder to mitigate this bottleneck, while keeping the original decoder. Training is fast because attention is highly parallelizable. However, during inference, the decoder uses its histories, becoming less parallelizable. To understand when the decoder might become a bottleneck, we fix the input length N and measure the computational time as a function of the target length M,

$$\texttt{Time} = \bar{c}_1 + \bar{c}_2 M + \bar{c}_3 M^2 \tag{6.17}$$

in three operating modes as follows:

1. Forward+Backward, e.g., at training time
2. Forward only, e.g., forward-pass where the input to the decoder is provided in advance
3. Inference, e.g., the decoder using its own back histories as the input.

Through a regression analysis method, the results in Table 6.3 show that the relative decoder cost during inference is almost one order of magnitude larger than that during training, e.g. forward+backward or forward only. The regression analysis in Table 6.3 also reveals that the encoder-decoder attention cost is greater than the decoder self-attention cost. Therefore, this motivates the study on efficient **encoder-decoder attention** where the goal is to reduce the computational time or the complexity of the encoder-decoder attention.

Mode	$\bar{c}_2/\bar{c}_1$ (10^{-3})	$\bar{c}_3/\bar{c}_1$ (10^{-6})
Forward+Backward	1.08	0.17
Forward only	1.14	0.25
Inference	**9.96**	1.30

Table 6.3 Empirical computational time as a function of the target length M where $\bar{c}_1, \bar{c}_2, \bar{c}_3$ are the coefficients in Equation 6.17. The analysis is based on the HuggingFace implementation of BART [316] and the input length is 1024.

6.4.1 Sparsity in Encoder-Decoder Attention

In summarization tasks, we hypothesize that the model might perform content selection *implicitly*. If this hypothesis holds true, it could result in sparsity in the encoder-decoder attention. For example, the attention weights could concentrate on a few salient input sentences. Based on this hypothesis, this section will investigate it in detail.

First, let M = the summary length, N = the input length, N_1 = the number of input sentences, and N_2 = the average number of words in a sentence (such that $N = N_1 N_2$). The standard encoder-decoder attention in Equation 6.18 (scaling factor omitted) where $\mathbf{Q} \in \mathcal{R}^{M \times D}$ and $\mathbf{K}, \mathbf{V} \in \mathcal{R}^{N \times D}$ has the complexity $\mathcal{O}(MN) = \mathcal{O}(MN_1N_2)$. Note that we fix the representation dimension D, so D is omitted in our complexity notation.

$$\mathbf{A} = \text{softmax}(\mathbf{Q}\mathbf{K}^T)\mathbf{V} \tag{6.18}$$

If the attention is concentrated on some r sentences,[6] by selecting appropriate r, the speed of the encoder-decoder attention can be improved by a factor of N_1/r on average. This is equivalent to approximating Equation 6.18,

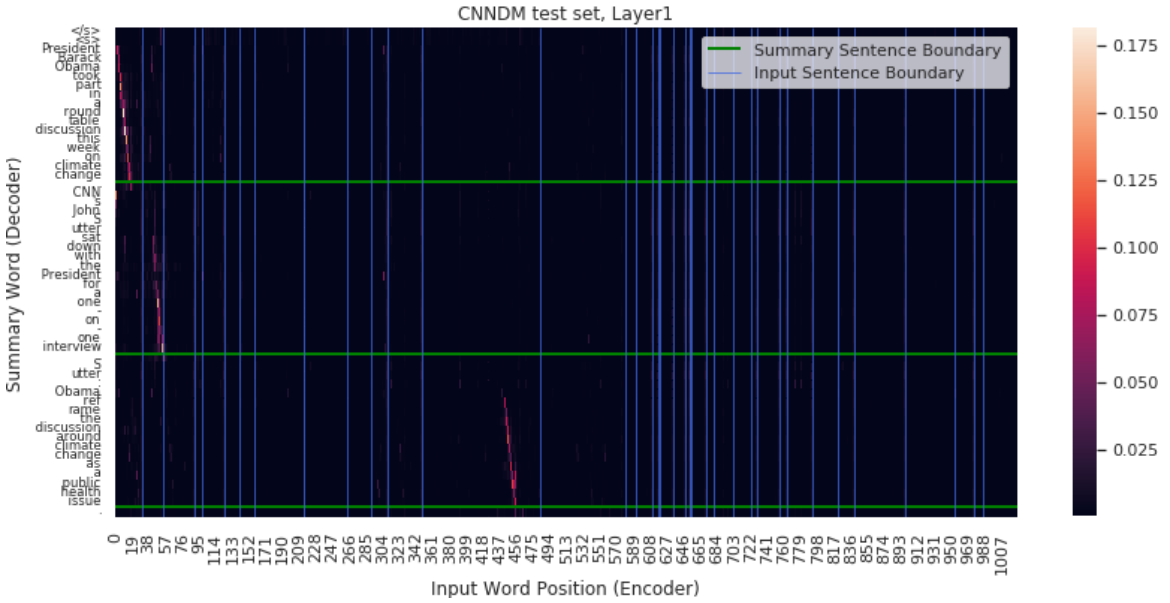
$$\mathbf{A} \approx \hat{\mathbf{A}} = \text{softmax}(\mathbf{Q}\hat{\mathbf{K}}^T)\hat{\mathbf{V}} \tag{6.19}$$

where $\hat{\mathbf{K}}, \hat{\mathbf{V}} \in \mathcal{R}^{rN_2 \times D}$, resulting in $\mathcal{O}(MrN_2)$.

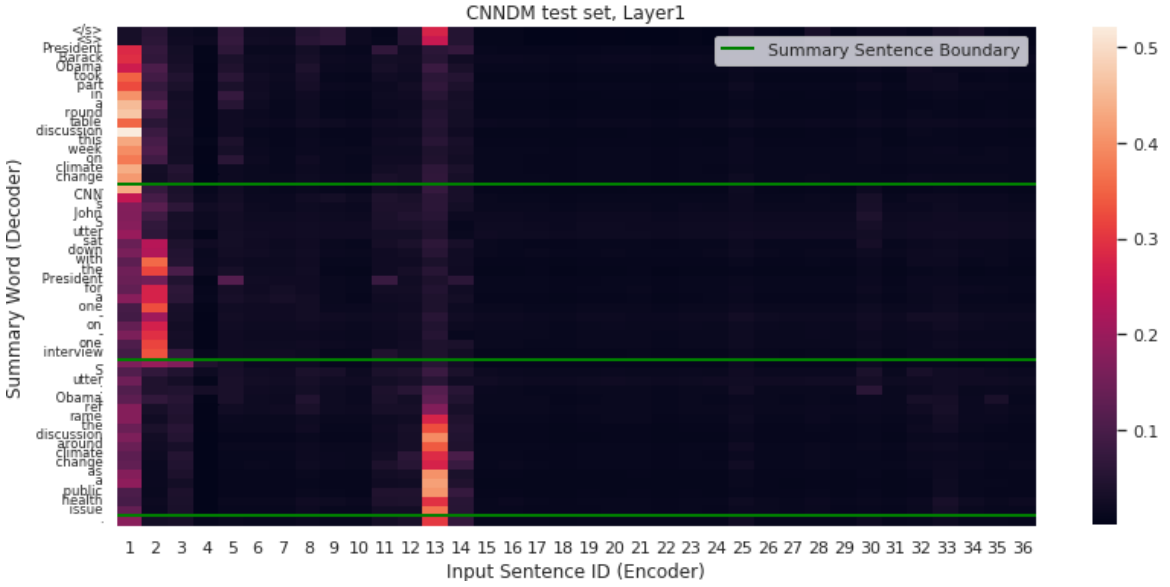
Word-level and Sentence-level Attention Weight Plots

To understand the encoder-decoder attention on summarization tasks empirically, we average the attention weights over all heads and plot the results in Figures 6.7, 6.8, 6.9, and 6.10. For instance, Figure 6.7 shows that the decoder attends particularly to input sentences #1,#2,#13 in the summary generation. Compared to Figure 6.7, Figure 6.8 shows a wider spread of attention over sentences in a more abstractive task. When using LoBART, Figure 6.9 and Figure 6.10 show a similar trend of the sparsity to BART scenarios. These figures also explain Figure 6.11a (in Section 6.4.2) that $\sum_i^{\mathcal{I}_m^r} \alpha_{m,i}^{\mathtt{s}}$ is only low or moderate because most sentences get assigned some attention weights, despite being non-salient.

[6] Motivated by the observations shown in Figures 6.7, 6.8, 6.9, and 6.10.

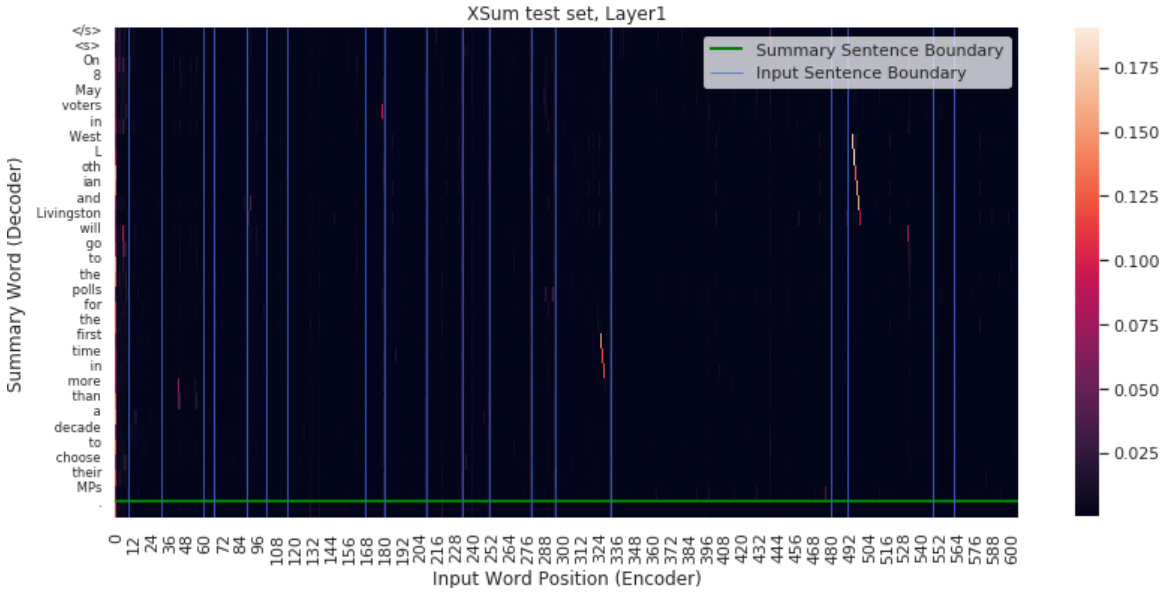


(a) Word-Level

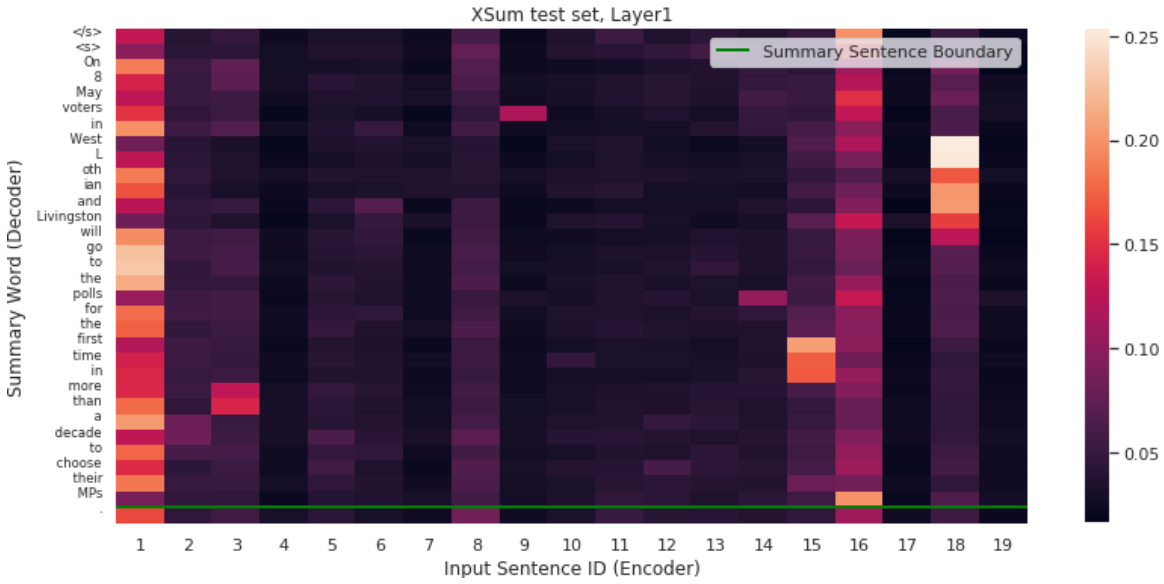


(b) Sentence-Level ($\alpha^{s}_{m,i}$)

Fig. 6.7 Example of BART's encoder-decoder attention evaluated on CNNDM test set. Blue vertical lines indicate input sentence boundaries and green horizontal lines indicate target sentence boundaries.

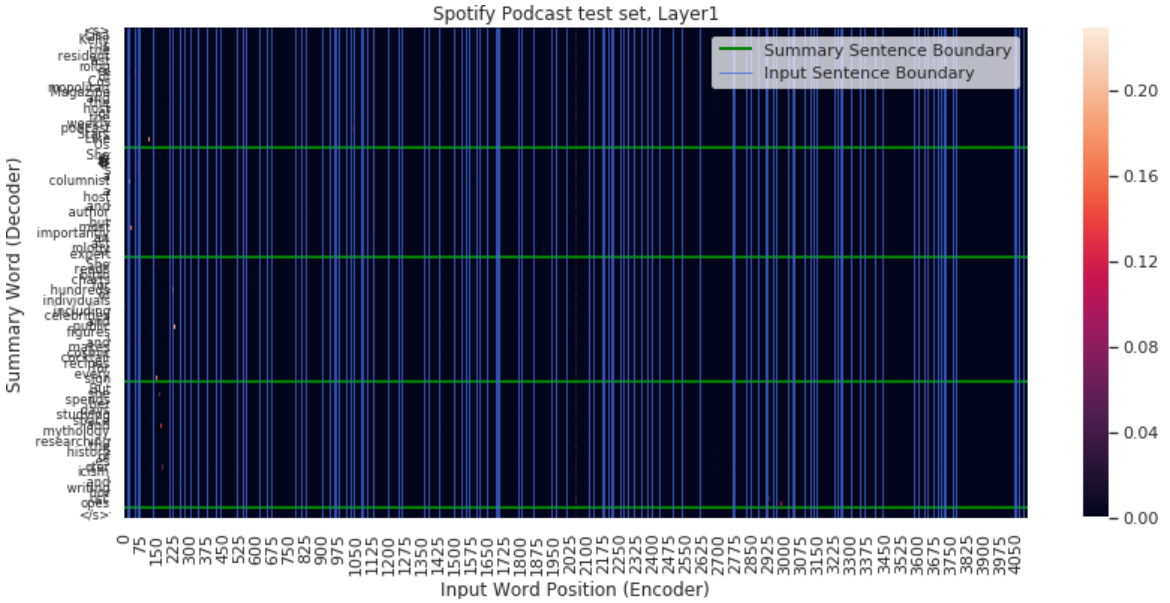


(a) Word-Level

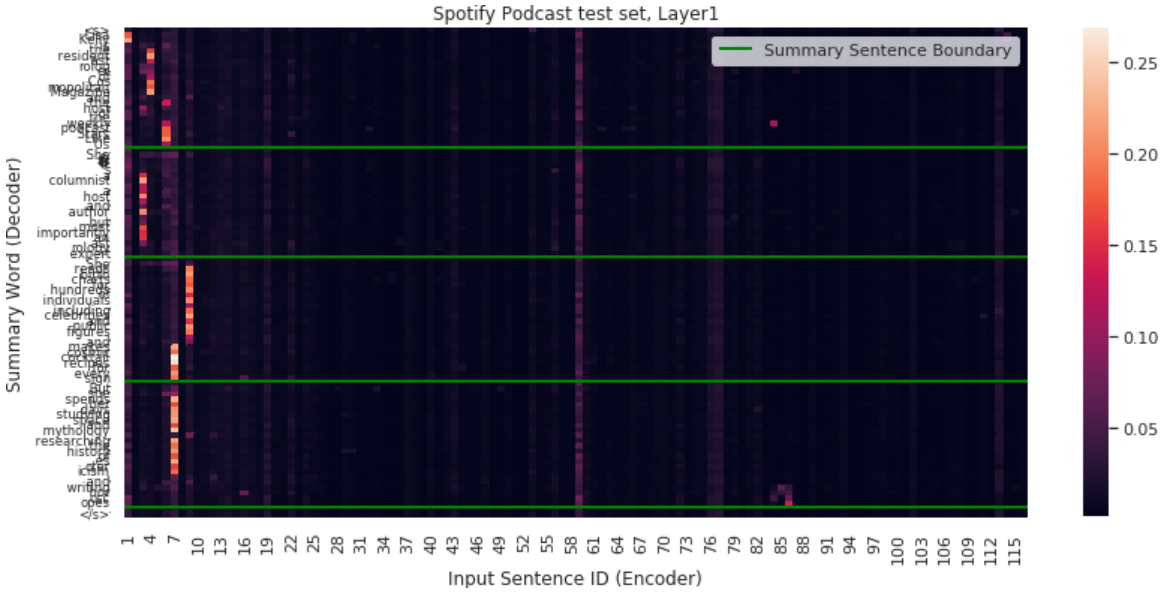


(b) Sentence-Level ($\alpha^{s}_{m,i}$)

Fig. 6.8 Example of BART's encoder-decoder attention evaluated on XSum test set. Blue vertical lines indicate input sentence boundaries and green horizontal lines indicate target sentence boundaries.

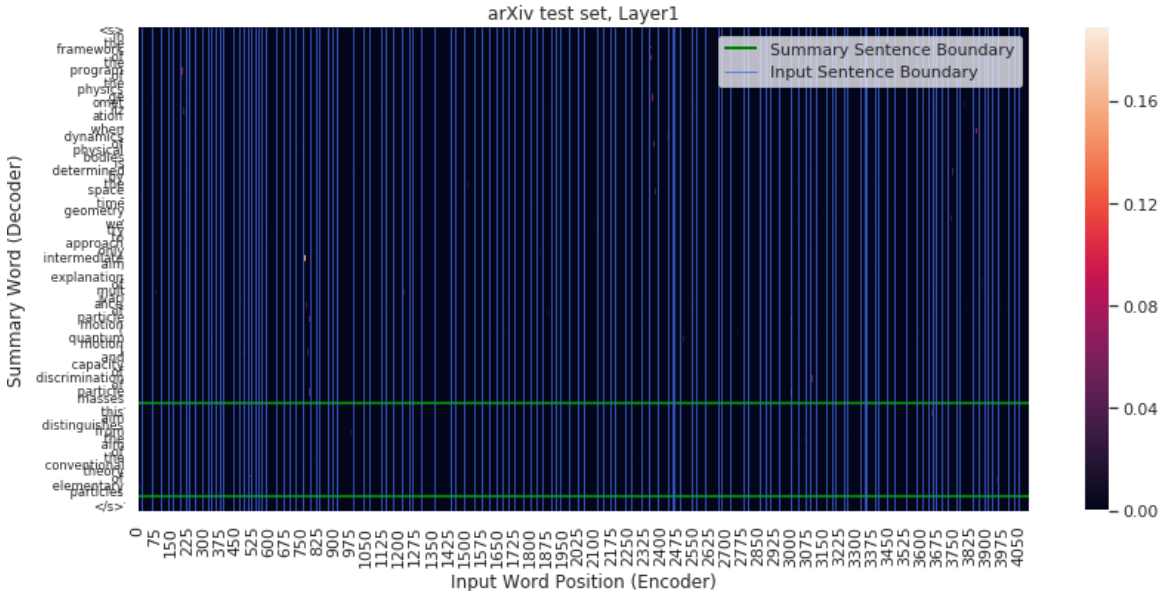


(a) Word-Level

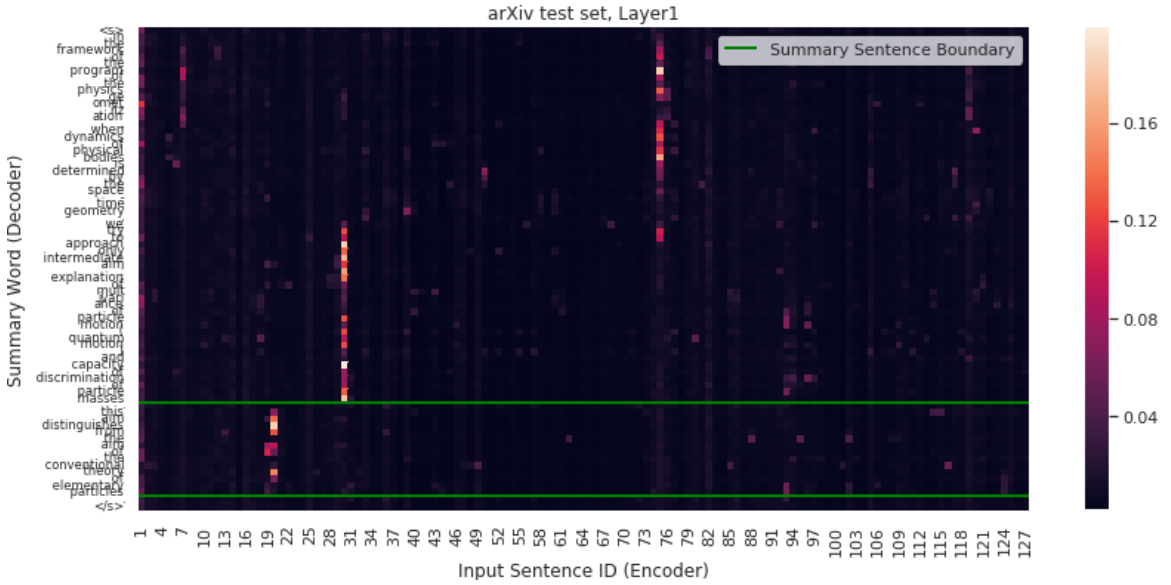


(b) Sentence-Level ($\alpha^{s}_{m,i}$)

Fig. 6.9 Example of LoBART's encoder-decoder attention evaluated on Podcast test set. Blue vertical lines indicate input sentence boundaries and green horizontal lines indicate target sentence boundaries.



(a) Word-Level



(b) Sentence-Level ($\alpha^{s}_{m,i}$)

Fig. 6.10 Example of LoBART's encoder-decoder attention evaluated on arXiv test set. Blue vertical lines indicate input sentence boundaries and green horizontal lines indicate target sentence boundaries.

6.4.2 Approximating Encoder-Decoder Attention

In Section 6.4.1, it was demonstrated that the encoder-decoder attention is *sparse*. This section aims to compute an approximation of the attention mechanism where only r input sentences are attended over instead of all sentences. Let the subscript (i, j) denote the position of

the j-th word in the i-th input sentence, e.g., $\mathbf{K} = [\underbrace{\mathbf{k}_{1,1}, \mathbf{k}_{1,2}, \mathbf{k}_{1,J_1}}_{\text{sent1}}, ..., \underbrace{\mathbf{k}_{i,1}, \mathbf{k}_{i,J_i}}_{\text{sent}i}, ..., \underbrace{\mathbf{k}_{N_1,1}, \mathbf{k}_{N_1,J_{N_1}}}_{\text{sent}N_1}]$.
At inference time, as the outputs are generated sequentially, i.e.,

$$\mathbf{a}_m = \text{softmax}(\mathbf{q}_m \mathbf{K}^T)\mathbf{V} \tag{6.20}$$

the top-r sentences can be determined independently for each $\mathbf{q}_m$. Consider the following sum of attention weights as sentence saliency at decoding step m of sentence i,[7] similar to the sentence-level RNN-based attention score in Section 5.4 and Section 6.2.2:

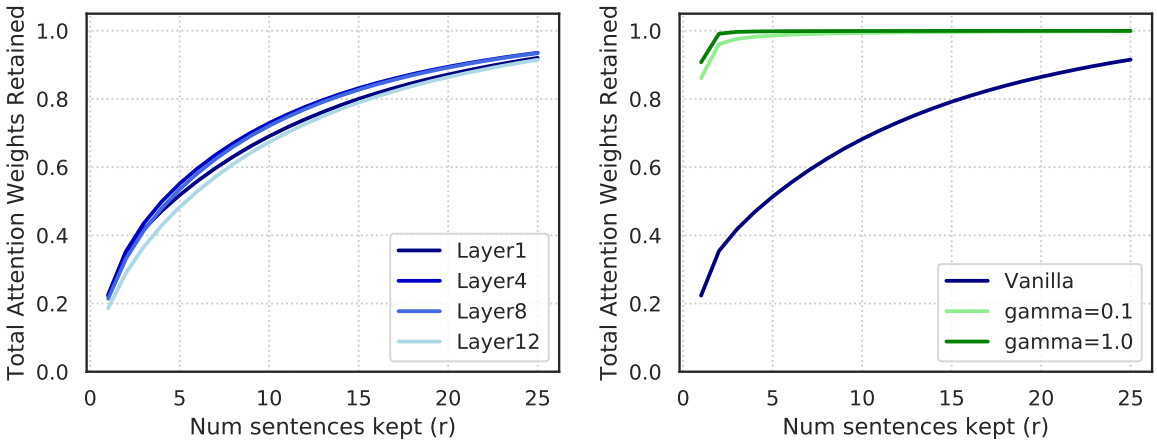
$$\alpha^{\text{s}}_{m,i} = \frac{1}{Z_m} \sum_{j=1}^{J_i} \exp(\mathbf{q}_m \cdot \mathbf{k}_{i,j}) \tag{6.21}$$

where $Z_m = \sum_{\forall i'} \sum_{\forall j'} \exp(\mathbf{q}_m \cdot \mathbf{k}_{i',j'})$. We then compute $\sum_i \alpha^{\text{s}}_{m,i}$ up to r sentences ranked by $\alpha^{\text{s}}_{m,i}$. This is based on the assumption that if the attention weights are sparse, summing $\alpha^{\text{s}}_{m,i}$ only on the top-r sentences could yield a value close to 100%. The experimental results in Figure 6.11a show that $r = 25$ is required to achieve the sum of attention weights at 90%. In addition to the vanilla BART model, we can fine-tune BART explicitly to make the attention sparse using the following criterion,

$$\mathcal{L}_{\text{A}} = \mathcal{L}_{\text{xent}} + \gamma \mathcal{L}_{\text{sparse}} \tag{6.22}$$

where $\mathcal{L}_{\text{xent}}$ is the teacher-forced cross entropy loss, $\mathcal{L}_{\text{sparse}} = \frac{1}{M} \sum_{m=1}^{M} \mathcal{H}(\boldsymbol{\alpha}^{\text{s}}_m)$, and entropy $\mathcal{H}(\boldsymbol{\alpha}^{\text{s}}_m) = -\sum_{i=1}^{N_1} \alpha^{\text{s}}_{m,i} \log \alpha^{\text{s}}_{m,i}$. We show in Figure 6.11b that the fine-tuned models (both $\gamma = 0.1$ and $\gamma = 1.0$) retain close to 100% of attention weights for small r. Next, we will investigate how selecting r sentences impacts the summarization performance.

[7] **Multi-head attention**: we omit the heads in the equations and expressions for simplicity. Both BART and LoBART models have 16 heads. In Figure 6.11, we average $\alpha^{\text{s}}_{m,i}$ over heads, before the summation. When computing an uncertainty measure such as entropy $\mathcal{H}(.)$ or KL-divergence KL(.), we compute the measure for each head separately and take the average. In obtaining $\mathcal{I}^r_m$, we average $\alpha^{\text{s}}_{m,i}$ over heads, before the top-r operation, i.e. all heads get assigned the same subset of sentences, but the differences are across layers and decoding timesteps.



(a) Layers 1,4,8,12 **(b)** $\mathcal{L}_{\text{A}}$-tuned (Layer1)

Fig. 6.11 Sum of attention weights against the number of retained sentences (r) evaluated on CNNDM.

To obtain an empirical upper bound performance of the approximation in Equation 6.19, for each $\mathbf{q}_m$, we can get *ideal* $\mathbf{k}, \mathbf{v}$ corresponding to the top r sentences ranked by $\alpha^{\text{s}}_{m,i}$,

$$\mathcal{I}^r_m = [(i,j) \text{ s.t. } i \in \text{top-}r(\alpha^{\text{s}}_{m,i})] \tag{6.23}$$

and hence, $\hat{\mathbf{K}}_m = [\mathbf{k}_{i,j} : (i,j) \in \mathcal{I}^r_m]$, and the same method is applied to obtain $\hat{\mathbf{V}}_m$.

6.4.3 Exploiting Sentence Structure

The previous sections discussed sparsity and an ideal approximation of encoder-decoder attention. We note that the ideal approximation of the attention involves computing ideal selection ($\mathcal{I}^r_m$ in Equation 6.23) which requires computing the attention weights on all input words before summing into the sentence level ($\alpha^{\text{s}}_{m,i}$ in Equation 6.21). This process cannot make the decoder more efficient.

Motivated by the hierarchical architecture in Chapter 5, we propose a method to exploit the sentence structure in the source document and to allow a compact approximation, using the following partition for the sentence-level attention score (Equation 6.21),

$$\alpha^{\text{s}}_{m,i} \approx \tilde{\alpha}^{\text{s}}_{m,i} = \text{softmax}\left(f_1(\mathbf{q}_m) \cdot f_2(\mathbf{k}_{i,1}, ..., \mathbf{k}_{i,J_i})\right) \tag{6.24}$$

where $\sum_{i=1}^{N_1} \tilde{\alpha}^{\text{s}}_{m,i} = 1.0$, f_1 and f_2 are generic functions that transform $\mathbf{q_m}$ and $\mathbf{k}_{i,1}, ..., \mathbf{k}_{i,J_i}$ into the same embedding space, respectively. The exact forms of f_1 and f_2 adopted in this work will be discussed in Section 6.4.4. Essentially, this method modifies the standard encoder-decoder attention such that it performs sentence selection based on $\tilde{\alpha}^{\text{s}}_{m,i}$ (Equation 6.24) and subsequently computes subset attention $\hat{\mathbf{A}}$ (Equation 6.19).

The modified encoder-decoder attention consists of two components: i) sentence-level attention over N_1 sentences; and ii) word-level attention over rN_2 words. To analyze its complexity, let p denote a unit of matrix multiplication cost and q denote a unit of softmax cost. The costs associated with attention are:

(i) Sentence-level (Equation 6.24): $pMN_1D + qMN_1$

(ii) Word-level (Equation 6.19): $2pMrN_2D + qMrN_2$

The additional cost associated with the sentence-level representation on the encoder side grows with the input length $N = N_1N_2$. Thus, as opposed to $\mathcal{O}(MN_1N_2)$ in the case of vanilla encoder-decoder attention, the overall complexity of the modified attention is,

$$\text{Complexity} = \mathcal{O}(MN_1 + k_wMrN_2 + k_eN_1N_2) \tag{6.25}$$

where $k_w \approx \frac{2pD+q}{pD+q}$ and k_e depends on the exact form of sentence-level computation.

6.4.4 Model-based Neural Approximator

In Section 6.4.3, a general framework to approximate the encoder-decoder attention by exploiting the sentence structure was proposed. In this section, we present a realization of the framework using an RNN model. To utilize the simple partition and sentence-level structure in Equation 6.24, we use a linear mapping for f_1 and a bidirectional RNN for f_2 as follows:

$$f_1(\mathbf{q}_m) = \mathbf{W}_q\mathbf{q}_m \tag{6.26}$$

$$f_2(\mathbf{k}_{i,1}, ..., \mathbf{k}_{i,J_i}) = \mathbf{W}_k\mathbf{h}_i \tag{6.27}$$

$$\mathbf{h}_i = \text{RNN}(\mathbf{k}_{i,1}, ..., \mathbf{k}_{i,J_i}) \tag{6.28}$$

As illustrated in Figure 6.12, the standard transformer model is extended by augmenting two layers as follows:

1. Sentence-level encoder-decoder attention computing the approximated attention $\tilde{\alpha}^{\text{s}}_{m,i}$ as defined in Equation 6.24.

2. Sentence encoder computing the sentence-level representation as in Equation 6.28.

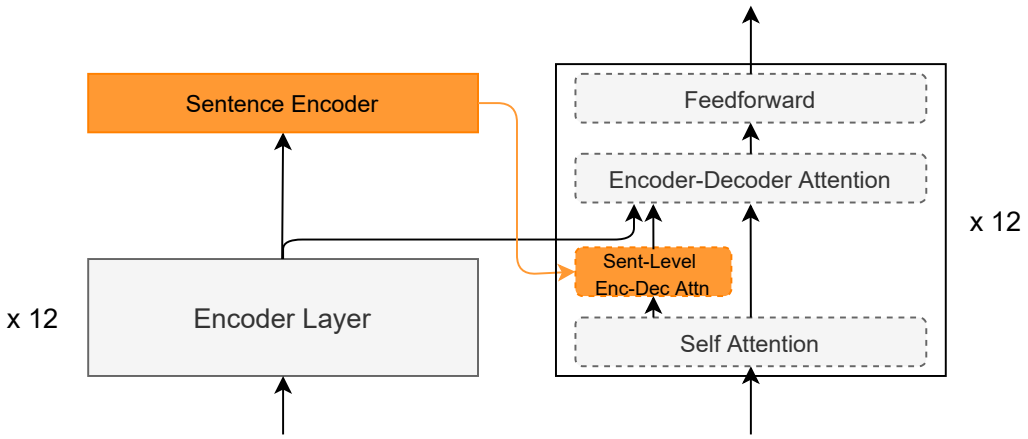


Fig. 6.12 Modified architecture with model-based approximator where the base model can be BART (or local-attention BART). Model-based neural approximation components are shown in orange.

KL Loss and Integrated Training

Let $\boldsymbol{\theta}_{\text{dec}}$ denote the original decoder and $\tilde{\boldsymbol{\theta}}$ denote the neural approximator. The parameters $\tilde{\boldsymbol{\theta}}$ are trained by minimizing KL-divergence,

$$\mathcal{L}_{\text{KL}} = \frac{1}{M}\sum_{m=1}^{M} \text{KL}\left(\boldsymbol{\alpha}_m^{\text{s}} \parallel \tilde{\boldsymbol{\alpha}}_m^{\text{s}}\right) \tag{6.29}$$

where

$$\text{KL}\left(\boldsymbol{\alpha}_m^{\text{s}} \parallel \tilde{\boldsymbol{\alpha}}_m^{\text{s}}\right) = \sum_{i=1}^{N_1} \alpha_{m,i}^{\text{s}} \log\left(\frac{\alpha_{m,i}^{\text{s}}}{\tilde{\alpha}_{m,i}^{\text{s}}}\right) \tag{6.30}$$

In addition, $\boldsymbol{\theta}_{\text{dec}}$ could also be trained simultaneously using the teacher-forced cross-entropy loss $\mathcal{L}_{\text{xent}}$. This setup is referred to as **integrated training**, and the combined training loss is,

$$\mathcal{L}_{\text{I}} = \mathcal{L}_{\text{xent}} + \lambda \mathcal{L}_{\text{KL}} \tag{6.31}$$

Note that $\mathcal{L}_{\text{I}}$ cannot be optimized in an end-to-end fashion because the top-r operation in Equation 6.23 is not differentiable. Hence, we interleave the training, i.e., update $\boldsymbol{\theta}_{\text{dec}}$ at fixed $\tilde{\boldsymbol{\theta}}$ and update $\tilde{\boldsymbol{\theta}}$ using $\mathcal{L}_{\text{KL}}$ only:

$$\Delta\boldsymbol{\theta}_{\text{dec}} = \nabla_{\boldsymbol{\theta}_{\text{dec}}} \mathcal{L}_{\text{I}}|_{\tilde{\boldsymbol{\theta}}} = \nabla_{\boldsymbol{\theta}_{\text{dec}}} \mathcal{L}_{\text{xent}}|_{\tilde{\boldsymbol{\theta}}} + \lambda \nabla_{\boldsymbol{\theta}_{\text{dec}}} \mathcal{L}_{\text{KL}}|_{\tilde{\boldsymbol{\theta}}} \tag{6.32}$$

$$\Delta\tilde{\boldsymbol{\theta}} = \nabla_{\tilde{\boldsymbol{\theta}}} \mathcal{L}_{\text{I}} = \cancelto{0}{\nabla_{\tilde{\boldsymbol{\theta}}} \mathcal{L}_{\text{xent}}} + \lambda \nabla_{\tilde{\boldsymbol{\theta}}} \mathcal{L}_{\text{KL}} \tag{6.33}$$

where $\nabla_{\tilde{\boldsymbol{\theta}}} \mathcal{L}_{\text{xent}} \overset{0}{\rightarrow}$ indicates that the gradient of $\mathcal{L}_{\text{xent}}$ with respect to $\tilde{\boldsymbol{\theta}}$ is not included in computing $\Delta\tilde{\boldsymbol{\theta}}$. In addition, since during training, we compute both $\alpha^{\text{s}}_{m,i}$ (ideal) and $\tilde{\alpha}^{\text{s}}_{m,i}$ (approx), we can use either of them in the top-r selection. Thus, inspired by scheduled sampling [11], we try mixing them using a scheduled mechanism as follows: $\alpha^{\text{s}}_{m,i}$ with probability $1 - \frac{\text{step}}{\text{epoch\_size}}$, otherwise $\tilde{\alpha}^{\text{s}}_{m,i}$.

6.5 Experiments

The experiments in this chapter focus on long-document summarization. This work examines sentence filtering (Section 6.2) and efficient attention mechanisms (Section 6.3 and Section 6.4) in the context of adapting pre-trained transformer models to abstractive summarization tasks. First, long-input summarization datasets are introduced. Then, the hierarchical RNN model from the previous chapter is compared against pre-trained transformer baselines.

6.5.1 Datasets

The details about AMI and CNN/DailyMail are provided in the previous chapter (Section 5.5.1). In addition, this chapter uses additional long-document summarization datasets, including Spotify Podcast Summarization, arXiv, PubMed, and XSum datasets, which are described below. Table 6.4 summarizes the statistics of the datasets. For the datasets examined, AMI, Podcast, arXiv, and PubMed are considered long-document tasks as their average document has over 1,000 tokens which is more than the limit of top-performing summarization models such as BART [159]. The Podcast dataset has the highest compression ratio of nearly 100, indicating that it has the highest redundancy.

Dataset	#Doc	Source Document			Summary	Ratio†
		Avg.	90th%	#Sent		
AMI	137	6,200	9,641	783.9	172	36.0
CNN/DailyMail	312k	870	1,463	67.4	67.4	12.9
XSum	227k	489	984	27.9	27.9	17.5
Podcast	106k	5,727	11,677	86.6	61.1	93.7
arXiv	216k	8,584	16,108	364.9	367	23.5
PubMed	133k	3,865	7,234	86.2	260	44.8

Table 6.4 Data statistics measured by the average number of tokens and 90th percentile. The AMI dataset is already processed into utterances, and we treat each utterance as one sentence. For other datasets, we use the NLTK tokenizer to split input texts into sentences. †The ratio between the average number of tokens in a document per the average number of tokens in a summary.

Spotify Podcast Summarization

Spotify Podcast dataset [38] is part of TREC[8] Challenge 2020. The dataset contains 105,390 episodes from different podcast shows on Spotify.[9] The podcasts were selected to be predominantly in English. The episodes are of various audio quality, many topics, and structural formats, e.g., interviews, conversations, monologues etc. The audio files are available, and the mean duration of an episode of 31.6 minutes. Metadata, including show description, episode description and other information, is provided. One set of transcripts of all audio files is given. The ASR system used was Google Cloud Platform's Cloud Speech-to-Text API (GCP-ASR). A set of 1,600 episodes was manually evaluated, and the word error rate was found at 18.1%. The first release contains 105,360 episodes from 18,376 shows – Figure 6.13 shows the histogram plot of the number of episodes for each show. The creator-provided episode descriptions are used as ground-truth summaries. Since there are different data pre-processing criteria and different splits, we summarize all the splits used in our experiments in Table 6.5. The average number of words per episode (mean±std) is 5727.8±4152.7, and the average number of words per summary is 61.1±63.2.

[8]https://trec.nist.gov/
[9]https://podcastsdataset.byspotify.com/

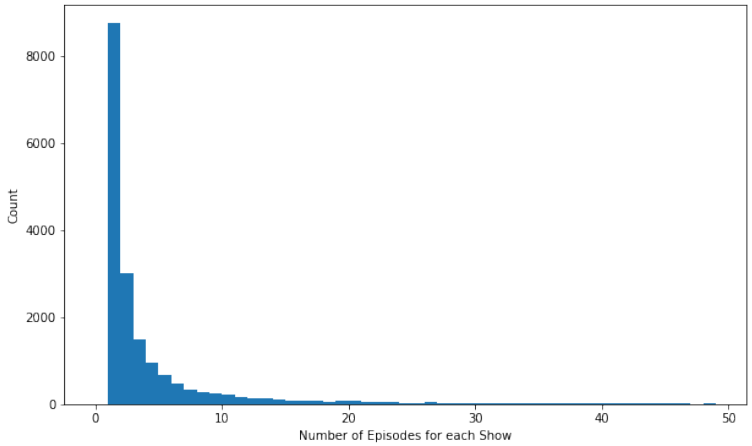


Fig. 6.13 Histogram plot of the number of episodes for each show.

Data Split		Size	Description
All	-	105,390	All episodes available in the first release
CUED	train	92,294	First 100k in **All** minus those with description $<$ 5 tokens
	dev	4,927	Last 5,390 in **All** minus those with description $<$ 5 tokens
SPTF	brass	66,242	**All** minus filtering by Spotify
	brass_sub	66,142	**SPTF_brass** minus those in the **SPTF_toy** set
	train	60,415	**CUED_train** that are in **SPTF_brass_sub**
	dev	2,189	**CUED_dev** that are in **SPTF_brass_sub**
	test	1,027	Held-out test set released in August 2020 the competition

Table 6.5 Spotify Podcast data splits. *brass set* [125] was obtained by filtering the entire summarization training set using three heuristics: (1) too long (>750 characters) or too short (<20 characters); (2) description too similar to other descriptions; (3) description too similar to its show description where sentence similarity is calculated using the `sklearn` library.

arXiv and PubMed Summarization

arXiv[10] and PubMed[11] are two datasets of long and structured documents, consisting of academic articles (scientific papers) with abstracts as summaries proposed by Cohan et al. [39]. The datasets are obtained from arXiv and PubMed OpenAccess repositories. The arXiv dataset contains 203,037, 6,436, and 6,440 in training, validation, and test splits, respectively.

[10]https://arxiv.org/
[11]https://pubmed.ncbi.nlm.nih.gov/

The PubMed dataset contains 119,924, 6,633, and 6,658 in training, validation, and test splits, respectively

XSum

Extreme Summarization (XSum) dataset is introduced by Narayan et al. [209]. XSum contains news articles extracted from the BBC. The goal of XSum is to have a highly abstractive summarization dataset, so the first sentence of each news article is treated as the summary. As a result, there are 36% novel unigrams[12] in the XSum reference summaries on average compared to 17% in the CNN/DailyMail reference summaries. However, it should be noted in some cases, the information in the first sentence is not present in the subsequent parts of the article, and this leads to hallucination [193], and Chapter 7 will investigate hallucination or information consistency. The train, validation, and test splits are 204,045, 11,332, and 11,334 documents.

A note on reference summaries

Despite being used widely, it should be noted that standard datasets including CNN/DailyMail and XSum have pseudo summaries instead of real summaries [250]. For example, the reference summaries in CNN/DailyMail are news 'highlights', which are bullet point items (usually a few sentences) that appear at the top of each news article. Although they can be (and have been) used as summaries, they are primarily designed to grab readers' attention instead of summarizing the news. For XSum, the summaries were created by extracting the 'first sentence' of the article from the BBC where some of these first-sentence summaries even contain information that is not present in the article [193]. A first-sentence summary is also called a lead sentence, which is meant to quickly grab readers' interest rather than acting as a summary. For Spotify Podcast, creator-provided descriptions are treated as summaries. Although some descriptions provide good overviews about the corresponding podcast episodes, up to a quarter of them are considered *bad* summaries in human evaluation.[13]

6.5.2 Comparing Hierarchical RNN and Transformer

An initial investigation is to compare the performance of the hierarchical RNN model (HIER) developed in Chapter 5 to the performance of foundation model based systems. The results in Table 6.6 show that: on the AMI dataset, HIER performs better than BART as expected. The AMI transcripts are much longer than the maximum length of BART, meaning that there

[12]Novel unigrams refer to unigrams or words that do not appear in source documents.
[13]The information about the podcast summary assessment dataset is provided in Appendix A.

is a significant information loss due to truncation, and even self-supervised pre-trained BART does not perform better than HIER. On the other hand, on the CNN/DailyMail dataset, both HIER and BART systems can handle the entire input sequence (though with some exceptions for BART). BART significantly outperforms HIER on CNN/DailyMail, demonstrating the effectiveness of the pre-training + fine-tuning paradigm. On the Spotify Podcast dataset, despite having truncated documents as the inputs, BART is able to perform better than HIER similar to CNN/DailyMail. This suggests that in the scenario when the size of training data is large (e.g., in the order of 100k samples for CNN/DailyMail and Podcast), using a foundation model such as BART can be effective. Nevertheless, the fact that the source documents still exceed the maximum limit of BART, the subsequent experiments will investigate methods to handle long sequences. Lastly, the initial results of LoBART (introduced in Section 6.3) show that it performs better than BART on both AMI and Podcast. These results show the effectiveness of expanding context length through attention localization. LoBART will be examined in detail in further experiments in Section 6.3.

			AMI			CNN/DailyMail			Spotify Podcast‡		
Model	**MaxLen**	**#Param**	**R1**	**R2**	**RL**	**R1**	**R2**	**RL**	**R1**	**R2**	**RL**
LeadSent	-	-	24.38	3.65	15.99	38.65	17.00	30.55	19.73	4.37	12.56
HIER	1000×50†	53M	44.36	14.62	41.10	35.73	14.00	33.01	24.28	10.11	22.25
BART	1024	406M	40.75	13.17	38.00	44.03	20.92	40.99	31.55	12.86	28.18
LoBART	4096	409M	42.18	13.62	38.56	*	*	*	32.09	13.11	28.43
LoBART	8192	413M	43.01	14.40	40.31	*	*	*	31.85	12.97	28.22

Table 6.6 Comparison between HIER and BART where both models are trained on a target dataset. AMI results are the average of 3 runs. ‡The results are on the development set of CUED-split. †HIER splits the input text into sentences where a sentence can have up to 50 tokens and the maximum number of sentences is 1000. LoBART is discussed in Section 6.3 where LoBART(4k) has $W = 1024$ and LoBART(8k) has $W = 512$. '*' indicates that LoBART's results are the same as BART's results on CNN/DailyMail. This is because LoBART is an extension of BART to handle sequences longer than the maximum length of BART. However, the documents in CNN/DailyMail are within the maximum length of BART, so there is no difference in the results.

6.5.3 Sentence Filtering

In the previous chapter, the hierarchical model did not require a sentence filtering stage to operate (i.e., there was *no* truncation). In contrast, this chapter considers using foundation models for summarization tasks, and the maximum length of the source document is limited. For example, to fine-tune the vanilla BART model, the source document has to be truncated

to 1,024 tokens as BART only has a positional embedding up to the 1024-th token. In this experiment, we investigate sentence filtering in the following scenarios of fine-tuning BART.

Sentence Filtering Baselines

During training, the input sequences are truncated to be within the maximum length of 1,024 tokens. Four simple sentence filtering baselines are considered as follows,

1. Truncate: truncate the sentences that exceed the maximum length
2. Random: randomly select sentences such that the total number of tokens does not exceed the maximum length
3. TextRank: compute score using TextRank [198] (described in Section 4.1.1) with GloVe embedding [227]
4. HIER: following the previous chapter (in Section 5.4), the hierarchical model (HIER), trained on CNN/DailyMail and fine-tuned on Podcast

Results in Table 6.7 show that while truncation is a strong baseline, HIER yields the best performance among the baselines considered. Random selection performs poorly as expected. Unexpectedly, the TextRank algorithm yields the worst results. This is likely due to the sentence embedding obtained by averaging GloVe embeddings and non-salient words such as 'the', 'and', 'or' that were not excluded in the computation. As the TextRank baseline is considerably worse than the Truncation baseline, this work did not try to further improve the TextRank baseline.

Method	**ROUGE F_1**		
	R1	**R2**	**RL**
Random	25.65	6.89	22.10
Truncate	29.09	9.92	25.37
TextRank	24.97	6.40	21.47
HIER	**29.37**	**10.02**	**25.51**

Table 6.7 Comparison of sentence filtering baselines applied at the inference stage. The downstream abstractive summarizer is BART-large with a maximum positional embedding of 1024 fine-tuned on Podcast (at the training stage, the input sequences were truncated to 1024). The evaluation is on the development set of the SPTF-split podcast data

Subsequently, we investigate the scenario where sentence filtering is applied at both training and inference stages. Based on the results in Table 6.7, we select HIER as the default filtering

method. Table 6.8 shows that the best summarisation performance is achieved when sentence filtering is applied at both training and inference stages.

Stage		ROUGE F_1		
Training	**Inference**	**R1**	**R2**	**RL**
Truncate	Truncate	29.09	9.92	25.37
Truncate	HIER	29.37	10.02	25.51
HIER	HIER	**29.74**	**10.25**	**25.71**

Table 6.8 Sentence filtering applied at both training time and test stages. The evaluation is on the development set of the SPTF-split podcast data.

Training Model-based Multi-task Content Selection (MCS)

In preliminary experiments, the number of selected sentences is varied from the model limit to a few sentences, and it is found that more aggressive selection at test time degrades the performance. Therefore, MCS selects input sentences up to the model limit. The training criterion of MCS is defined as $\mathcal{L}_{\text{MCS}} = \lambda \mathcal{L}_{\text{label}} + (1-\lambda)\mathcal{L}_{\text{seq2seq}}$ (in Equation 6.9). This experiment examines the following setups:

- $\lambda = 0.0$: MCS is the same as HIER (experimented in Section 5.4). The model is trained only on $\mathcal{L}_{\text{seq2seq}}$. At inference, it uses only the attention score.
- $\lambda = 1.0$: This model is similar to the extractive models in Hsu et al. [114], Pilault et al. [231] where it is trained on extractive labels only.
- $\lambda \in [0.0, 1.0]$: This is when the model is trained in the multi-task setup.

In Table 6.9, we show that when coupled with BART, MCS yields better summarization performance than both Attention-only ($\lambda = 0.0$) and Extractive-only ($\lambda = 1.0$) baselines. Furthermore, we also found that MCS achieves a higher recall rate of sentences with $\texttt{sim}(S_i, Y) > 0$ than the two baselines.

Comparing Oracle and Model-based Sentence Filtering

The previous experiment examined the methods for training MCS. However, there exist sentence filtering approaches such as truncation or oracle methods (Section 6.2.1). The aim of this experiment is, therefore, to compare these sentence filtering approaches. At training, we investigate five different training methods (show on the X-axis of Figure 6.14) on BART:

System	%Recall	R1	R2	RL
Attention-only ($\mathcal{L}_{\text{seq2seq}}$)	38.85	26.90	9.70	18.78
Extractive-only ($\mathcal{L}_{\text{label}}$)	35.26	26.39	8.90	18.03
Multi-task ($\mathcal{L}_{\text{MCS}}$)	40.50	27.28	9.82	19.00

Table 6.9 The impact of test-time content selection methods on BART(1k) trained using Oracle-pad-rand. Optimal $\lambda = 0.2$ is tuned in the range $[0.0, 1.0]$ on the validation set.

Truncation, MCS, Oracle-pad-lead, Oracle-pad-rand, Oracle-no-pad. At inference, we also apply all sentence filtering methods on each of the training setup. Note that in practice an oracle method can be used at training as the groud-truths are available. However, at inference, an oracle method only demonstrates an empirical upper bound.

In Figure 6.14, since any oracle method only represents an upper-bound performance at test time, the best performance is obtained by MCS (in blue), and the upper-bound performance is obtained by the optimal oracle method (in green). The results show that although applying Oracle-no-pad at training (i.e., using optimal set of source sentences) yields the highest upper bound, the abstractive model in fact does not learn how to perform abstraction. For instance, with truncation (TRC) or MCS at the inference stage (i.e., test time), Oracle-no-pad yields the lowest performance level. The best way to fine-tune the abstractive model shown in Figure 6.14 is using Oracle-pad-rand. Compared to Oracle-pad-lead, Oracle-pad-rand is better. This is likely because it introduces more diversity to the abstractive model. Compared to the model-based MCS method, Oracle-pad-rand is also computationally less expensive.

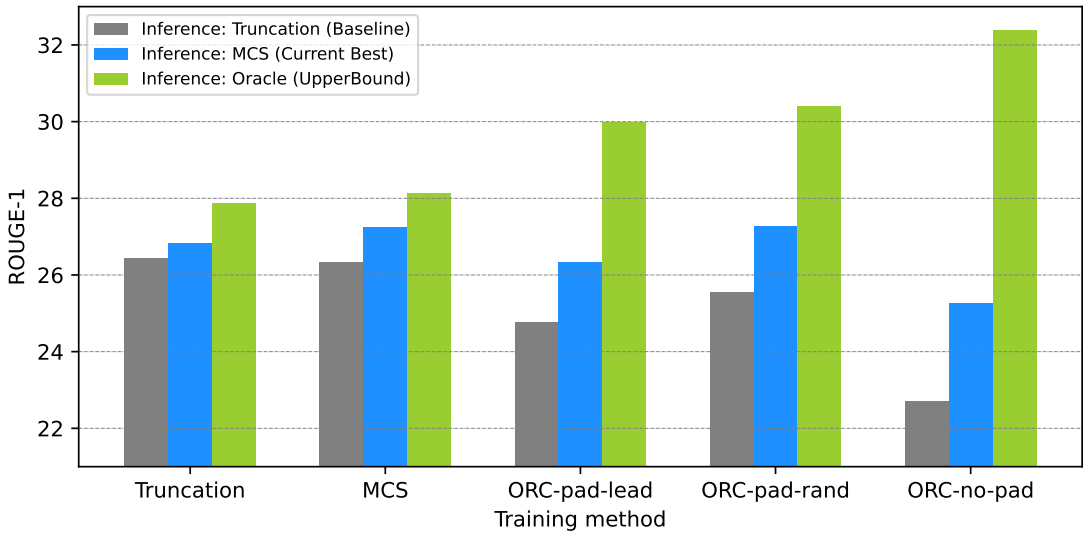


Fig. 6.14 The impact of training-time content selection methods on BART(1k).

In addition, the results in the following experiments (Table 6.13 in Section 6.5.5) show that when there is no sentence filtering at the inference stage (i.e., truncation is performed), LoBART(4k) and LoBART(8k) benefit from Oracle-pad-rand, whereas BART(1k) does not. This is because, in the 1k setting, sentence filtering is more aggressive; as a result, the large mismatch between training and inference leads to a poor result. Thus, we suggest that the best sentence filtering during training is Oracle-pad-rand given that sentence filtering will be used at the inference stage, or the input span is long.

6.5.4 Local Attention Model

Based on the analysis in Section 6.3, this work trains different configurations of BART and local-attention BART (LoBART) models up to our VRAM limit of 32 GB (V100). The results in Table 6.10 and Figure 6.15 show that:

1. Expanding the model to accommodate longer input spans, BART(1k) $\rightarrow$ BART(2k), improves over the baseline BART(1k). This is as opposed to training longer-span models by freezing bottom layers, which does not show any improvement over fine-tuning vanilla BART.

2. Using larger window size W leads to improved performance in all experiments.

3. Although LoBART(8k) with $W = 512$ can process longer input spans than LoBART(4k) with $W = 1024$, it performs worse and we suggest that this is because LoBART(8k)'s

window is too small. For example, the previous analysis in Section 6.3.1 suggests that W should be larger than 700, to efficiently utilize transfer learning from BART.

System	Max N	W	VRAM	R1	R2	RL
BART(N=1k)	1024	Full	8.9	26.43	9.22	18.35
LoBART(2k)	2048	128	9.6	25.88	8.89	17.87
LoBART(2k)	2048	256	10.2	25.93	8.80	17.82
LoBART(2k)	2048	512	11.6	26.35	8.98	18.19
LoBART(2k)	2048	1024	14.2	26.44	9.26	18.25
BART(2k)	2048	Full	14.5	26.63	9.41	18.65
LoBART(4k)	4096	128	12.8	26.42	9.02	18.12
LoBART(4k)	4096	256	14.1	26.66	9.22	18.33
LoBART(4k)	4096	512	16.7	26.75	9.54	18.54
LoBART(4k)	4096	1024	22.0	**27.02**	**9.57**	**18.78**
LoBART(8k)	8192	128	19.3	26.45	9.04	18.23
LoBART(8k)	8192	256	21.1	26.72	9.30	18.36
LoBART(8k)	8192	512	27.1	26.90	9.47	18.50

Table 6.10 Summarization results on the Podcast dataset. BART & LoBART memory requirement in training and performance. (nk) denotes the maximum input length of $n \times 1024$. VRAM (in GB) is limited to 32 GB due to the available GPU for training.

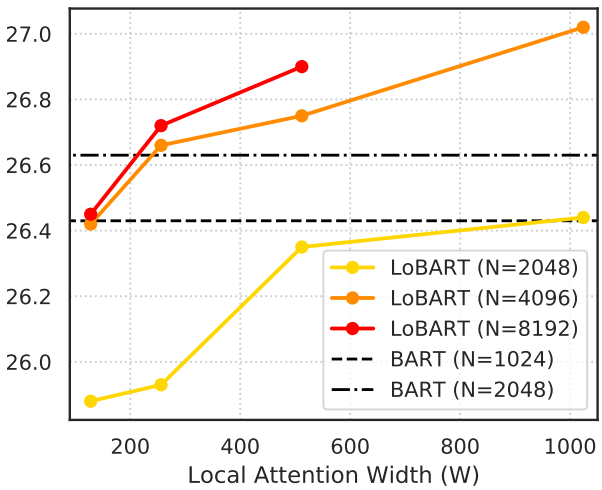


Fig. 6.15 ROUGE-1 on the Podcast dataset for BART and LoBART with different N and W values – the training is constrained by a maximum VRAM of 32 GB.

Ablation on Impact of Initialization

The previous investigation showed that applying local attention to BART is effective for long-document summarization. It was assumed that BART achieved good performance due to pre-training. This ablation will study this in more detail where we compare three initialization: random initialization (no pre-training), BART-large (with pre-training), and BART-large-CNNDM (with pre-training + adapting to summarization). The results in Table 6.11 verify that pre-training is crucial as shown by a large gap with performance between random intialzation and BART-large initialization. Also, transfer learning further improves the performance as shown by the gain from BART-large-initialization.

System	Initialization	R1	R2	RL
BART(1k)	Random	14.61	0.82	11.54
	BART-large	25.82	9.07	17.99
	BART-large-CNNDM	26.43	9.22	18.35

Table 6.11 Podcast results. The impact of transfer learning. Truncation is applied at both the training and test stages.

Next, the impact of initialization is examined on LoBART. The arXiv dataset is chosen as it allows the comparison between LoBART and LED [10]. The results in Table 6.12 show an improvement from transfer learning (BART-large → BART-large-CNNDM) in both Truncate and Oracle-pad-rand setups. In addition, we also compare LoBART against LED. LED, a concurrent work to LoBART, also applies local attention to the encoder of BART. LED is only trained on truncated inputs and it is initialized from BART-large, and LED(4k) is comparable to to LoBART(4k) with truncation and initialized from BART-large. Despite the similarity, LoBART(4k) achieved slightly better performance than LED(4k).

System	Train†	Initialization	R1	R2	RL
LED(4k)	Truncate	*BART-large	44.40	17.94	39.76
LoBART(4k)	Truncate	BART-large	46.17	17.96	40.74
	Truncate	BART-large-CNNDM	46.90	18.88	41.50
	Oracle-pad-rand	BART-large	45.25	17.40	39.96
	Oracle-pad-rand	BART-large-CNNDM	46.59	18.72	41.24

Table 6.12 arXiv results. The impact of transfer learning on initializing LoBART. At test time, there is *no* content selection. *To our understanding, LED-large was initialized from BART-large. The results of LED(4k) is quoted from its paper. †Sentence filtering method at training.

6.5.5 Combined Approach: Sentence Filtering + Local Attention

The previous experiments studied sentence filtering and local attention, and both methods were shown to be effective for long-document summarization. The aim of this experiment is to investigate the combination of both techniques: LoBART and sentence filtering as illustrated in the pipeline in Figure 6.16. This section will evaluate this pipeline on the Podcast, arXiv, and PubMed datasets.

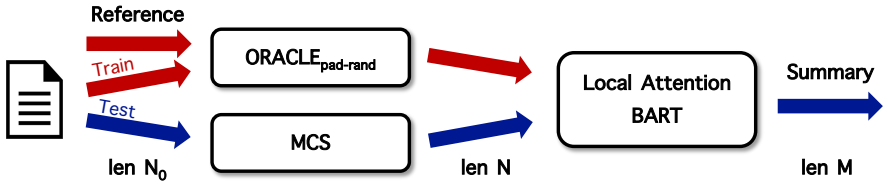


Fig. 6.16 Combined Approach Pipeline: Sentence Filtering and Local Attention Model

Spotify Podcast Results

Oracle-pad-rand is selected at training and model-based Multi-task Content Selection (MCS) is selected at inference because this setup yielded the best performance in the previous experiments in Section 6.5.3. These sentence filtering methods are coupled with BART and LoBART systems. Table 6.13 demonstrates that performance gain is achieved in all settings when adding MCS. By comparing different configurations with MCS, it can be seen that the gain from MCS in LoBART(8k) system is the lowest. This is because the average length is 5727, meaning that many podcast inputs LoBART(8k) do not benefit from sentence filtering.

For the Podcast Summarization Challenge 2020 (Section 6.5.7), the best single model system (CUED-filt) used an attention-based sentence filtering (i.e., HIER) at both stages, and HIER was combined with fine-tuned BART. Here, this combined approach outperforms CUED-filt by improved sentence filtering at both stages as demonstrated by BART(1k)-ORC+MCS. Additionally, local self-attention allows training on longer sequences, and the LoBART(4k)-ORC+MCS system has yielded the best results. Lastly, although LoBART(8k) requires higher computational cost to train, it does not perform as well as LoBART(4k) due to its smaller attention window, and it also has a lower improvement from adding MCS.

System	Max N	W	Sent. Filtering		R1	R2	RL
			Train.	Infer.			
BART(1k)	1024	Full	✗	✗	26.43	9.22	18.35
BART(1k)	1024	Full	✗	MCS	26.82	9.39	18.57
BART(1k)	1024	Full	ORC	✗	25.54	9.00	17.83
BART(1k)	1024	Full	ORC	MCS	27.28	9.82	19.00
LoBART(4k)	4096	1024	✗	✗	27.02	9.57	18.78
LoBART(4k)	4096	1024	✗	MCS	27.53	9.95	19.08
LoBART(4k)	4096	1024	ORC	✗	27.36	10.04	19.33
LoBART(4k)	4096	1024	ORC	MCS	**27.81**	**10.30**	**19.61**
LoBART(8k)	8192	512	✗	✗	26.90	9.47	18.50
LoBART(8k)	8192	512	✗	MCS	27.02	9.52	18.62
LoBART(8k)	8192	512	ORC	✗	27.16	9.84	19.08
LoBART(8k)	8192	512	ORC	MCS	27.49	9.98	19.25

Table 6.13 The results on the Podcast test set. The impact of sentence filtering at both stages: training stage = ORC (Oracle-pad-rand) and inference stage = Model-based MCS.

System	Max N	W	Sent. Filtering		arXiv			PubMed		
			Train.	Infer.	R1	R2	RL	R1	R2	RL
BART(1k)	1024	Full	✗	✗	44.96	17.25	39.76	45.06	18.27	40.84
BART(1k)	1024	Full	✗	MCS	46.11	18.79	40.83	46.46	19.54	41.91
BART(1k)	1024	Full	ORC	✗	42.03	15.62	37.15	43.20	17.02	39.19
BART(1k)	1024	Full	ORC	MCS	47.68	19.77	42.25	46.49	19.45	42.04
LoBART(4k)	4096	1024	✗	✗	46.90	18.88	41.50	47.40	20.43	42.95
LoBART(4k)	4096	1024	✗	MCS	48.05	20.11	42.58	47.76	20.76	43.27
LoBART(4k)	4096	1024	ORC	✗	46.59	18.72	41.24	47.47	20.47	43.02
LoBART(4k)	4096	1024	ORC	MCS	**48.79**	**20.55**	**43.31**	**48.06**	**20.96**	**43.56**

Table 6.14 The results on arXiv and PubMed (complementary with the results in Table 6.15). The impact of sentence filtering at both stages: training stage = ORC (Oracle-pad-rand) and inference stage = Model-based MCS.

ArXiv and PubMed Results

To verify the effectiveness, this set of experiments examines BART(1k) and LoBART(4k) with sentence filtering on arXiv and PubMed. First, when applying Oracle-no-pad, we examine the percentage of input documents that are shorter than the abstractive summarizer's input span (e.g., 1024 for BART). In the 1k setting, this percentage is only 2.8% on arXiv (12% on PubMed). In the 4k setting, this percentage is 39% on arXiv (71% on PubMed). These findings that suggest that an oracle method with padding is required; otherwise, the abstractive summarizer may not learn how to perform abstraction as found on the Podcast experiment in Figure 6.14. Based on the best configurations on Podcast, we train BART(1k) and LoBART(4k) using truncation (TRC) or Oracle-pad-rand. At inference, the hierarchical model fine-tuned on arXiv/PubMed for the model-based MCS approach is used.

The results on ArXiv: Table 6.15 shows the results on arXiv (and PubMed) as reported in previous work and our systems. These results show that both BART(1k)+MCS and LoBART(4k)+MCS outperform all existing systems. To better understand the advantages of the approach, the following systems are compared as follows:

	Type	System	arXiv			PubMed		
			R1	R2	RL	R1	R2	RL
Previous Work	Abs	Discourse-Aware [39]	35.80	11.05	31.80	38.93	15.37	35.21
	Mix	Ext+TLM [231]	41.62	14.69	38.03	42.13	16.27	39.21
	Ext	ExtSum-LG+Rd[324]	44.01	17.79	39.09	45.30	20.42	40.95
	Abs	Pegasus [340]	44.21	16.95	38.83	45.97	20.15	41.34
	Abs	DANCER [86]	45.01	17.60	40.56	46.34	19.97	42.42
	Abs	BigBird(3k) [338]	46.63	19.02	41.77	46.32	20.65	42.33
	Abs	LED(4k) [10]	44.40	17.94	39.76	-	-	-
	Abs	LED(16k) [10]	46.63	19.62	41.83	-	-	-
	Mix	CTRLsum(BART+BERT) [101]	46.91	18.02	42.14	-	-	-
This Work	Abs	†BART(1k)	44.96	17.25	39.76	45.06	18.27	40.84
	Mix	‡BART(1k)+MCS	47.68	19.77	42.25	46.49	19.45	42.04
	Abs	‡LoBART(4k)	46.59	18.72	41.24	47.47	20.47	43.02
	Mix	‡LoBART(4k)+MCS	**48.79**	**20.55**	**43.31**	**48.06**	**20.96**	**43.56**

Table 6.15 The main results on arXiv and PubMed of the combined approach compared to existing approaches. †denotes truncation applied, and ‡denotes Oracle-pad-rand applied at training time.

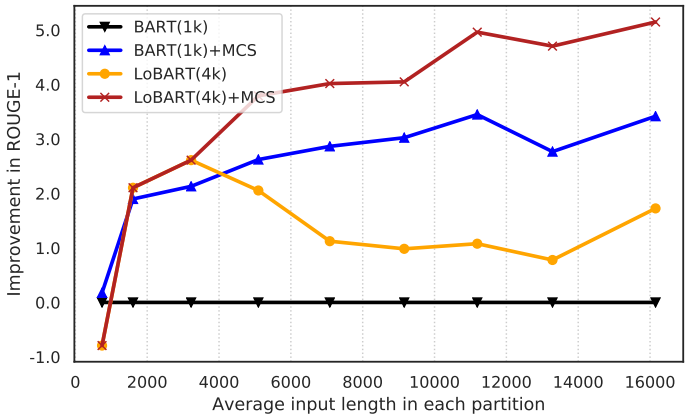
• *CTRLsum [101] versus our BART(1k) baseline*: CTRLsum extends BART by conditioning it with extracted keywords V using a BERT-based model, e.g., $p(Y|X,V)$. Their BERT-based model uses a sliding window allowing it to extract V in long sequences, but their BART is still limited to the first 1,024 tokens. As a result, CTRLsum performs better than vanilla BART(1k). The results of BART(1k)+MCS show that explicitly performing sentence filtering is more effective than extracting keywords and modifying the conditional probability.

• *LED [10] & BigBird [338] versus our LoBART(4k) system*: LoBART(4k) has a similar architecture to LED(4k) without the global attention pattern for special tokens. Instead, LoBART(4k) benefits from knowledge transferred from training on CNNDM (previously shown in Table 6.12) and Oracle-pad-rand at the training time, which yields a larger gain when MCS is applied (i.e., the system trained with truncated data has a smaller gain when MCS is applied). Compared to BigBird, LoBART(4k) has a longer input span, e.g. 3,072 vs. 4,096. However, BigBird benefits from utilizing more recent summarization-specific pre-training (Pegasus) [340] which is better than our transfer learning. BigBird also incorporates a global attention pattern similar to LED as well as a random attention pattern. LoBART without MCS performs worse, so this suggests that an improvement for LoBART could be made by incorporating global and random attention patterns. Nevertheless, we show that adding MCS to either BART(1k) or LoBART(4k) yields a significant improvement, resulting in state-of-the-art results at the time of conducting the experiments in both settings. Moreover, although the gain from adding MCS is comparable to the gain observed in extending LED(4k) to LED(16k), sentence filtering adds less training cost.

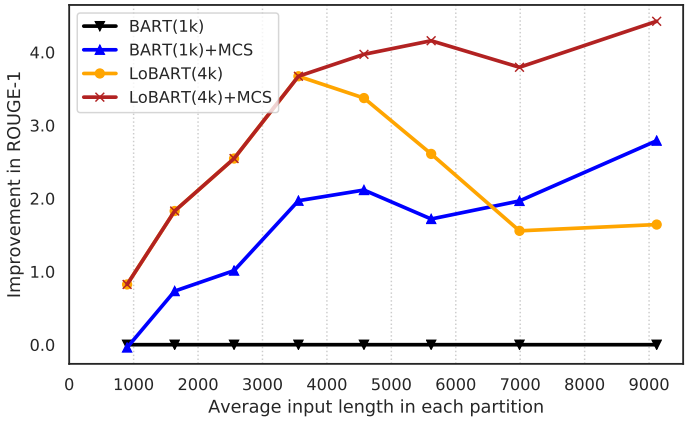
The results on PubMed: Similar to the results on arXiv, LoBART(4k)+MCS achieves state-of-the-art results shown in Table 6.15. In contrast to the arXiv results, BART(1k)+MCS does not outperform LoBART(4k) nor BigBird, and the gain from MCS is not as high in both 1k and 4k settings. These findings are likely because PubMed documents (avg. 3,865 tokens) are on average shorter than arXiv documents (avg. 8,584 tokens). Additionally, Table 6.15 shows that local attention yields better performance on PubMed, while MCS yields better performance on arXiv. Therefore, the next investigation will be on the performance of MCS at different input lengths.

Fine-grained analysis on input lengths: A fine-grained analysis is conducted by partitioning the test sets by input lengths. Figures 6.17a and 6.17b show the performance of various systems relative to the performance of BART(1k) at different input lengths. The results on arXiv and PubMed illustrate that as the input length N increases, (1) the improvement of systems *with* MCS increases and subsequently plateaus out; (2) the improvement of systems *without* MCS decreases once the input exceeds the length limit but then plateaus, suggesting

that fixed-span systems without sentence filtering perform worse once the maximum fixed-span is reached. For instance, below 4,000 input words, LoBART(4k) without MCS performs better than BART(1k)+MCS.



(a) arXiv (Len:Avg=8,584, 90$^{\text{th}}$%=16,108)



(b) PubMed (Len:Avg=3,865, 90$^{\text{th}}$%=7,234)

Fig. 6.17 ROUGE-1 relative to that of BART(1k) system evaluated on different partitions by length.

6.5.6 Exploiting Sparsity in Encoder-Decoder Attention

Section 6.4 previously discussed the decoder's cost in the training and inference stages, which motivated the study of efficient encoder-decoder attention. This section will examine the observed sparsity (Section 6.4.1), the approximation of encoder-decoder attention using a subset of input sentences (Section 6.4.2), and the summarization performance of the modified architecture (Section 6.4.3 and Section 6.4.4).

First, to investigate the approximation of the encoder-decoder attention, this experiment examines selection baselines and the **empirical upper bound performance via ideal selection**. Note that $\mathcal{I}_m^r$ denote the subset of r selected source sentences at the decoder time step m.

1. $\mathcal{I}_m^r$ = *N/A*: Vanilla encoder-decoder attention where there is *no* sentence selection ($r =$ All and $\mathcal{I}_m^r =$ N/A), i.e., all source sentences are used in computing the attention. This setup is a baseline without approximation

2. $\mathcal{I}_m^r$ = *Random*: Random selection of the subset of source sentences. This setup should represent a weak baseline

3. $\mathcal{I}_m^r$ = *Ideal*: Ideal selection of the subset of source sentences is based on Equation 6.23. This process guides the decoder to attend over only top sentences instead of attending over all sentences. This setup is the empirical upper bound of the approximation.

As the approximation based on a subset of source sentences is expected to perform better on a more extractive task, the first experiment is conducted on widely-used CNN/DailyMail (CNNDM). The base abstractive summarization system is BART fine-tuned on CNNDM where its performance is shown in the first row in Table 6.16.

Previous work on CNNDM [169] used the top-3 sentences as the extractive summary, and an example of a sentence-level attention plot in Figure 6.7b shows that salient sentences can be under five sentences. Hence, this experiment sets the number of retained sentences (r) set to 5. The results in Table 6.16 show that:

- For the vanilla model, despite the sum of attention weights being around 50% at $r = 5$ (in Figure 6.11a), the model is sufficiently sparse, and constraining to r ideal sentences (All $\rightarrow \mathcal{I}_m^{r,\text{Ideal}}$) results in small performance degradation.

- Forcing for sparsity (in Figure 6.11b) does *not* yield a significant performance improvement; but this forcing also makes the model more sensitive to sentence selection.

System	r	$\mathcal{I}_m^{r\,\dagger}$	R1	R2	RL
Vanilla	All	N/A	44.03	20.92	40.99
($\gamma = 0.0$)	5	Random	39.06	14.32	36.07
	5	Ideal	43.94	20.82	40.81
$\mathcal{L}_{\texttt{A}}$-tuned	All	N/A	41.62	19.39	38.62
($\gamma = 0.1$)	5	Random	28.43	7.72	24.00
	5	Ideal	44.22	21.01	41.19
$\mathcal{L}_{\texttt{A}}$-tuned	All	N/A	40.28	18.47	37.44
($\gamma = 1.0$)	5	Random	21.38	4.22	17.69
	5	Ideal	43.61	20.46	40.60

Table 6.16 Sparsity and †Selection Method ($\mathcal{I}_m^r$) described above on CNNDM.

Through these results, this experiment verifies that sparsity can be exploited such that constraining the encoder-decoder attention to only 5 sentences can achieve comparable or better performance than attending over all sentences.

Approximated Encoder-Decoder Attention Performance via Neural Approximator

The previous experiment only examined the upper bound performance that could be achieved by exploiting the sparsity. This experiment, on the other hand, investigates the model-based neural approximation outlined in Section 6.4.4. The modified encoder-decoder attention consists of (1) sentence-level attention over N_1 source sentences to select top-r sentences; and (2) word-level attention over rN_2 words.

Baselines of the modified architecture are:

- Vanilla + Random: Vanilla model and inference obtained by random selection ($\mathcal{I}_m^{r,\text{Rnd}}$)
- Vanilla + Ideal: Vanilla model and inference obtained by ideal selection ($\mathcal{I}_m^{r,\text{Idl}}$)

Modified Architecture: It is augmented by the model-based neural approximator for selecting top-r. The selection is done by predicting the sentence-level encoder-decoder attention score ($\mathcal{I}_m^{r,\text{Apx}}$) for each head and each layer. The modified architecture can be in three different setups, which are described in detail in Section 6.4.4:

- KL-only: Training loss is $\mathcal{L}_{\text{KL}}$

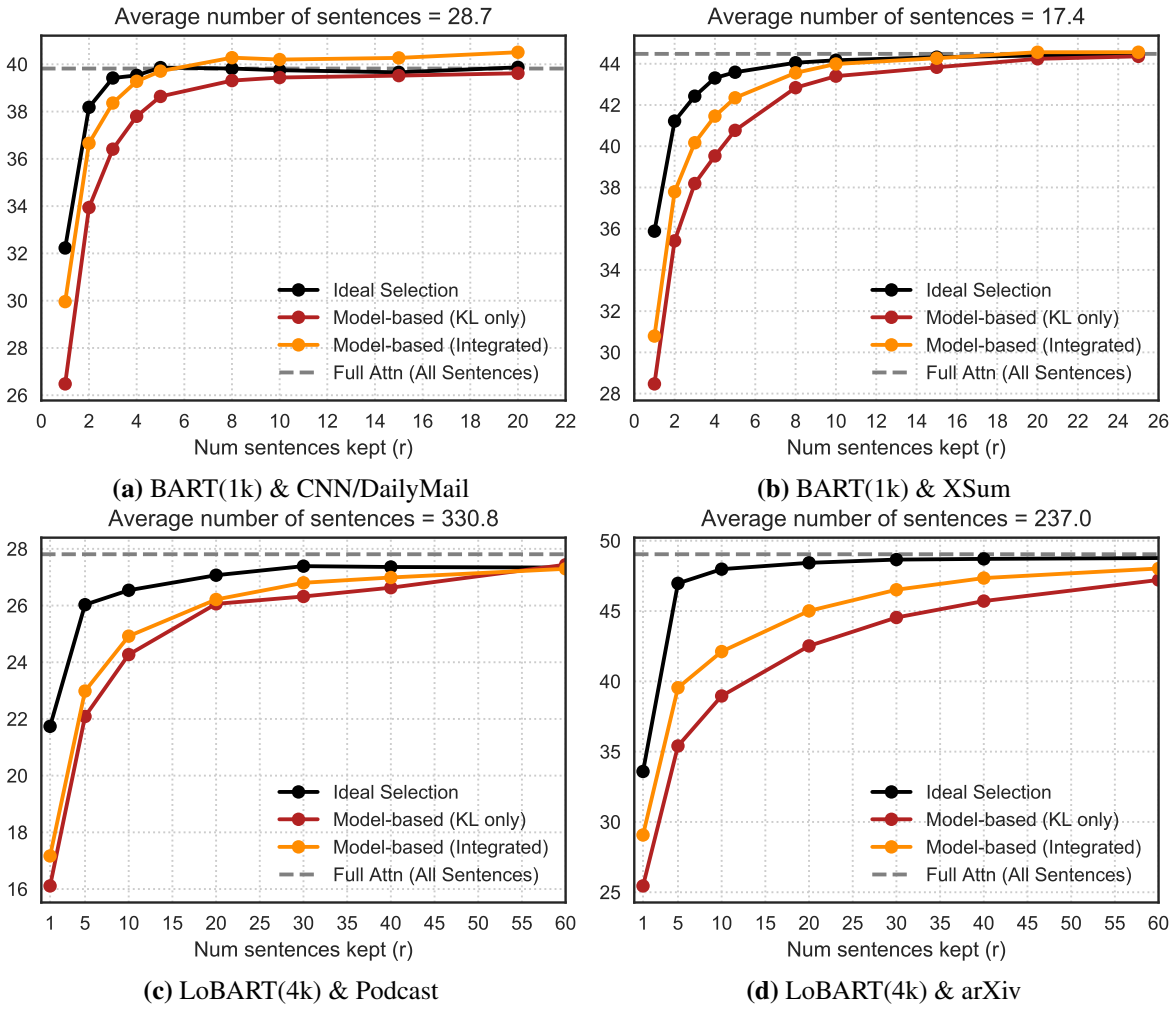
- Integrated Training (Ideal): Training loss is $\mathcal{L}_{\mathtt{I}}$ (in Equation 6.31) where ideal top-r sentences are selected at training, i.e., via $\alpha^{\mathtt{s}}_{m,i}$
- Integrated Training (Approx): Training loss is $\mathcal{L}_{\mathtt{I}}$ (in Equation 6.31) where r sentences are selected at training based on the sentence-level encoder-decoder attention scores predicted by the neural approximator, i.e., via $\tilde{\alpha}^{\mathtt{s}}_{m,i}$
- Integrated Training (Mixed): Training loss is $\mathcal{L}_{\mathtt{I}}$ (in Equation 6.31) where r sentences are selected at training using $\alpha^{\mathtt{s}}_{m,i}$ with probability $1 - \frac{\text{step}}{\text{epoch\_size}}$, otherwise $\tilde{\alpha}^{\mathtt{s}}_{m,i}$

The results in Table 6.17 show that the modified architecture with KL-only training clearly outperforms the random selection baseline, and the performance degradation of this system can be further reduced by integrated training. The performance level of the modified architecture is close to the ideal selection baseline. These results verify the *effectiveness* of the modified archirecture that attends to a subset of sentences. Also, Table 6.17 shows that the best setup is integrated training (Approx), which uses $\mathcal{I}^{r,\text{Apx}}_m$ as reference in training. This result is likely because integrated training is initialized from the KL-only setup.

Model	Train	Inference	R1	R2	RL
Vanilla	✗	All	44.03	20.92	40.99
Vanilla	✗	$\mathcal{I}^{r,\text{Rnd}}_m$	39.06	14.32	36.07
Vanilla	✗	$\mathcal{I}^{r,\text{Idl}}_m$	43.94	20.82	40.81
Modified Arch.	KL-only	$\mathcal{I}^{r,\text{Apx}}_m$	43.02	20.02	39.89
Modified Arch.	Integrated Training (Ideal)	$\mathcal{I}^{r,\text{Apx}}_m$	43.03	20.04	40.05
Modified Arch.	Integrated Training (Approx)	$\mathcal{I}^{r,\text{Apx}}_m$	43.72	20.40	40.70
Modified Arch.	Integrated Training (Mixed)	$\mathcal{I}^{r,\text{Apx}}_m$	43.31	20.21	40.35

Table 6.17 Performance on CNNDM where $r = 5$ for both training and inference stages. KL-only = Model $\tilde{\boldsymbol{\theta}}$ trained on $\mathcal{L}_{\text{KL}}$; Integrated Training = $\boldsymbol{\theta}_{\text{dec}}$ and $\tilde{\boldsymbol{\theta}}$ trained on $\mathcal{L}_{\mathtt{I}}$. Rnd = Random, Idl = Ideal, Apx = Approximation, Mix = Scheduled between Idl and Apx.

In addition to CNNDM, the modified architecture is applied to a BART system trained on XSum ($\leq$1k words), and to LoBART trained on Podcast and arXiv ($\leq$4k words). Figure 6.18 shows that in all datasets investigated, the performance of the modified architecture (both using KL-only and integrated training) converges to the ideal selection upper bound. Also, the system trained using integrated training achieves better performance than the system trained with KL-only. These results also confirm that the performance of the proposed method converges to that of the full attention baseline across all models and datasets.



(a) BART(1k) & CNN/DailyMail

(b) BART(1k) & XSum

(c) LoBART(4k) & Podcast

(d) LoBART(4k) & arXiv

Fig. 6.18 Performance (ROUGE-1) of BART & LoBART. The integrated training is based on $\mathcal{I}_m^{r,\text{Apx}}$.

Optimal number of sentences: This experiment investigates the optimal number of sentences r^* which is the number of sentences that the ideal selection system's performance plateaus or reaches the full attention baseline's performance. Shown in Figure 6.18, the values of r^* are approximately 5, 10, 30, and 30 for CNNDM, XSum, Podcast, and arXiv, respectively. Although XSum has fewer sentences on average compared to CNNDM, $r^*_{\text{XSum}} > r^*_{\text{CNNDM}}$. This is because XSum is more abstractive; hence, requires information from more sentences. For longer summarization tasks as shown by Podcast and arXiv, the performance degradation appears larger, meaning that the task of constraining to salient sentences in longer tasks is more challenging, and larger r is required.

In summary, these results demonstrate empirically that a neural network can predict sparsity, therefore, allowing efficient sentence selection. The summarization performance of the

modified encoder-decoder approach converges to that of the full attention baseline, while switching the complexity from $\mathcal{O}(MN_1N_2)$ to $\mathcal{O}(MN_1 + k_wMrN_2 + k_eN_1N_2)$.

However, this framework requires additional modules to the original attention mechanism, and the real gain in speed will depend on the balance of the sparsity against the computational cost of the additional modules. Consequently, the challenge is to make these additional modules highly efficient. As the particular network realization selected in this experiment is to show the feasibility of the framework, it may have limitations which could prevent it from yielding a wall-clock time improvement in practice. The choice of using RNN for the sentence encoder in Equation 6.28 leads to additional computational cost (specifically k_e), and this additional cost could be high as the computational cost of RNN grows with $N_1N_2D^2$. As the goal is to obtain a sentence-level representation, it is possible to replace the RNN with a hierarchical attention that runs over sentences. This would instead lead to a computational cost that grows with N_1N_2D. Sentence-level query and key mappings in Equations 6.26 and 6.27 could also incur a large computational cost.

6.5.7 Ensemble of Summarization Models

Before concluding this chapter on abstractive summarization with foundation models, this experiment aims to examine ensemble methods (discussed in Section 2.4), which can be useful in obtaining the best performance. For example, in machine learning competitions, one could build a class of models on a dataset to achieve the best performance. Although system combination is common in the areas of speech recognition [70, 78] and machine translation [146, 273, 265, 73], applying ensemble methods to natural language generation (NLG) tasks such as *summarization* can be challenging due to the diverse set of possible outputs, and it has received less attention in the NLG research. This experiment investigates ensemble methods for summarization, and this section summarizes ensemble-based systems submitted to the Spotify Podcast Summarization Challenge at TREC 2020 and the Medical Problem List Summarization (ProbSum) at BioNLP 2023.

Spotify Podcast Summarization Challenge 2020

Hosted at TREC[14] 2020, this challenge is to build a summarization system for generating a summary that captures the most important information for a podcast (given an audio file and ASR transcript). Our systems are fine-tuned BART and fine-tuned BART with the sentence filtering with HIER model (described in Section 5.4). In order to achieve

[14]Text REtreival Conference (TREC): https://trec.nist.gov/

the best performance possible, we applied sequence-level training loss $\mathcal{L}_{\text{SL}}$ (described in Section 2.2.2) and token-level ensemble.

This challenge adopted ROUGE as an automatic evaluation method during development. The results in Table 6.18 show the: (1) applying sentence filtering at both training and inference stages improves the BART-based system; (2) As a backbone, BART fine-tuned on XSum is better than BART fine-tuned on CNNDM. This is likely because XSum is closer to Podcast (e.g., higher compression ratio and more abstractive) than CNNDM; (3) sequence-level training (via $\mathcal{L}_{\text{SL}}$ loss) significantly improves the ROUGE scores, similar to the findings in previous work [224]; (4) ensemble further improves the performance, and the ensemble with different data shuffles is more effective than the ensemble with different checkpoints. This is likely because of the diversity in the ensemble members.

System	**R1**	**R2**	**RL**
*Fine-tuned BART-CNNDM	26.57	9.14	23.43
Fine-tuned BART-CNNDM (w/ HIER at inference)	26.96	9.53	23.70
Fine-tuned BART-CNNDM (w/ HIER at training and inference)	26.96	9.75	23.71
*Fine-tuned BART-XSum (w/ HIER at training and inference)	27.10	9.96	23.92
†Fine-tuned BART-XSum (w/ HIER at training and inference) + $\mathcal{L}_{\text{SL}}$	27.91	10.25	24.67
Ensemble of 3 shuffles	28.56	10.83	25.23
Ensemble of 3 checkpoints	28.12	10.44	24.87
*Ensemble of 3 shuffles $\times$ 3 checkpoints	**28.57**	**10.86**	**25.28**

Table 6.18 ROUGE F_1 scores on the test set of Podcast (SPTF-test). *These systems were submitted to the competition, and human evaluation was performed by NIST as shown in Table 6.19. †The base model for each ensemble is Fine-tuned BART-XSum (w/ HIER at training and inference) + $\mathcal{L}_{\text{SL}}$.

Subsequently, four systems (noted by * in Table 6.18) were submitted to the competition. NIST[15] human annotators evaluated the summaries of 179 randomly selected podcast episodes. Human evaluation was performed on a 4-point scale (Excellent, Good, Fair, Bad) as well as eight Yes/No questions. For example, Q1 is "Does the summary include names of the main people (hosts, guests, characters) involved or mentioned in the podcast?". The exact guideline for human evaluation can be found in Appendix A.

Human evaluation results in Table 6.19 show that the ensemble of 3 receives the highest human rating at 1.777 on average, compared to the BART baseline at 1.564 and creator-provided description at 1.291. We note that on this subset of 179 episodes, the ensemble

[15]The National Institute of Standards and Technology (NIST)

of 3 also achieves ROUGE-L similar to the ensemble of 9. These results demonstrate the effectiveness of the token-level ensemble technique as measured by both automatic and human evaluation, and our ensemble approach achieved the best result out of 8 participating teams in the summarization challenge.

System	Avg	%E	%EG	%EGF
Creator Description	1.291	15.6	39.7	73.7
Fine-tuned BART-CNNDM	1.564	21.8	50.3	84.4
Fine-tuned BART-XSum w/ HIER†	1.687	25.7	56.4	86.6
Ensemble of 3 shuffles	**1.777**	26.3	**58.7**	**92.7**
Ensemble of 3 shuffles × 3 checkpoints	1.704	**27.4**	58.1	84.9

Table 6.19 NIST Evaluation on randomly selected 179 episodes in the test set. Avg = Average of Excellent (3 points), Good (2 points), Fair (1 point), Bad (0 points). %E is the percentage of episodes graded Excellent, %EG is the percentage of episodes graded Excellent or Good, and %EGF is the percentage of episodes graded Excellent, Good, or Fair. †HIER sentence filtering is applied at both training and inference.

System	Q1	Q2	Q3	Q4	Q5	Q6(↓)	Q7	Q8
Creator Description	55.9	34.1	72.1	55.3	63.7	3.4	77.7	52.0
Fine-tuned BART-CNNDM	**63.7**	**44.1**	82.7	61.5	74.9	**6.7**	88.3	70.4
Fine-tuned BART-XSum w/ HIER†	62.0	43.0	87.2	**63.1**	79.3	7.8	**88.8**	73.7
Ensemble of 3 shuffles	**63.7**	39.1	**89.9**	**63.1**	**80.4**	7.8	88.3	**77.1**
Ensemble of 3 shuffles × 3 checkpoints	62.0	40.8	86.0	60.3	**80.4**	10.1	86.6	76.5

Table 6.20 NIST Evaluation on randomly selected 179 episodes in the test set. Q1-Q8 are the percentage of *yes* answers where a higher number is better except Q6. †HIER sentence filtering is applied at both training and inference.

Medical Problem List Summarization Challenge (ProbSum at BioNLP) 2023

The Problem List Summarization Task (Shared Task 1A) at the BioNLP Workshop 2023 is to summarize patients' medical progress notes in a limited data setting [80]. The data consists of 765 progress notes along with medical summaries and the test set (held-out) consists of 237 progress notes. This task has two important challenges. First, the data is in the medical domain. Second, the data is small in size. To address the medical domain problem, this work employs Clinical-T5 [157], which is a T5 model initialized from scratch and pre-trained

on the union of MIMIC-III and MIMIC-IV databases.[16] Clinical-T5 is fine-tuned to the ProbSum training set. To address limited training data, ensemble techniques are used. This is because when working with a small dataset, individual models could be prone to overfit specific aspects of the data. By training multiple models on the same dataset, each model can potentially capture different aspects of the data.

Ensemble Baseline: Firstly, this work compares different ensemble methods, including weight averaging, token-level ensemble, and MBR decoding. A progress note in this dataset consists of *assessment*, *subjective*, and *objective* parts. To study the importance of each section, initial experiments trained a T5 model on each section individually and found that the assessment part is the most informative in generating the summary.

For simplicity, the notation is $\boldsymbol{\theta}_{\texttt{A}}$ for model trained on *assessment*-only input part and $\boldsymbol{\theta}_{\texttt{AS}}$ for model trained on *assessment*+*subjective* input. This work trains nine individual $\boldsymbol{\theta}_{\texttt{A}}$ models and nine individual $\boldsymbol{\theta}_{\texttt{AS}}$ models where all models are initialized from Clinical-T5-Large weights, and have the same training hyperparameters except random seeds for data shuffling. The results in Table 6.21 show that: (1) Weight averaging does not yield an improvement over a single model. This is likely because individual models end up in a different region of the loss surface. A possible improvement is using *Greedy Soup* [318] where a new model is added only if it improves weight averaging performance on some held-out data; (2) Both token-level ensemble and MBR decoding yield better performance than single models. This suggests that combining models at a higher level could improve the robustness and an ensemble less sensitive to the training loss surface.

Method	**ROUGE-L**	
	$\boldsymbol{\theta}_{\texttt{A}}$	$\boldsymbol{\theta}_{\texttt{AS}}$
Individual	29.84±0.69	29.44±0.45
Weight Averaging	29.39	28.00
Token-level Ensemble	30.50	30.04
MBR Decoding	30.72	30.30

Table 6.21 ROUGE-L on the test data of ProbSum. This table compares combination methods of nine `A` models and nine `AS` models. ± indicates a standard deviation using 9 systems.

Hierarchical Ensemble of Summarization Models (HESM): As an extension to standard ensemble techniques, this experiment investigates the Hierarchical Ensemble approach

[16] Initially, zero-shot prompting, oracle extractive summarization, and abstractive summarization based on T5 and Clinical-T5 approaches are compared using 5-fold cross-validation. The results showed that Clinical-T5 achieved the best performance, so it was selected as the base model in the subsequent ensemble experiment.

proposed in Section 2.4. This experiment utilizes the previous nine $\boldsymbol{\theta}_{\mathtt{A}}$ models and nine $\boldsymbol{\theta}_{\mathtt{AS}}$ models. The results of different hierarchical combination setups are provided in Table 6.22. The first block shows the performance when combining one $\boldsymbol{\theta}_{\mathtt{A}}$ and one $\boldsymbol{\theta}_{\mathtt{AS}}$ in a token-level ensemble, followed by an MBR combination stage over 9 of these ensembles. Similarly, the second block shows the performance when combining three $\boldsymbol{\theta}_{\mathtt{A}}$ and three $\boldsymbol{\theta}_{\mathtt{AS}}$ each in a token-level ensemble fashion followed by an MBR decoding stage over 3 of these ensembles. Overall, we observe gains from the hierarchical ensemble, and based on this, the HESM system achieved the best performance out of 10 participating teams in the ProbSum competition [80].

Method	**Num in each TokEns** $\boldsymbol{\theta}_{\mathtt{A}}$	$\boldsymbol{\theta}_{\mathtt{AS}}$	**MBR**	**Num. Models**	**ROUGE-L**
Token-level Ensemble	1	1	✗	2	31.17±0.67
HESM	1	1	9	18	32.31
Token-level Ensemble	3	3	✗	6	31.50±0.42
HESM	3	3	3	18	31.87
HESM	3	3	9	54	31.88

Table 6.22 ROUGE-L of HESM on the test data. (a, b) denotes token-level ensemble consisting of a $\boldsymbol{\theta}_{\mathtt{A}}$ models and b $\boldsymbol{\theta}_{\mathtt{AS}}$ models. MBR $= c$ denotes the outputs of c token-level ensembles combined using MBR decoding. For HESM(3,3) w/ MBR=3, ensembles with non-overlap members are chosen.

6.6 Chapter Summary

This chapter investigated abstractive summarization in the context of utilizing foundation models. Given the enormous size of foundation models, it can be difficult to simply fine-tune them on long-input summarization tasks. Thus, the chapter has examined two complementary directions to use foundation models on abstractive summarization.

First, sentence filtering can reduce the number of tokens to be within the limit of the base model, and applying the oracle with random padding method at the training time and the model-based MCS method at inference time was the most effective. Second, this chapter studies efficient models where local attention is applied to the BART model. Through experiments, local attention adaption was shown effective as well as complementary to sentence filtering. The combined approach achieved state-of-the-art summarization performance on various long-input summarization tasks at the time of conducting the experiments. In addition, this section shows that there is sparsity in the encoder-decoder attention that allows us to reduce the computational cost with minimal degradation. The experiments showed

that the summarization performance of the modified encoder-decoder attention that exploits the sentence structure and sparsity converges to that of the full attention baseline. This chapter also investigated ensemble methods for summarization models, which resulted in the performing systems in two summarization competitions.

So far, Chapter 5 and Chapter 6 have only focused on improving abstractive summarization, and the performance has been measured by the ROUGE score. However, as the summarization systems improve, they can reach the point where the ROUGE score starts correlating only *weakly* with human judgements [50]. Another aspect is that these advanced summarization systems, especially those with foundation models, are capable of generating highly *fluent* texts. However, they may not necessarily generate summaries that are *consistent* with the source documents. Therefore, the focus of the next chapter will be *assessing* the information consistency between the source documents and the summary.

Chapter 7

Automatic Summary Assessment

Chapter 5 and Chapter 6 discussed abstractive summary generation. The generated summaries were evaluated against reference summaries using automatic metrics (e.g., ROUGE) or human evaluation (e.g., in the Spotify Challenge in Section 6.5.7). In contrast, this chapter focuses on automatic summary assessment without gold-standard references. The goal of automatic summary assessment is to rank generation systems or rank hypothesis summaries. First, Section 7.1 discusses the limitations of existing assessment methods, which necessitates further improvements of the assessment methods. Section 7.2 focuses on information consistency via question answering (QA) where existing QA based approaches and their limitations are first discussed, and then this section proposes Multiple-choice Question Answering and Generation (MQAG), which can assess information consistency without span-based comparisons. Section 7.3 focuses on zero-shot methods, which can assess any aspect of summarization, based on large language models (LLMs) prompting. Zero-shot methods include standard LLM prompting and a novel LLM comparative assessment framework. Subsequently, Section 7.4 includes experiments, results, and discussions.

7.1 Motivation

Assessment, or evaluation methods, are important for the comparison and improvement of automatic generation systems. As manual evaluation can be time-consuming, tedious, and difficult to scale and reproduce, it necessitates the need for automatic assessment methods. Designing a reliable automatic method for natural language generation (NLG) task is challenging and is still an open problem for researchers. Summarization, which is one of the NLG tasks, can be even more difficult due to the specific requirements in each use case. For example, given a source document, there could be multiple good summaries which could

also depend on the specific requirements, while in contrast in machine translation, the set of possible outputs is less diverse.

The background chapter (in Section 4.3) discussed summary assessment and existing methods, including reference-based and reference-free methods. For example, *n*-gram based methods such as ROUGE [162] are widely used in summary assessment. However, ROUGE is reference-based, and it has been shown to correlate weakly with human judgements [215, 229, 12, 67, 50, 49]. Furthermore, modern summarization systems, such as those based on large pre-trained language models [159, 340], are capable of generating highly fluent output; however, these systems have been shown to generate false or unsupported information [143]. This phenomenon is now commonly referred to as *hallucination* [348, 193, 123]. In particular, it has been shown that the current abstractive summarization systems tend to be less faithful, on average, as they are more abstractive [61, 204, 150]. As a result, the focus of developing automatic methods has been on *information consistency*, which is to evaluate whether the generated outputs are factually consistent with their source documents.

To assess information consistency, several forms of methods have been proposed such as knowledge representation [90], textual entailment [68, 143, 193], and question answering (QA) [66, 297, 61, 261, 49]. These methods have been described in Section 4.3. The QA-based methods, in particular, have shown promising results in assessing information consistency, and Section 7.2 will delve into the details of QA-based methods in addition to Section 4.3 as well as propose an improvement to QA-based methods.

In addition, given the impressive zero-shot performance of large language models (LLMs) across various natural language tasks [15, 31], interesting questions arise. For example, whether, or to what extent, LLMs can be used in automatic summary assessment (as well as broader generative tasks), and what prompts or setups are effective in using existing LLMs. To this end, Section 7.3 will delve into summary assessment using zero-shot LLMs.

7.2 Information Consistency via Question Answering

A question answering (QA) approach consists of a question-generation system and an answering model. Figure 7.1 illustrates a general framework for information consistency assessment. For example, in QA-based method, knowledge representation is done by question generation, and a comparison is done by question answering.

This question answering approach is motivated by *extrinsic* evaluation [277], a process by which the quality of the summary is evaluated by its impact on a downstream task such as document categorization [191], information retrieval [240], and question answering [202, 37]. Extrinsic evaluation using a downstream question answering has been studied, for example,

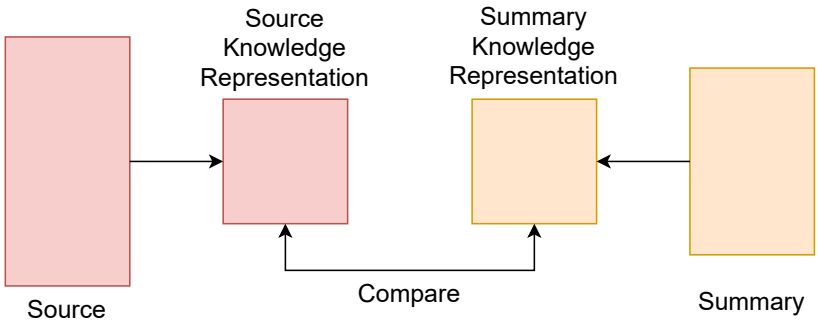


Fig. 7.1 General Framework for Information Consistency Assessment

Morris et al. [202] used four multiple-choice GMAT[1] reading comprehension tests and measured how many questions were answered correctly when presenting different contexts, including source documents, their summaries, and no context. To automate an extrinsic evaluation with question answering, QA-based methods are designed to automatically measure to what extent a summary provides sufficient information to answer questions derived from the source document [66, 297, 61, 261]. Before discussing our proposed improvement to QA methods, below will first describe span-based methods.

7.2.1 Span-based Question Answering

Generally, question answering based methods work as follows. Given automatically generated questions, the first answer is derived from the source, the second answer is derived from the summary, and then the two answers are compared. The answers are generally in the form of extractive text spans, so we refer to these span-based question answering and generation methods as **SpanQAG**.

Following Section 4.3, the notation for the formulae measuring the consistency is as follows:

- X = source document
- Y = summary (to be assessed)
- Y^* = reference summary (if available)
- Q = question
- A = answer

[1] Graduate Management Admission Test

- $P_{\mathtt{G}}(.)$ = question generation, which is a probability distribution over question (and answer) given a context
- $P_{\mathtt{A}}(.)$ = question answering, which is a probability distribution over options given a question and a context
- $f(.)$ = answer verification, which is a function to score two answers

Question Generation (QG): One of the challenges with a QA-based approach is question generation, which is the process used to generate a good series of questions. The QG stage is generally modelled by the distribution $P_{\mathtt{G}}(.)$ where the most basic form is $P_{\mathtt{G}}(Q|\text{context})$. This QG model could be trained on a reading comprehension dataset such as NewsQA [287], RACE [151] or SQuAD [247] as a sequence-to-sequence task. A good QG model should cover all aspects of the context. Good coverage can be achieved by, for example, conditioning the question generation on both context and entity, i.e. $P_{\mathtt{G}}(Q|\text{context}, \text{entity})$, where important entities could be extracted using name entity recognition [297]. If computational cost is not a constraint, an alternative method is Monte Carlo approximation where a series of questions and answers are sampled from the distribution $P_{\mathtt{G}}(Q|\text{context})$.

Question Answering (QA): Given a question and a context, question answering is to output an answer. This QA stage is modelled by $P_{\mathtt{A}}(A|Q, \text{context})$. For example, given the extractive nature of the task, an encoder-only model such as BERT is usually used as a QA system [297]. For each token, this model predicts the probability of the token being the start or the end of the extractive span.

Answering verification and scoring $f(.)$: Once two answers are obtained, they are then compared using $f(.)$ to get the score. This process is referred to as answer verification (i.e., answer comparison), and it is typically implemented using exact token matching, token overlap F1, BERTScore, or a learned metric [51]. Additionally, Dong et al. [56] proposed to correct unfaithful entities in summaries using span-based question answering such that an entity in a summary is iteratively masked and an extractive answering is used to verify the entity using the source document. Nevertheless, given the nature of existing span-based methods where the answering system extracts answer spans before the two answer spans are compared, these methods have limitations in that they require span comparison in the answer verification stage, which can be challenging in abstractive summarization. To avoid span-based answer verification, the next section proposes an alternative question answering-based approach where multiple-choice question generation and answering systems are used where the answers are now in the form of *probability distributions* rather than *text spans*.

Having described a general framework of a QA-based approach, which consists of a QG system, a QA system, and an answer verification method, here we provide examples of existing QA-based approaches. The following QA-based methods are grouped into:

- *recall-oriented*: it measures how much of the salient information (e.g., in the source document or reference summary) can be determined using the summary.
- *precision-oriented*: it measures how much of the information in the summary can be verified using the source document.
- *both*: it combines recall-oriented and precision-oriented methods.

Recall-oriented methods

• **APES**, proposed by Eyal et al. [66], is one of the first QA-based methods. APES generates questions from the reference summary, i.e. $P_{\mathtt{G}} = P_{\mathtt{G}}(Q|Y^*)$, and it uses a QA system to determine the total number of questions answered correctly according to the summary. For each summary to assess, questions are successively generated from a reference summary, by masking each of the named entities in the reference, and the masked entities are treated as target answers. The answering system $P_{\mathtt{A}}$ in APES is based on Chen et al. [20]'s model.

$$\text{APES}(X,Y) = \mathop{\mathbb{E}}_{Q,A\sim P_{\mathtt{G}}(Q,A|Y^*)}[f(A,P_{\mathtt{A}}(A|Q,Y))] \tag{7.1}$$

The expectation in Equation 7.1 is approximated using samples drawn from $P_{\mathtt{G}}(Q,A|Y^*)$. This method can be considered recall-oriented as it measures how much of the salient information in the reference summary can be determined using the summary.

• **SummQA**, proposed by Scialom et al. [262], extended APES by generating the questions from the source document, i.e. $P_{\mathtt{G}} = P_{\mathtt{G}}(Q|X)$, and probing the summary for information retrieved from the input text. This method is recall-oriented as it measures how much of the salient information in the source document (as represented by the generated questions) can be determined using the summary.

$$\text{SummQA}(X,Y) = \mathop{\mathbb{E}}_{Q,A\sim P_{\mathtt{G}}(Q,A|X)}[f(A,P_{\mathtt{A}}(A|Q,Y))] \tag{7.2}$$

Similar to APES, SummQA masks entities to generate questions, and masked entities are used as reference answers. SummQA weighs each question equally, so it lacks a way to select questions that reflect the most important information of the input compared to APES

assuming that the reference summary Y^* already contains only salient information. Hence, the authors of SummQA later proposed QuestEval [261] as an improvement.

Precision-oriented methods

• **QAGS**, proposed by Wang et al. [297], assesses the faithfulness of a summary based on the hypothesis that the summary Y is considered unfaithful (with respect to the source document X) if, given a question, the answer extracted from X is different from the answer extracted from Y. This measure is precision-oriented as it measures how much of the information in the summary (as represented by the generated questions) can be verified using the source.

$$\text{QAGS}(X,Y) = \mathop{\mathbb{E}}_{Q\sim P_{\text{G}}(Q|Y)} [f(P_{\text{A}}(A|Q,X), P_{\text{A}}(A|Q,Y))] \tag{7.3}$$

The QG system is based on BART fine-tuned to the NewsQA dataset [287] and the QA system is based on BERT fine-tuned on the SQuAD-2.0 dataset [246].

• **FEQA**, proposed by Durmus et al. [61], is precision-oriented similar to QAGS. This approach masks important text spans (e.g., noun phrases, entities) in the summary. Each span is considered the gold answer, and the corresponding question is generated using the QG model. The QA model then finds answers to these questions in the source document, and the answers are compared against the gold answers.

$$\text{FEQA}(X,Y) = \mathop{\mathbb{E}}_{Q,A\sim P_{\text{G}}(Q,A|Y)} [f(A, P_{\text{A}}(A|Q,X))] \tag{7.4}$$

The QG system is based on BART fine-tuned to adapted QA2D dataset [47] and the QA system is based on BERT fine-tuned to SQuAD-1.1 [247] or SQuAD-2.0 [246].

Recall and Precision oriented (QuestEval)

As an extension to precision-oriented QAGS/FEQA and recall-oriented SummQA, Scialom et al. [261] proposed QuestEval. This approach generates questions from both the source and the summary separately to obtain a precision score and a recall score, and QuestEval is a combination of the precision-based and recall-based scores. Compared to SummQA, QuestEval-Recall is different in that it assigns a weighting function $g(Q,X)$ to take into account the importance of each question with respect to the source document X, and a trained model is used to predict the importance score.

$$\text{QuestEval-Precision}(X,Y) = \mathop{\mathbb{E}}_{Q,A\sim P_{\mathtt{G}}(Q,A|Y)} \left[f\left(A, P_{\mathtt{A}}(A|Q,X)\right)\right] \tag{7.5}$$

$$\text{QuestEval-Recall}(X,Y) = \mathop{\mathbb{E}}_{Q,A\sim P_{\mathtt{G}}(Q,A|X)} \left[g(Q,X) \cdot f\left(A, P_{\mathtt{A}}(A|Q,Y)\right)\right] \tag{7.6}$$

Finally, the QuestEval score is the harmonic mean (i.e., F1-score) of their precision and recall scores. Both QG and QA systems are trained on SQuAD-2.0 [246] with a T5 backbone.

Lastly, we note that SpanQAG methods are based on a hypothesis that the quality of a generated summary is linked to the number of (relevant) questions that can be answered by using the summary, and the relevancy of questions is modelled by the importance score in QuestEval [261] or unanswerability in QAGS [297].

7.2.2 Multiple-choice Question Answering and Generation (MQAG)

Since current summarization systems generate highly fluent summaries, the focus is now on assessing whether summaries contain the same information as that of the source, or whether it is contradictory. One way to view information would be to consider the set of questions that are answerable given a certain passage. If a summary is consistent with the source, then one would expect the set of answerable questions by the summary to overlap with those of the source and yield similar answers. Though span-based QA approaches are similarly motivated, existing span-based frameworks use text similarity measures, either in the form of lexical or representation space. In contrast, we attempt to measure information using multiple-choice questions, which allows for a more abstract understanding of information and enables the convenient use of standard information-theoretic measures. In addition, a summarization system may inject *world knowledge* that it acquired during pre-training. For example, a summarization system could contain an entity which is not directly in the source, but the entity can be linked via reasoning paths using world knowledge as observed by Dong et al. [57]. For example, the source may only mention the city "Cambridge", while the summary may refer to it as "UK". In this example, it can be challenging for span-based methods, while we expect multiple-choice systems to be capable of matching this example.

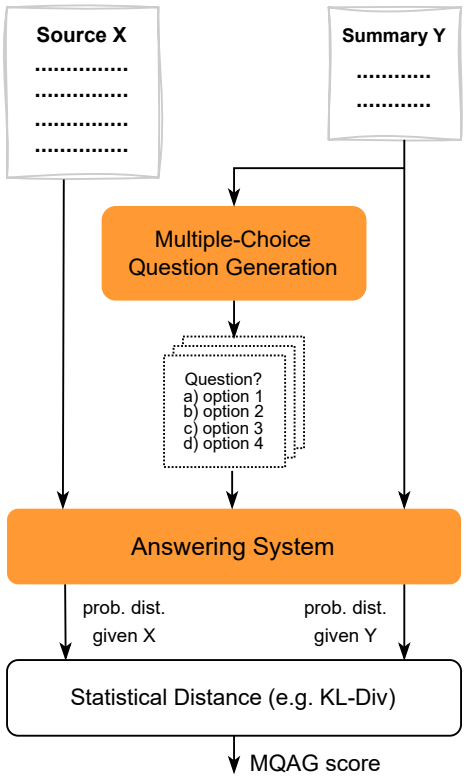


Fig. 7.2 Multiple-choice Question Answering and Generation (MQAG) framework. The answers are represented by probability distributions over choices instead of text spans in span-based question answering approaches.

MQAG Score as Information Consistency Measure

Notation: X = source document, Y = summary, Q = question, and $O = o_1, o_2, o_3, \ldots$ = options associated with the question Q.

The Multiple-choice Question Answering and Generation (MQAG) framework is illustrated in Figure 7.2, and MQAG computes information (in)consistency as follows,

$$\mathcal{I}(X,Y) = \mathop{\mathbb{E}}_{Q,O\sim P_{\mathtt{G}}(Q,O|Y)} \left[\mathcal{D}\left(P_{\mathtt{A}}(O|Q,X), P_{\mathtt{A}}(O|Q,Y)\right)\right] \tag{7.7}$$

$$\mathcal{I}(X,Y) = \int_{Q,O} \mathcal{D}\left(P_{\mathtt{A}}(O|Q,X), P_{\mathtt{A}}(O|Q,Y)\right) P_{\mathtt{G}}(Q,O|Y) \mathrm{d}O\mathrm{d}Q \tag{7.8}$$

$$\mathcal{I}(X,Y) \approx \frac{1}{N}\sum_{i=1}^{N} \mathcal{D}\left(P_{\mathtt{A}}(O^{(i)}|Q^{(i)},X), P_{\mathtt{A}}(O^{(i)}|Q^{(i)},Y)\right) \tag{7.9}$$

where $\{Q^{(i)}, O^{(i)}\}$ is sampled from $P_{\mathtt{G}}(Q, O|Y)$, the question-option generation model, $P_{\mathtt{A}}(O^{(i)}|Q^{(i)}, X)$ and $P_{\mathtt{A}}(O^{(i)}|Q^{(i)}, Y)$ are the option distributions given the source and summary, respectively, and $\mathcal{D}$ is a statistical distance such as KL-divergence. Compared to existing SpanQAG such as QAGS [297] that extracts answer spans and performs text comparison, the proposed method computes a probability mass function (PMF) over generated options and then the PMFs can be compared directly. Based on the information inconsistency score in Equation 7.9, we define the MQAG score as,[2]

$$\text{MQAG-Score}(x, y) = 1 - \mathcal{I}(x, y) \tag{7.10}$$

We refer to Equation 7.10 as the **MQAG-Sum** score as the questions are generated from the summary Y. MQAG-sum is precision-oriented similar to QAGS [297] and FEQA [61]. Furthermore, it is possible to generate questions, $\{Q, O\}$ using the source X instead of the summary Y, $\{Q^{(i)}, O^{(i)}\}$ is sampled from $P_{\mathtt{G}}(Q, O|X)$. We will refer to this variant as the **MQAG-Src** score. MQAG-Src is expected to measure the amount of source information present in the summary, i.e., the coverage of the summary, while MQAG-Sum is expected to measure the consistency of the summary with respect to the source. MQAG-Src is recall-oriented similar to SummQA [262]. To account for consistency (i.e., precision) and coverage (i.e., recall) similar to QuestEval [261], we also consider a simple combination,

$$\text{MQAG-F1} = 2 \cdot \frac{\text{MQAG-Sum} \times \text{MQAG-Src}}{\text{MQAG-Sum} + \text{MQAG-Src}} \tag{7.11}$$

Statistical Distances $\mathcal{D}$

Given two probability distributions over options O (e.g., one conditioned on source X, and the other conditioned on summary Y), a statistical distance $\mathcal{D}$ measures the distance between the probability distributions. There are multiple statistical distances which can be used. In this work, we consider some of the main distances and investigate their properties as well as their empirical performance in our MQAG framework as follows,

- KL-Divergence:

$$\mathcal{D}_{\mathtt{KL}} = \sum_{o \in O} P_{\mathtt{A}}(o|Q, X) \log \left(\frac{P_{\mathtt{A}}(o|Q, X)}{P_{\mathtt{A}}(o|Q, Y)} \right) \tag{7.12}$$

[2] If $\mathcal{D} > 1$, when using KL-divergence, the MQAG score can be negative, but the maximum value is 1.0.

- One-Best (i.e. argmax matching):

$$\mathcal{D}_{\texttt{OB}} = \begin{cases} 0, & \text{if } o_x = o_y \\ 1, & \text{otherwise} \end{cases} \tag{7.13}$$

  where $o_x = \arg\max_o P_{\texttt{A}}(o|Q,X)$ and $o_y = \arg\max_o P_{\texttt{A}}(o|Q,Y)$. $\mathcal{D}_{\texttt{OB}}$ simply determines whether the two answers match or not.

- Total Variation:

$$\mathcal{D}_{\texttt{TV}} = \frac{1}{2} \left\| P_{\texttt{A}}(O|Q,X) - P_{\texttt{A}}(O|Q,Y) \right\|_1 \tag{7.14}$$

- Hellinger:

$$\mathcal{D}_{\texttt{HL}} = \frac{1}{\sqrt{2}} \left\| \sqrt{P_{\texttt{A}}(O|Q,X)} - \sqrt{P_{\texttt{A}}(O|Q,Y)} \right\|_2 \tag{7.15}$$

Examples of the properties of the statistical distances on Bernoulli distributions are illustrated in Figure 7.3. It can be seen that KL divergence is unbounded, which means the value can be exceedingly large. One-best, in contrast, is bounded between 0.0 and 1.0; however, one-best is discontinuous. Total variation and Hellinger distance are continuous and bounded between 0.0 and 1.0.

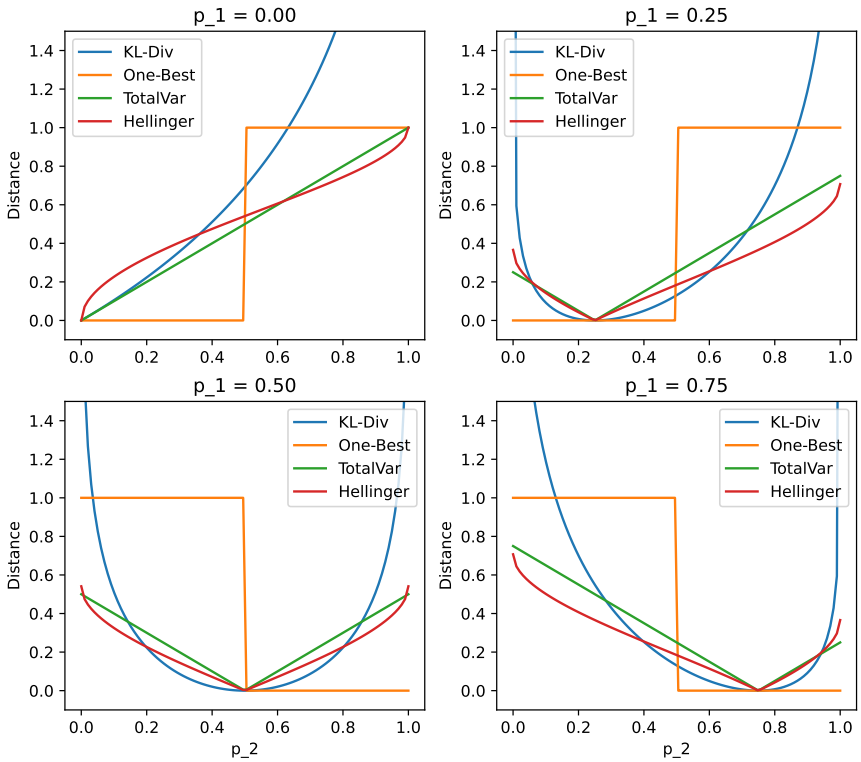


Fig. 7.3 Statistical distances between two Bernoulli distributions $\mathbf{p}_1 = [p_1; 1-p_1]$ and $\mathbf{p}_2 = [p_2; 1-p_2]$ at different values of p_1. We show 4 plots of different values of p_1 = 0.00, 0.25, 0.50, 0.75, and Y-axis represents distance $\mathcal{D}$ and X-axis represents p_2.

7.2.3 MQAG Realization

This section describes question generation system, question answering system, answerability of generated questions, as well as the datasets for system development.

Question Generation (`G1`, `G2`)

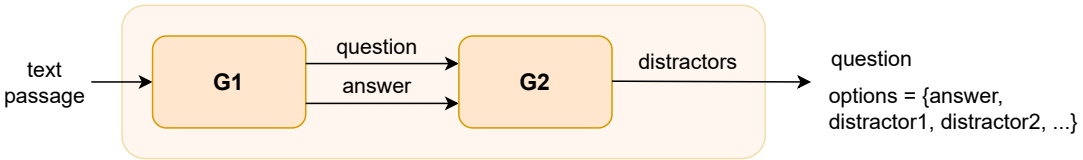


Fig. 7.4 Question Generation Pipeline which consists of two generation systems.

Multiple-choice question generation is implemented in two stages as illustrated in Figure 7.4. The motivation for two generation systems is based on initial experiments which showed that a single-generation system (generating the question and 4 options together) often gave

low-quality distractors, and using two-generation systems improved the quality of distractors. In this setup, the first model G1 generates the question Q and answer A, then the second model G2 generates the distractors $O_{\setminus A}$ given Q and A.

$$P_{\mathtt{G}}(Q,O|Y) = P_{\mathtt{G2}}(O_{\setminus A}|Q,A,Y)P_{\mathtt{G1}}(Q,A|Y) \tag{7.16}$$

where $O = \{A, O_{\setminus A}\}$ denotes all options/choices. In this work, we set the number of options to 4 as we use the RACE dataset. Both G1 and G2 are sequence-to-sequence T5-large models [244]. The question answer generation system G1 can be fine-tuned to either RACE or SQuAD, and the distractor generation system G2 is fine-tuned to RACE.

Question Answering (A)

The answering stage contains one model A, which is Longformer-large [10] with a multiple-choice setup illustrated in Figure 7.5 following Yu et al. [335], Raina and Gales [245]. The input to the model is a concatenation of context, question and option. The answering model A is fine-tuned to RACE.

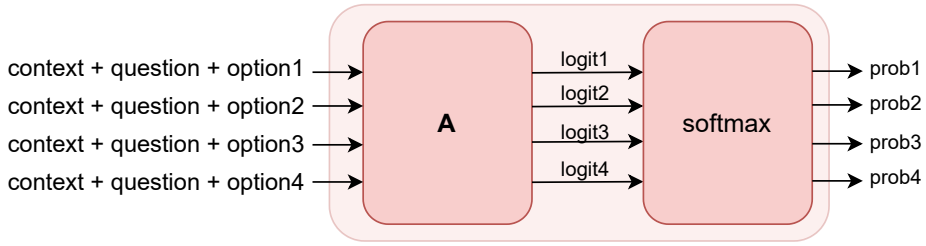


Fig. 7.5 Question Answering System

Answerability of Generated Questions

Because not all generated questions are of high quality, this work considers filtering out low-quality questions through question-context answerability measures [147, 117]. This work considers a simple answerability measure based on the entropy of the probability distribution over the options. We define the effective number of options,

$$\mathcal{N}_Y(Q,O) = 2^{\mathcal{H}[P_{\mathtt{A}}(O|Q,Y)]} \tag{7.17}$$

where

$$\mathcal{H}[P_{\mathtt{A}}(O|Q,Y)] = -\sum_{o\in O}(P_{\mathtt{A}}(o|Q,Y)\log P_{\mathtt{A}}(o|Q,Y)) \tag{7.18}$$

is base-2 entropy, so $\mathcal{N}_Y(Q,O)$ ranges from 1.0 to the number of options, e.g. 4.0. When Q is generated from Y but $\mathcal{N}_Y(Q,O)$ is high, this question Q should be deemed *unanswerable* as it is not answerable even when using the same context. As a result, we use $\mathcal{N}_Y(Q,O)$ as an answerability criterion to *reject* questions which have $\mathcal{N}_Y(Q,O)$ higher than a threshold denoted by $\mathcal{N}_Y^\tau$.

System Development Data

RACE [151] is a multiple-choice reading comprehension dataset where each example consists of context, question, answer, and 3 distractors (i.e., incorrect options). SQuAD [247] is a collection of question-answer pairs derived from Wikipedia articles, and the correct answers can be any sequence of tokens in the given context. The statistics are provided in Table 7.1 where *abstractiveness* is measured by 1.0 minus the length of the longest sequence that exists in both the context and the answer per the answer length: $1.0 - \text{ROUGE-L}_{\text{Precision}}(\text{Answer}, \text{Context})$.

Dataset	Size	Length		Abstractiveness
		Context	Answer	
SQuAD	98.2k	317.8	11.0	0.0%
RACE	97.7k	138.3	11.3	39.1%

Table 7.1 Statistics of datasets for training MQAG systems. Length = the number of tokens. Abstractiveness of 0% indicates that in SQuAD the answer always exists in the context.

7.3 Zero-shot Methods

The previous section proposed MQAG, a bespoke method based on question answering for information consistency assessment. This section, on the other hand, aims to utilize recent advances in NLP where large language models (LLMs) have enabled zero-shot capabilities across various NLP tasks [15, 309, 31]. An interesting application of LLMs is in the automated assessment of natural language generation (NLG), a highly challenging area with great practical benefit. Investigating the ability and application of pairwise comparisons via LLMs has been relatively underexplored, with recent work using pairwise rankings for information text retrieval [237] and separately for assessing LLM-based chat assistants on open-ended questions where outputs are compared to that of a baseline system [26, 351].

Pairwise comparison or comparative assessment [285] is motivated by the fact that humans often find it more intuitive to compare two options rather than scoring each one independently. For example, scoring each example directly usually requires a detailed rubric,

but it can be subjective and, therefore, challenging to perform absolute score prediction. On the other hand, comparative assessment requires only a preference between two options without precisely quantifying the exact level of the assessment. In this section, we explore and compare two options for exploiting the emergent abilities of LLMs for zero-shot NLG assessment: (1) absolute score prediction, and (2) comparative assessment which uses relative comparisons between pairs of candidates. These methods are illustrated in Figure 7.6.

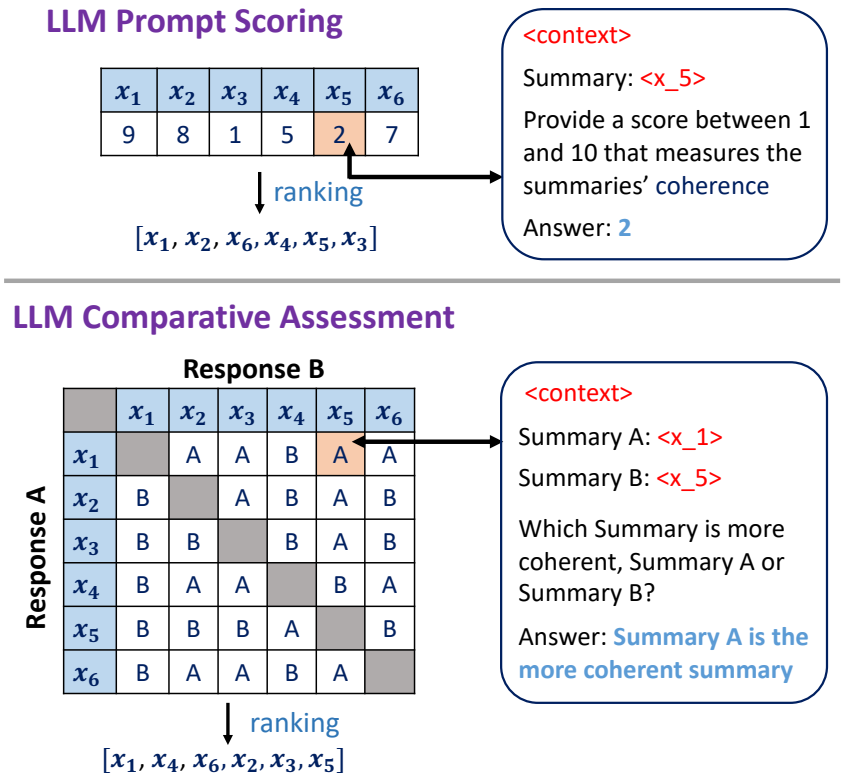


Fig. 7.6 Prompting methods: prompt-scoring and comparative assessment. In prompt-scoring, an LLM is prompted to generate a raw score, while in comparative assessment, the LLM compares candidates in a pairwise manner, and the comparisons are subsequently converted into scores or ranks.

7.3.1 LLM Prompt Scoring

As a baseline for utilizing an LLM in a zero-shot setup, we consider a simple approach by prompting an LLM to assess the quality of a summary between 1-10, for a particular attribute. This approach is referred to as **LLM prompt scoring** shown in Figure 7.6.

Applying ChatGPT in a prompt scoring manner to summary assessment was also investigated in recent works by Wang et al. [301], Kocmi and Federmann [140]. Also, concurrent work, G-Eval [167], extends standard prompt scoring by using detailed prompts (e.g., gener-

ated by an LLM) as well as computing a continuous score through the expected score over a score range (e.g., 1-5 normalized by their probabilities). As the aim of this work is to compare LLM prompt scoring and LLM comparative assessment, we consider simple prompt templates shown in Figure 7.7. We apply prompt scoring on both open-source moderate-sized LLMs such as FlanT5 and Llama2 and closed-source ChatGPT.

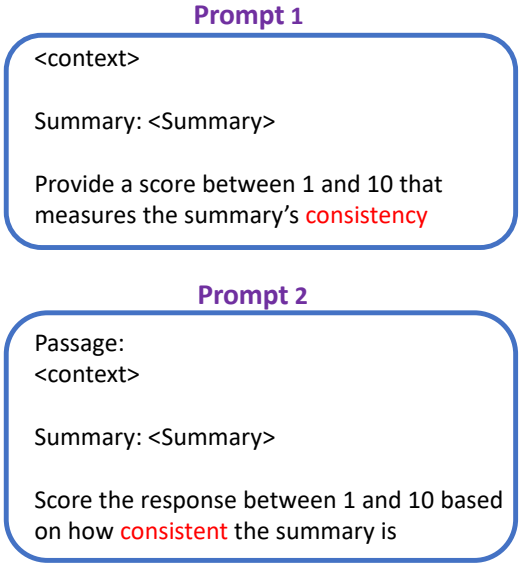


Fig. 7.7 Scoring template 1 and template 2 for the prompt-scoring approach

7.3.2 LLM Comparative Assessment

As an alternative approach to LLM prompt scoring, this work proposes **LLM comparative assessment**. Instead of prompting an LLM to give a score for each individual example, this method prompts an LLM to make a comparison between a pair of examples.

Assume that there is a context X (e.g., a text passage or dialogue) and a set of N candidate responses, $\{Y_1, Y_2, ..., Y_N\}$. For a given attribute (e.g., coherence, consistency, fluency) the N candidates have true underlying scores, $\{u_1, u_2, ..., u_N\}$.[3] The objective is to accurately predict the true ranks, $\{r_1, r_2, ..., r_N\}$, of the candidate scores. In comparative assessment, one uses pairwise comparisons to determine which of the two input responses is better. Let $z_{ij} \in \{0, 1\}$ represent the true outcome of whether y_i is higher ranked than y_j, such that $z_{ij} = \mathbb{1}(u_i > u_j)$. Here, an LLM is used to model the probability that response Y_i is better than response Y_j, p_{ij},

[3] In a comparative assessment setup, predicted scores often only have relative meaning. The relative values, nevertheless, are equivalent to the ranks of the candidates. Thus, the performance of LLM comparative assessment can still be evaluated using system-level or summary-level correlations (described in Section 4.3.3)

$$p_{ij} = P(z_{ij}|Y_i, Y_j, X) \tag{7.19}$$

which can be converted into hard decisions, $\hat{z}_{ij}$, by selecting the most likely outcome,

$$\hat{z}_{ij} = \begin{cases} 1, & \text{if } p_{ij} > 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{7.20}$$

Let $\mathcal{C} = \{c_k\}_{k=1...R}$ represent a set of comparisons, where R is the total number of comparisons, and each comparison $c = (i, j)$ indicates the indices of the two considered candidate responses. For example, the set of all possible comparisons, $\mathcal{C} = \{(i, j) \mid i, j \in [1, ..., N], i \neq j\}$, could be used. As the maximum number of comparisons R grows quadratically with N, a smaller subset of comparisons can be used in order to reduce the computation. This will be discussed in "Comparisons to Ranks". Also, it should be noted that the LLM could be biased in the sense that $p_{ji} \neq 1 - p_{ij}$. Debiasing the LLM is not in the scope of this work, but it is covered in Liusie et al. [173]

Comparative Assessment Prompt Design

To leverage the emergent ability of LLMs, we use comparative prompts that probe a model to decide which of the two candidates is better. Let $f(.|T)$ be a function that converts candidate responses Y_i and Y_j as well as context X into a prompt $\mathcal{P}$ using a prompt template T,

$$\mathcal{P} = f(Y_i, Y_j, X|T) \tag{7.21}$$

The focus is on finding a simple, general and robust assessment method. Extensive prompt engineering is not in the scope of the investigation (despite possible performance gains), but it is an interesting future direction. We evaluate two simple and suitable prompts in our initial investigations. Our prompts for comparative assessment are shown in Figure 7.8.

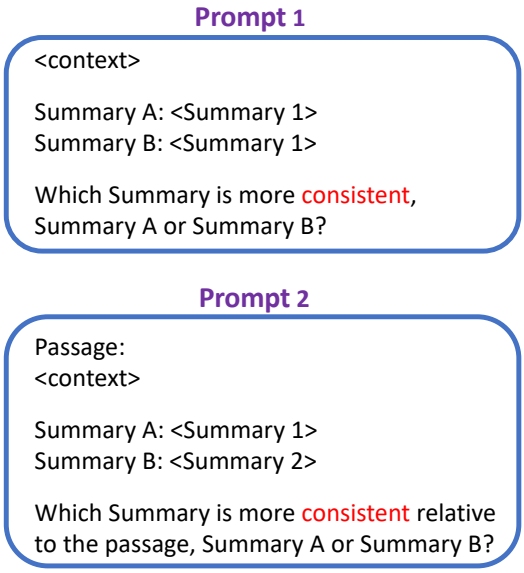


Fig. 7.8 Comparative prompt templates 1 and 2. When assessing different attributes, only the attribute is changed (e.g., consistent → engaging) and for response assessment, the word 'summary' is replaced with 'response'.

A central aspect of LLM comparative assessment is the methodology of obtaining comparative decisions. In this work, we consider two approaches for leveraging LLMs for comparative assessment; first for when one has output token-level probabilities (Prompt-Based Classifier) [174], and second for when only the output texts are available.

• **Prompt-Based Classifier**: If output probabilities are available, an efficient method to get probability estimates of the predictions is to leverage prompt-based classifiers. Let $P(Y|X;\boldsymbol{\theta})$ represent an LLM's conditional language model distribution of the output sequence Y given the textual input X. For prompt-based classifiers, the LM probabilities of specific label words (Y_k) are used as a proxy for the class decisions. For example in summarization assessment, given a prompt $\mathcal{P}$ ending in '... which summary is better', one can set Y_i = '*Summary A*' and Y_j = '*Summary B*' and define the probability that response i is better than response j as:

$$p_{ij} = \frac{P(Y_i|\mathcal{P};\boldsymbol{\theta})}{P(Y_i|\mathcal{P};\boldsymbol{\theta}) + P(Y_j|\mathcal{P};\boldsymbol{\theta})} \tag{7.22}$$

• **Text Generation**: Alternately, if only limited API access is available, responses can be sampled from the conditional LM given the input prompt $\mathcal{P}$,

$$\tilde{Y}^{(k)} \sim P(Y|\mathcal{P};\boldsymbol{\theta}) \tag{7.23}$$

Let $g(\tilde{Y}) \in \{0,1\}$ be a function that maps the text response to the comparative decision, and $\tilde{Y}$ is rejected unless $\tilde{Y}$ is in a predefined set such as $\{Summary\ A, Summary\ B\}$. By generating K output samples from the LLM, one can estimate the comparative probability p_{ij} by looking at the fraction of generated samples that selects Y_i over Y_j.

$$p_{ij} = \frac{1}{K}\sum_{k=1}^{K} g(\tilde{Y}^{(k)}) \tag{7.24}$$

Comparisons to Ranks

Although the full set of possible comparisons yields the most information for the rankings, this requires $R = N \times (N-1)$ comparisons, which can be computationally expensive. For computational efficiency, we can consider 3 different comparison selection strategies:

- **random**: comparisons are randomly selected from the set of all possible comparisons.
- **no-repeat**: if (Y_i, Y_j) is selected then (Y_j, Y_i) will not be selected.
- **symmetric**, if (Y_i, Y_j) is selected, then (Y_j, Y_i) will also be selected.

Given a set of selected comparisons $\mathcal{C}$ and weights of a comparative assessment system $\boldsymbol{\theta}$, one can generate a predicted rank ordering $\hat{r}_{1:N}$ of the candidate responses. A simple but effective approach is to sort the candidates by the **win-loss** ratio.

$$\text{win-loss-ratio of } Y_i = \frac{\text{the number of wins of } Y_i}{\text{the number of comparisons involving } Y_i} \tag{7.25}$$

which can then be ordered to convert the scores into predicted ranks $\hat{r}_{1:N}$.

7.4 Experiments

The experiments in this chapter first examine assessment methods, including model-free and model-based methods, and provide benchmark results for the proposed long-input podcast summary assessment data. Then, with the aim of selecting reference summary-document pairings for training, we apply summary assessment for data selection, and we investigate the

summarization methods trained on different selection methods. The next set of experiments examines the proposed MQAG system (Section 7.2.2). Further, to exploit large language models for zero-shot NLG assessment (Section 7.3), the experiments compare comparative assessment to standard prompt engineering.

7.4.1 Datasets

A summary assessment dataset generally consists of a set of source documents X, system-generated summaries Y, and human-evaluated scores z,

$$\text{Dataset} = \left\{ X^{(j)}, \left(Y_i^{(j)}, z_i^{(j)} \right)_{i=1,\ldots,N} \right\}^{j=1,\ldots,M} \tag{7.26}$$

where in this example, there are N summarization systems and M source documents.[4] To evaluate automatic assessment methods on a dataset (in the format in Equation 7.26), correlations between predicted scores by automatic methods and human-evaluated scores are used. These correlations can be system-level or summary-level correlations, which are described in Section 4.3.3.

In addition to using existing datasets (which are predominantly based on news summarization datasets), this work creates a new summary assessment dataset, which we call **Podcast Summary Assessment (PSA)**. The podcast data was previously used in Chapter 6 for developing summarization systems, and a portion of the podcast data was annotated for the Spotify Summarization Challenge. We compile the annotated portion to make the PSA dataset. This new dataset has two unique aspects: (1) long-input, speech podcast-based, documents; and (2) an opportunity to detect inappropriate reference summaries in podcast corpus. Table 7.2 provides an overview of the datasets that will be used in our experiments, and their descriptions are provided below.

QAG-CNNDM & QAG-XSum

Wang et al. [297] annotated 235 CNN/DailyMail summaries of the system in Gehrmann et al. [82] and 239 XSum summaries of fine-tuned BART [159]. The annotation was performed at the sentence level to indicate if hallucination occurs or not. Subsequently, for each summary, the faithfulness (consistency) score is obtained by averaging the sentence-level human scores.

[4] Note that SummEval has multiple aspects of human-evaluated scores.

Corpus	Type	Size	Length Src.	Length Sum.	Annotation
QAGS-CNNDM	News	235	355.8	54.4	Faithfulness
QAGS-XSum	News	239	403.7	19.7	Faithfulness
XSum-H	News	†5×500	442.1	20.5	Faithfulness, Factuality
SummEval	News	†16×100	404.0	63.7	Coherence, Faithfulness, Fluency, Relevance
Podcast Summary Assessment (§A)	Podcast	†20×179	5950	88.3	4-point scale Overall (Informative & Fluency) and 8 binary attributes (e.g. names, topic, etc.)

Table 7.2 Summary of datasets annotated for summary assessment. Length statistics is the number of words calculated using the NLTK tokenizer. †#systems (N) × #documents (M).

XSum-Hallucination (XSum-H)

Maynez et al. [193] annotated 2500 XSum summaries using 3 crowd-sourced workers on two aspects: (1) Faithfulness = whether the information is faithful with respect to the source document at the token level. The judgements are then averaged; (2) Factuality = whether the summary level is factual with respect to the source document as well as world knowledge.

SummEval

Proposed by Fabbri et al. [67], this dataset consists of human judgements of 16 summarization systems on 100 news articles randomly selected from the test set of the CNN/DailyMail dataset [104, 208]. Each of the summaries was annotated on four aspects: (1) *relevancy* (how well the summary selects important content from the source, and summaries with redundancies should be penalized), (2) *consistency* (how factual the summary is with respect to the source document, and summaries that contain hallucinated facts should be penalized), (3) *coherency* (how well all sentences are connected, and summaries should not just be a heap of related information), and (4) textitfluency (the quality of individual sentences such as no formatting problems, capitalization errors, or ungrammatical sentences). Each summary was graded on a 5-point scale by three expert annotators for each of the dimensions.[5].

Podcast Summary Assessment

Based on the Spotify Podcast Challenge at TREC 2020 [125], we compiled podcast summaries from 19 summarization systems and the creator descriptions of 179 podcasts, making the corpus of 3580 document-summary pairs. The human evaluation was performed on

[5]Note that REALSumm [13] (although not used in this thesis) is a similar corpus based on 100 CNN/DailyMail documents and 25 summarization systems. Their 2500 summaries were evaluated using the Lightweight Pyramid method [269]

a holistic 4-point scale considering a combination of consistency, coverage, and fluency. Compared to existing summary assessment data which are predominantly from news summarization, this dataset has unique aspects in its long-input (e.g., the input documents are more than 10 times longer than CNN/DailyMail articles on average) and speech-based documents. The anonymized version of this corpus has been made available under the CC-BY-4.0 license at https://github.com/potsawee/podcast_summary_assessment. More information about this dataset and the benchmark of summary assessment methods are provided in Appendix A.

7.4.2 MQAG

First, preliminary experiments are conducted to ensure that MQAG is reasonable. Then, different configurations of MQAG are investigated, including the analysis of statistical distances, variants of MQAG, and answerability. Two MQAG variants are built: $\text{MQAG}_{\text{SQuAD}}$ and $\text{MQAG}_{\text{RACE}}$, which differ in the training data of the question+answer generator `G1`, while the distractor generator `G2` and answering system `A` are both trained on RACE. As shown in Table 7.1, $\text{MQAG}_{\text{RACE}}$ is expected to yield more abstractive QA pairs than $\text{MQAG}_{\text{SQuAD}}$.

MQAG: Statistical Distances

As discussed in Section 7.2.2, we compare statistical distances for MQAG with the generator trained on SQuAD and MQAG with the generator trained on RACE. The results for different distances are shown in Table 7.3. It can be seen that in both configurations, KL-divergence yields lower correlations than other distances, and on average total variation slightly outperforms the one-best distance and achieves similar performance to the Hellinger distance. This is likely because total variation and Hellinger distances are continuous and bounded unlike KL-divergence or one-best distances. Note that total variation is selected as the main distance for subsequent experiments. Another observation is that $\text{MQAG}_{\text{SQuAD}}$, despite generating more extractive questions, achieves higher correlations than $\text{MQAG}_{\text{RACE}}$ on most tasks except on Podcast and SummEval.

Distance	QAG-		XSum-H		PSA	SumE
	CNN	XSum	Faith	Fact		
Config: MQAG-Sum, G1 = SQuAD						
KL-Div ($\mathcal{D}_{\mathtt{KL}}$)	0.478	0.374	0.177	0.226	0.251	0.936
One-Best ($\mathcal{D}_{\mathtt{OB}}$)	0.476	0.354	0.295	0.254	0.677	0.872
Total Var ($\mathcal{D}_{\mathtt{TV}}$)	0.508	0.396	0.269	0.267	0.225	0.870
Hellinger ($\mathcal{D}_{\mathtt{HL}}$)	0.499	0.399	0.266	0.269	0.201	0.870
Config: MQAG-Sum, G1 = RACE						
KL-Div ($\mathcal{D}_{\mathtt{KL}}$)	0.450	0.283	0.135	0.179	0.789	0.954
One-Best ($\mathcal{D}_{\mathtt{OB}}$)	0.453	0.225	0.240	0.221	0.839	0.928
Total Var ($\mathcal{D}_{\mathtt{TV}}$)	0.462	0.309	0.221	0.244	0.770	0.933
Hellinger ($\mathcal{D}_{\mathtt{HL}}$)	0.473	0.323	0.215	0.244	0.751	0.927

Table 7.3 Comparison of Statistical Distances using MQAG-Sum without answerability.

MQAG-Sum, MQAG-Src, MQAG-F1

Motivated by span-based methods that can be categorized into: precision-oriented, recall-oriented, or both (in Section 7.2.1), this experiment examines three variants of MQAG scores: MQAG-Sum (precision), MQAG-Src (recall), MQAG-F1 (both). Our results in Table 7.4 show that MQAG-Src, which assesses how much source information is contained in the summary by generating questions from the source, achieves lower PCCs than MQAG-Sum on all datasets. This finding aligns with our expectation, as the summaries were graded by humans predominantly on the consistency aspect (which MQAG-Sum was designed to measure) rather than the quantity of source information present (which MQAG-Src measures). When combining MQAG-Src and MQAG-Sum into MQAG-F1, we only observe a small gain on two test settings. Therefore, MQAG-Sum is selected as our main MQAG configuration for the remaining investigations.

	QAG		XSum-H		PSA	SumE
	CNN	XSum	Faith	Fact		
Config: G1 = SQuAD, D = Total Variation						
MQAG-Sum	0.508	0.396	0.269	0.267	0.225	0.870
MQAG-Src	0.272	0.017	0.093	0.037	0.470	0.707
MQAG-F1	0.490	0.393	0.286	0.261	0.475	0.863
Config: G1 = RACE, D = Total Variation						
MQAG-Sum	0.462	0.309	0.221	0.244	0.770	0.933
MQAG-Src	0.233	0.143	0.069	0.087	0.144	0.588
MQAG-F1	0.468	0.301	0.217	0.252	0.731	0.866

Table 7.4 Comparison of MQAG-Src, MQAG-Sum, and MQAG-F1 without answerability.

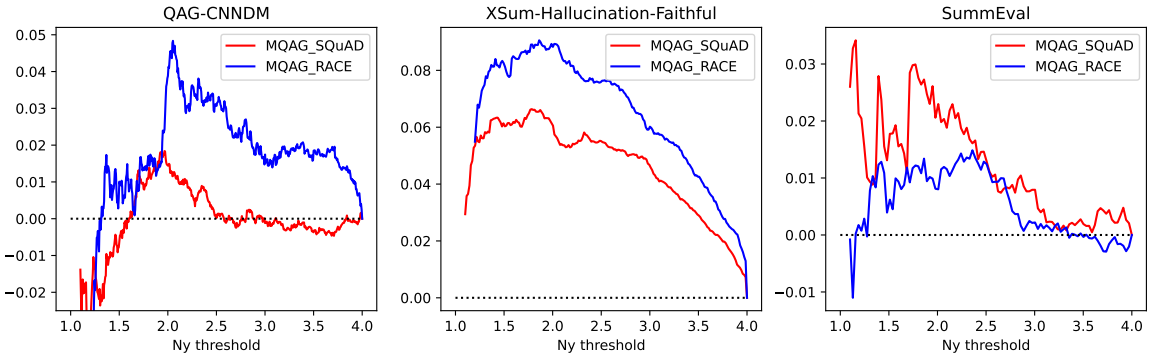
MQAG: Answerability

So far, all of the generated questions are used in computing the MQAG score. However, the question generation system could generate *unanswerable* questions. Thus, this experiment examines whether filtering out unanswerable questions could improve performance. This work proposes using a simple entropy-based score from the answering system (defined in Equation 7.17) to measure answerability. This measure is the effective number of options which is bounded between 1 and 4 as there are four options in our setup.

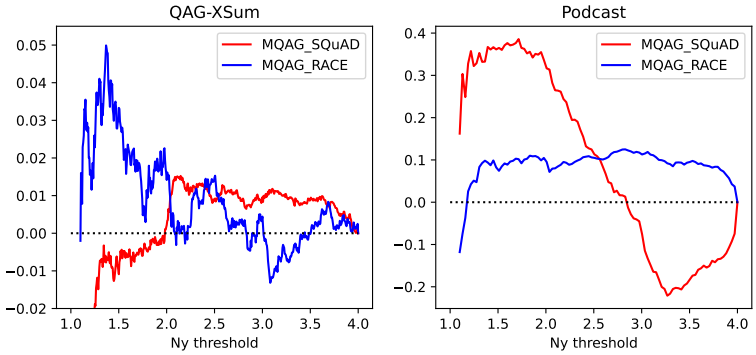
Figure 7.9 shows the results as the answerability score is swept from 4.0 (keeping all questions) to 1.0 (only keeping those that the answering system A is highly confident). It can be seen that as we filter out high-entropy questions, there is an upward trend in performance across all tasks. In addition, as shown in the figure, setting $\mathcal{N}_Y^\tau$ at 2.0 seems to be a reasonable answerability threshold. At this threshold, $\mathcal{N}_Y^\tau = 2.0$, out of 50 automatically generated questions, about 36 questions are kept for $\text{MQAG}_{\text{SQuAD}}$ and about 30 questions are kept for $\text{MQAG}_{\text{RACE}}$. The number of remaining questions is similar across all datasets as shown in Table 7.5. Thus, we set $\mathcal{N}_Y^\tau = 2.0$, and the performance of MQAG using this answerability criterion is presented and compared against baseline systems in Table 7.6.

System	QAG-CNNDM	QAG-XSum	XSum-H	PSA	SummEval
$\text{MQAG}_{\text{SQuAD}}$	35.0	37.4	34.0	34.7	37.0
$\text{MQAG}_{\text{RACE}}$	30.5	30.0	30.0	30.5	31.1

Table 7.5 The percentage of remaining questions at $\mathcal{N}_Y^\tau = 2.0$.



(a) Left: QAG-CNNDM, Center: XSum-H, Right: SummEval



(b) Left: QAG-XSum, Right: PSA

Fig. 7.9 ΔPCC of MQAG-Sum with total variation (i.e. PCC $-$ PCC$_{N_Y^\tau=4.0}$) against the answerability threshold $\mathcal{N}_Y^\tau$ on X-axis. MQAG without answerability is equivalent to setting $N_Y^\tau = 4.0$, and the results at this operating point can be seen on the right-most point in each plot. As we reduce the threshold ($N_Y^\tau \downarrow$), more questions are rejected.

Comparison of MQAG Against Baselines

The previous experiments investigated different configurations of MQAG, and the findings suggested an optimal setup as follows. (1) statistical distance is total variation (2) the variant is MQAG-sum, where generation stage G generates questions from summary y (3) the answerability threshold $\mathcal{N}_Y^\tau$ is set to 2.0. This experiment compares MQAG, based on the optimal setup, against existing baselines. The experimental results are shown in Table 7.6. The observation is that MQAG achieves a higher correlation than the best span-based method on 5 out of 6 tasks. When compared to all existing baselines, MQAG achieves state-of-the-art performance on 4 out of 6 tasks. To investigate the impact of the abstractiveness of summaries on the evaluation performance, we split QAG-XSum and XSum-H datasets[6] into two portions of the same size by abstractiveness as measured by the longest sequence in the

[6]XSum summaries are more abstractive than CNNDM summaries, so using XSum should enable us to investigate the impact of abstractiveness better than CNNDM.

summary that exists in the source per the length of the summary (i.e., ROUGE-L precision of summary y using source x as the reference). The results in Table 7.8 show that although $\text{MQAG}_{\text{RACE}}$ achieves lower PCCs than $\text{MQAG}_{\text{SQuAD}}$ (in Table 7.6) when evaluated on the more abstractive split, the performance $\text{MQAG}_{\text{RACE}}$ is much closer to that of $\text{MQAG}_{\text{SQuAD}}$. In addition, compared to MQAG, span-based methods show a larger drop in PCCs in the more abstractive split. This finding further illustrates the benefits of comparing answer distributions rather than text spans.

Method	QAG		XSum-H		PSA	SumEvl
	CNNDM	XSum	Faithful	Factual		
Baselines: Other Approaches						
ROUGE-1	0.337	0.012	-0.050	0.008	0.326	0.458
OpenIE-TripleMatching	0.381	0.131	0.019	-0.020	0.706	0.548
BERTScore	0.584	0.008	0.185	0.154	0.718	0.645
Entailment (BERT Model)	0.159	0.169	0.362	0.209	0.228	0.619
Baselines: SpanQAG						
QAGS	0.437	0.200	0.101	0.080	0.464	0.812
FEQA	0.322	0.283	0.297	0.171	0.603	0.464
QuestEval	0.250	0.173	0.421	0.197	0.579	0.838
Multiple-choice Question Answering and Generation (MQAG)						
$\text{MQAG}_{\text{SQuAD}}$	0.519	0.407	0.324	0.292	0.502	0.890
$\text{MQAG}_{\text{RACE}}$	0.502	0.313	0.306	0.270	0.855	0.945

Table 7.6 Pearson Correlation Coefficient (PCC) between the scores of evaluation methods and human judgements. PCCs are computed at the summary level on QAG and XSum-H, and at the system level on Podcast and SummEval. PCCs on Podcast are computed on 15 abstractive systems. Underline denotes where MQAG outperforms the best SpanQAG system, which is 5 out of 6 tasks. When compared to all baselines, MQAG achieves the highest PCC on 4 out of 6 tasks.

Method	QAG		XSum-H		PSA	SumEvl
	CNNDM	XSum	Faithful	Factual		
Baselines: Other Approaches						
ROUGE-1	0.318	0.053	-0.030	0.001	0.282	0.627
OpenIE-TripleMatching	0.337	0.130	0.019	-0.025	0.700	0.671
BERTScore	0.523	0.018	0.183	0.153	0.686	0.835
Entailment (BERT Model)	0.167	0.190	0.380	0.202	0.207	0.141
Baselines: SpanQAG						
QAGS	0.341	0.166	0.085	0.052	0.357	0.421
FEQA	0.275	0.277	0.300	0.155	0.504	0.270
QuestEval	0.181	0.175	0.415	0.176	0.425	0.812
Multiple-choice Question Answering and Generation (MQAG)						
$\text{MQAG}_{\text{SQuAD}}$	0.470	0.409	0.335	0.284	0.441	0.773
$\text{MQAG}_{\text{RACE}}$	0.460	0.308	0.322	0.266	0.779	0.920

Table 7.7 Spearman's rank correlation coefficient – complementing Table 7.6.

Method	QAG-XSum		XSum-H	
	Low	High	Low	High
QAGS	0.190	0.184	0.101	0.159
FEQA	0.296	0.163	0.290	0.124
QuestEval	0.215	0.061	0.398	0.326
$\text{MQAG}_{\text{SQuAD}}$	0.431	0.328	0.334	0.254
$\text{MQAG}_{\text{RACE}}$	0.277	0.295	0.319	0.249

Table 7.8 Performance as measured by PCC on the *low* abstractiveness and *high* abstractiveness of QAG-XSum and XSum-H (Faithful). The results on the entire datasets are in Table 7.6.

As MQAG is based on the Monte-Carlo approximation (Equation 7.9), sampling questions, answers, and distractors are stochastic and sampling N could potentially be computationally expensive. Thus, we perform an ablation study about the impact of the number of samples on the performance and variance. Second, although the previous section investigated training data, e.g. RACE versus SQuAD, another aspect of MQAG is the choice of model backbones. To ablate this, we both fine-tune different pre-trained backbones to MQAG as well as use zero-shot GPT-3 prompting.

Ablation 1: Impact of the Number of Questions (N)

As MQAG is a sampling-based approach, one of the concerns is the computational cost. This experiment analyses the impact of the number of generated questions on the performance of MQAG. The mean and standard deviation are presented in Figure 7.10 as N is varied from 1 to 50. The results show a smooth increase in correlation, which is expected because the framework is based on a Monte-Carlo approximation, and a similar finding was also observed in the results of QAGS [297].

Figure 7.10 also shows that the variance decreases with N. This shows the stability of MQAG. Although the performance curve has not completely plateaued at $N = 50$, since the computational cost of MQAG scales linearly with N, sampling 50 questions seems to be a reasonable compromise between computational efficiency and performance. It should be noted that the current approach does not explicitly measure the diversity of the generated questions. It would be possible to measure diversity, for example, by clustering generated questions. However, this work demonstrates empirically that diversity can be achieved through the exhaustive Monte-Carlo sampling, as the performance increases. This is because the performance is expected to remain similar if additional questions are not different from the ones already generated. An interesting next step would be to investigate if the same or similar performance can be achieved with as low N as possible, for example, by generating a smaller but more diverse set of questions and options such as varifocal question generation where questions are generated based on different focal points [217].

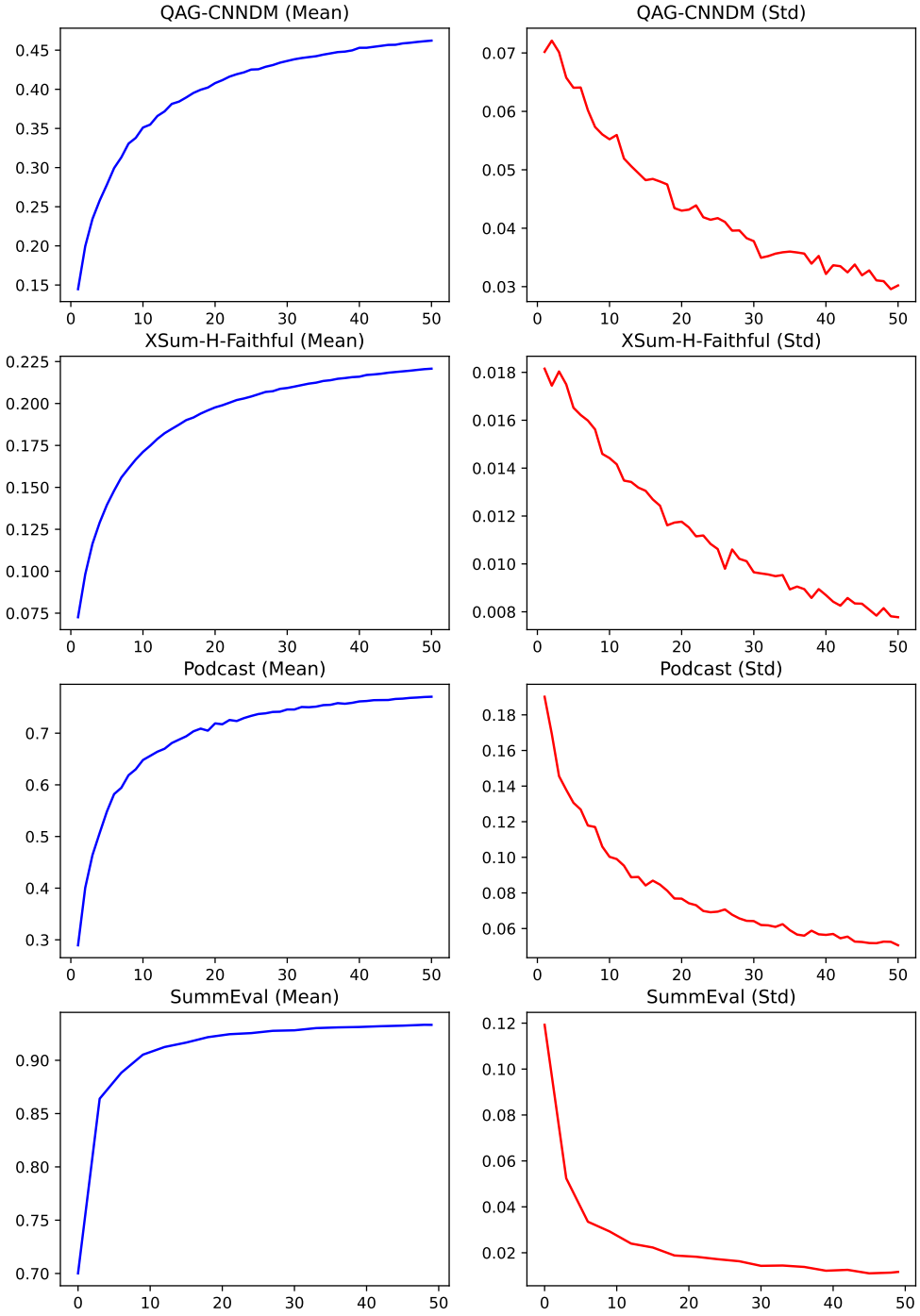


Fig. 7.10 Mean and standard deviation of Pearson correlation (Y-axis) of $\text{MQAG}_{\text{RACE}}$ on QAG-CNNDM when the number of generated questions N is varied from 1 to 50 (X-axis). Standard deviation is obtained via bootstrapping.

Ablation 2: Model Choices

So far, the experiments have only used the same backbones: T5-large for question generation and Longformer for question answering. This ablation investigates the significance of *pre-trained backbones* by swapping to less capable models, e.g., T5-large → T5-base for generation, and Longformer(4096) → RoBERTa(512) [170] for answering. The results in Table 7.9 show that: (1) For the generation stage, using a smaller model does not result in lower performance. This could be because T5-base has higher perplexity, and yields more diverse questions. (2) In contrast, for the answering stage, when using RoBERTa, with a shorter input length, the performance on SummEval (the input length is mostly shorter than 512) remains almost the same. However, as the input length is longer in other datasets, we observe a drop in PCC when using RoBERTa.

Model		Pearson Corr.		
Generation	Answering	SummEval	QAG-XSum	PSA
T5-base	RoBERTa	0.949	0.242	0.471
T5-base	Longformer	0.949	0.293	0.647
T5-large	RoBERTa	0.930	0.211	0.350
T5-large	Longformer	0.930	0.229	0.772

Table 7.9 Ablation on model choices in MQAG using N=20. SumE = SummEval (Consistency aspect), QAG-X = QAG-XSum, Podc = Podcast Assessment.

Additionally, given the impressive results of large language models (LLMs) across natural language generation tasks, we investigate the performance of LLMs in a zero-shot fashion instead of using fine-tuned T5 for multiple-choice question generation. Specifically, we use OpenAI GPT-3 [15] (text-davinci-003) where we query 50 questions and 4 options using the following prompt format:

```
Write 50 diverse multiple-choice questions with 4 options from the
following context: {context}
```

We found that GPT-3 generated 50 questions as specified in the prompt around 26% of the examples and the remaining only have 20 questions. The majority of questions (more than 95%) have 4 options, while the remaining have 2 options. In Table 7.10, the results show that zero-shot GPT-3 performs worse than our fine-tuned T5 systems in both multiple-choice question generation tasks. This illustrates that there is some sensitivity due to the quality of generated questions, and using our fine-tuned T5 is a better option than zero-shot GPT-3.

Backbone	QAG	
	CNNDM	XSum
T5 (SQuAD)	0.508	0.396
T5 (RACE)	0.462	0.309
GPT-3	0.392	0.130

Table 7.10 Pearson Coefficient Correlation (PCC). GPT-3 versus fine-tuned T5 using $\mathcal{D}_{\text{TV}}$ without answerability for the multiple-choice question generation stage.

7.4.3 Zero-shot Methods

The previous set of experiments assessed summaries on the *consistency* aspect using our proposed MQAG. In this experiment, the focus is on the zero-shot performance of large language models (LLMs) that could assess summaries on *any* aspect, including consistency. Two LLM prompting techniques are investigated: standard LLM prompt scoring (Section 7.3.1) and the proposed LLM comparative assessment (Section 7.3.2).

Experimental Setup

We investigate two families of open-source instruction-tuned LLMs. The first system is the encoder-decoder FlanT5 [32] or T5 [244] that have been instruction tuned on a diverse set of 1000 NLP tasks [305]. The second system is the decoder-only Llama2-chat [286], which are Llama2 systems tuned on publicly available instruction datasets. We investigate a range of model sizes; 220M, 770M, 3B and 11B for FlanT5, and 3B and 13B for Llama2.

Each LLM is used in both setups: prompt-scoring and comparative assessment. For the comparative assessment results, we consider the full set of possible comparisons, where all pairs of candidates in both permutations are compared by the framework, e.g., $N \times (N-1)$ comparisons where N is the number of candidates. Comparisons are made using the prompt-based classifier (as described in Section 7.3.2) using the prompt templates shown in Figure 7.8, where the system outputs a probability for Response A and Response B. The winner of the comparison is the response with the highest probability, where candidates are then ranked in order of their win-ratio (as described in Section 7.3.2). For the Llama2 systems, comparative prompts are appended with '`Answer:`' while scoring prompts end with '`Score:`'.

Experimental Results

The experiments are conducted on SummEval and Podcast Summary Assessment because these two datasets have multiple summaries from different systems for each document, allowing pairwise comparison. Regarding evaluation, summary-level ranking is also considered as

our MQAG method has already achieved high system-level correlations. Note that although the following section provides the results of summary assessment tasks, the investigation can conducted on other natural language generation (NLG) tasks such as dialogue generation or data-to-text generation. This is because the zero-shot framework is flexible, and not limited to only assessing summarization. The results on dialogue generation or data-to-text generation, though are not in the scope, can be found in our paper [173].

SummEval: Table 7.11 analyzes the effectiveness of comparative assessment on SummEval, where the following observations can be made:

Approach	COH		CON		FLU		REL	
	SCC	PCC	SCC	PCC	SCC	PCC	SCC	PCC
Baselines (reproduction)								
BERTScore (w/ Doc)	0.410	0.417	0.327	0.340	0.265	0.284	0.403	0.415
QuestEval	0.182	0.178	0.306	0.377	0.228	0.281	0.268	0.296
$MQAG_{RACE}$	0.170	0.188	0.288	0.355	0.193	0.223	0.166	0.213
Baselines (reported in existing work)								
UniEval (single-best) [354]	0.546	-	0.472	-	0.433	-	0.463	-
UniEval (continual) [354]	0.575	-	0.446	-	0.449	-	0.426	-
GPTScore FlanT5-3B [74]	0.470	-	0.436	-	0.421	-	0.344	-
GPTScore FlanT5-11B [74]	0.456	-	0.438	-	0.424	-	0.343	-
GPTScore GPT3 [74]	0.401	-	0.475	-	0.410	-	0.343	-
ChatGPT scoring [301]	0.451	0.456	0.432	0.512	0.380	0.443	0.439	0.473
Prompt Scoring (§7.3.1)								
FlanT5-220M	0.040	-0.015	-0.002	-0.060	0.002	-0.039	0.028	-0.051
FlanT5-770M	-0.036	-0.047	-0.016	-0.008	-0.015	-0.002	0.000	0.019
FlanT5-3B	0.145	0.120	0.198	0.151	0.039	-0.013	0.152	0.132
FlanT5-11B	0.007	-0.012	0.112	0.078	0.032	0.027	0.057	0.052
Llama2-chat-7B	0.086	0.086	0.090	0.102	0.018	0.014	0.078	0.089
Llama2-chat-13B	0.099	0.029	0.069	0.022	0.012	0.015	0.092	0.032
Comparative Assessment (§7.3.2)								
FlanT5-220M	0.040	0.047	-0.002	0.015	0.002	0.010	0.028	0.039
FlanT5-770M	0.298	0.318	0.263	0.317	0.206	0.233	0.351	0.370
FlanT5-3B	0.512	0.525	0.471	0.477	0.325	0.368	0.448	0.468
FlanT5-11B	0.442	0.464	0.372	0.426	0.302	0.349	0.434	0.476
Llama2-chat-7B	0.279	0.286	0.246	0.241	0.202	0.189	0.356	0.367
Llama2-chat-13B	0.409	0.416	0.399	0.445	0.308	0.345	0.453	0.492

Table 7.11 SummEval results (averaged over both prompts per system for prompt-scoring and comparative assessment) measured by summary-level SCC and PCC. COH = Coherency, CON = Information Consistency, FLU = Fluency, REL = Relevance.

(1) Moderate-sized LLMs are ineffective in the prompt-scoring set-up, with the best system (FlanT5-3B) achieving Spearman correlations of only between 0.10 and 0.20. The performance difference with ChatGPT prompt-scoring implies that scoring is likely an emergent ability only effective for larger LLMs.

(2) LLMs are able to achieve considerably higher correlations in the comparative assessment set-up, with performance higher for nearly all systems. Furthermore, comparative assessment leads to more robust performance, with most models with more than 3B parameters achieving Spearman correlations within the range of 0.30 and 0.50.

(3) Comparative assessment enables LLMs of under 1B parameters to perform well, with FlanT5-770M achieving moderate correlations. However, performance improves significantly when using LLMs with more than 3B parameters, although for SummEval there are diminishing (if any) performance gains by scaling up the LLM.

(4) The best comparative assessment LLM (FlanT5-3B) is competitive with all other zero-shot methods, including ChatGPT scoring (which is an LLM with two orders of magnitude more parameters), and achieves the best correlation in 3 of the 4 aspects considered.

(5) Comparative assessment achieves competitive performance with UniEval. Although UniEval has better overall performance, UniEval was designed for bespoke tasks and aspects (it is fine-tuned on synthetic data created for particular attributes) where the results in Table 7.12 show that UniEval has noticeable degradation in out-of-domain settings.[7] In contrast, our proposed comparative assessment is zero-shot and more general than UniEval.

[7]The results on data-to-text assessment (WebNLG) in Liusie et al. [173] also shows noticeable degradation.

Approach	System-level		Summary-level	
	SCC	PCC	SCC	PCC
Baselines (reproduction)				
BERTScore (w/ Ref)	0.739	0.793	0.251	0.264
BERTScore (w/ Doc)	0.686	0.718	0.219	0.225
UniEval (continual)	0.420	0.542	0.228	0.231
QuestEval	0.425	0.579	0.204	0.218
$\text{MQAG}_{\text{RACE}}$	0.779	0.855	0.126	0.138
Longformer-SFT*	0.896	0.926	0.196	0.203
Prompt Scoring (§7.3.1)				
Llama2-chat-7B	0.885	0.892	0.026	0.023
Llama2-chat-13B	0.800	0.918	0.253	0.259
Comparative Assessment (§7.3.2)				
Llama2-chat-7B	0.882	0.941	0.374	0.387
Llama2-chat-13B	0.971	0.986	0.455	0.474

Table 7.12 Podcast Summary Assessment results measured by SCC and PCC. *Longformer with Supervised Fine-tuning on the assessment task using 5-fold cross-validation.

Podcast Assessment: When considering podcast summarization (which has long inputs of over 5,000 tokens on average), only Llama2 models (which have a limit of 4,096 tokens) were used (as FlanT5 has a limit of 1,024 tokens). Table 7.12 shows that comparative assessment yields highly impressive performance for long-input podcast summarization, with comparative assessment out-competing all other baselines. Furthermore, although prompt-scoring has good system-level correlations, the lack of granularity leads to poor summary-level performance.

In conclusion, the zero-shot method with comparative assessment has demonstrated the flexibility of the framework in that it can be used to assess any aspect as shown on SummEval. In addition, it is effective in long-input summarization such as Podcast summarization. However, we note that the limitations of the approach could be due to the number of comparisons required, which grows quadratically with the number of candidates, coupled with running LLM inference can lead to large computational expenses.

7.5 Chapter Summary

This chapter investigated automatic summary assessment. As modern summarization systems have become highly fluent, the focus has been on assessing information consistency between the source and the summary. We propose MQAG – a novel scheme for assessing information consistency between source and summary based on the distance between multiple-choice answer distributions instead of text-based answer spans in existing question-answering methods. Our experiments demonstrated the potential of this alternative approach which outperforms existing techniques on various datasets. The realization of the framework exploits current multiple-choice question generation and answering systems. Its performance is expected to increase as backbone systems improve, for example, the diversity of questions generated and the selection of options. Also, the framework is more interpretable than black-box neural networks as generated questions and answers can be examined, allowing more insight into summary assessment.

Furthermore, following the success of zero-shot LLM in various NLP tasks, we examine LLM on summary assessment and propose the comparative assessment framework. Our experimental results demonstrated that comparative assessment with a moderate-size LLM outperforms vanilla prompt-scoring. LLM comparative assessment is flexible and effective, achieving comparable to approaches designed specifically for assessing NLG tasks.

Chapter 8

Generative AI Information Consistency

Chapter 7 investigated summary assessment, where one key aspect is assessing information consistency between the source document and the summary. Motivated by summary assessment methods, this chapter will extend the information consistency assessment methods to broader generative AI information consistency assessment with a focus on assessing long-form large language model (LLM) generation.

Despite the advances in scaling up foundation models and self-supervised learning, generative AI models may make up facts or misleading information in their responses with respect to actual knowledge, and this phenomenon is referred to as *hallucination* in the field of AI [123].[1] First, Section 8.1 provides an introduction and existing work about hallucination detection. Section 8.2 proposes SelfCheckGPT, which is a family of methods inspired by summary assessment for hallucination detection in generative large language models. Section 8.3 discusses data collection and annotation required for investigating language model hallucination. Section 8.4 provides experiments and discusses the results. Section 8.5 discusses the limitations and potential improvements to SelfCheckGPT.

8.1 Generative AI Hallucination

Generative AI models are being used in a range of domains such as text generation, image generation, and audio generation [14, 16]. These models are capable of high-quality outputs such as fluent and coherent texts and realistic images. Generative AI models are trained on a large amount of data, which enables them to handle users' queries in a zero-shot manner [32, 218, 341]. The models might be tasked with queries beyond the knowledge obtained

[1] A *hallucination* is an incorrect prediction made by an AI model. In the context of language models, a hallucination is a response which contains non-factual information with respect to actual knowledge. This corresponds to non-factuality in summarization defined by Maynez et al. [193] in the previous chapter.

from the training data, therefore, they must perform extrapolation, which could lead to *novel* responses. In some cases such as image generation, novel responses could be considered creative. However, in other cases such as text generation, novel responses could mean *unfaithful* as defined in Section 4.3. This is similar to inconsistency in summarization where the summary is not grounded by the source document or world knowledge investigated in Chapter 7. For example, language models could mix information from different sources and make a wrong inference, or they could make up facts that are not grounded by any sources. These problems are generally referred to as *hallucinations*. Therefore, it is important to understand the limitations of generative AI models and develop a mechanism to prevent them from returning such hallucinations.

8.1.1 Language Model Hallucination

Large language models like GPT-3 [15] and PaLM [31] are effective in generating realistic text for various applications but they have a tendency to hallucinate, underscoring the importance of evaluating their capabilities and limitations. The NLP community has developed several benchmarks to evaluate LLMs, such as SuperGLUE [298], BIG-Bench [275], MMLU [103], and HELM [161], covering a wide range of tasks. Yet, the specific challenge of hallucination in LLM output remains insufficiently addressed in these evaluations.

Hallucination has been studied in natural language generation (NLG) tasks, including summarization [119], machine translation [97, 356], dialogue generation [272], as well as other NLG tasks surveyed in Ji et al. [123]. However, it is less investigated in open-domain language model generation. One reason is that hallucinated contents are specific to the language model generating the responses, and therefore, a specifically crafted dataset is required to investigate each individual language model, which makes hallucination detection more challenging to standardize. Also, existing hallucination detection resources, such as HADES [166], are obtained by perturbing factual texts and thus may not reflect true open-domain LLM hallucination.

A possible approach to hallucination detection is to leverage existing intrinsic uncertainty metrics to determine the parts of the output sequence that the system is least certain of [337, 74]. However, uncertainty metrics such as token probability or entropy require access to token-level probability distributions, information which may not be available to users for example when systems are accessed through limited external APIs. An alternate approach is to leverage fact-verification approaches, where evidence is retrieved from an external database to assess the veracity of a claim [284, 99]. However, facts can only be assessed relative to the knowledge present in the database. Additionally, hallucinations are observed over a wide range of tasks beyond pure fact verification [143, 193].

Recently, Azaria and Mitchell [1] trained a multi-layer perception classifier where an LLM's hidden representations are used as inputs to predict the truthfulness of a sentence. However, this approach is a white-box approach that uses the internal states of the LLM, which may not be available and requires labelled data for supervised training. Another recent approach is self-evaluation [126], where an LLM is prompted to evaluate its previous prediction, e.g., to predict the probability that its generated response is true.

Sequence Level Uncertainty Estimation

Token probabilities have been used as an indication of model certainty. For example, OpenAI's GPT-3 web interface allows users to display token probabilities (as shown in Figure 8.1), and further uncertainty estimation approaches based on aleatoric and epistemic uncertainty have been studied for autoregressive generation [325, 181]. Additionally, conditional language model scores have been used to evaluate properties of texts [337, 74].

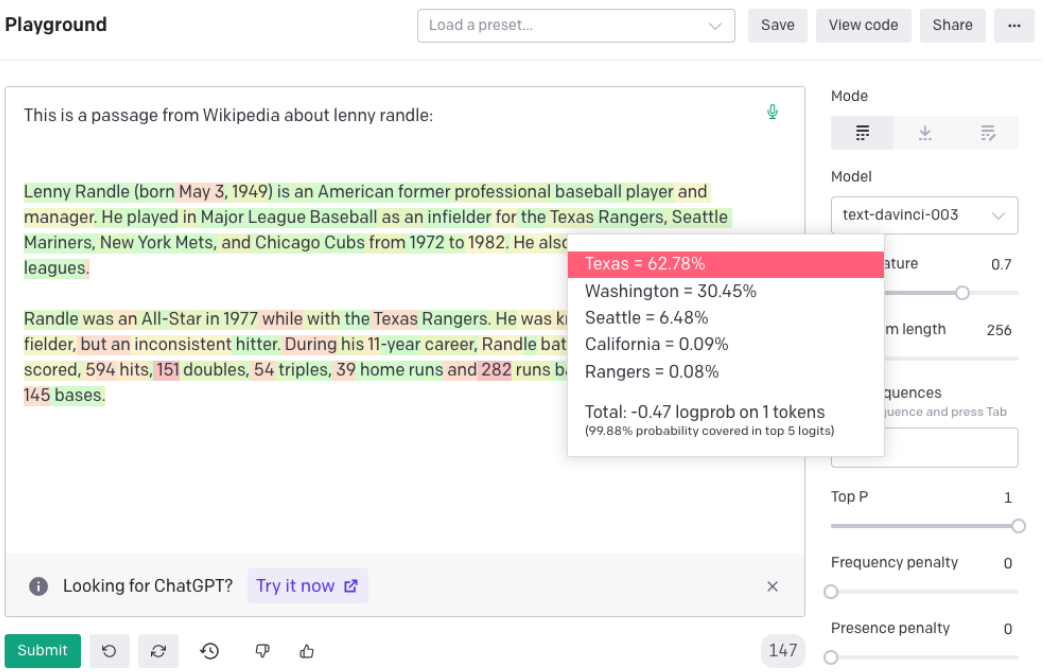


Fig. 8.1 Example of OpenAI's GPT-3 web interface with output token-level probabilities displayed.

Predictive entropy (in Equation 8.1) has been used to measure the model's uncertainty in classification tasks [76, 180, 182]. The predictive entropy of a random variable X which takes a value x from a set of possible values $\mathcal{X}$ is:

$$\mathcal{H}(X) = -\sum_{x \in \mathcal{X}} P(x) \log P(x) \tag{8.1}$$

Recently, semantic uncertainty has extended predictive uncertainty to natural language generation (NLG) [145]. As there can be multiple ways of generating the same concept in NLG, semantic uncertainty aggregates the likelihood of the outputs with the same meaning. It approximates the set of all possible outputs using Monte-Carlo samples. The authors demonstrated the effectiveness of semantic uncertainty on question-answering. For example, if predicted sequences of `What's is capital of France?` are [`Paris`, `It's Paris`, `Rome`] with probability [0.5, 0.4, 0.1]. These outputs correspond to two groups with probability [0.9, 0.1], and the predictive entropy is computed as in Equation 8.1.

Fact Verification

Existing fact-verification approaches follow a multi-stage pipeline of claim detection, evidence retrieval and verdict prediction [99, 355]. Such methods, however, require access to external databases and can have considerable inference costs.

8.1.2 Grey-Box Factuality Assessment

This section will introduce methods that can be used to determine the factuality of LLM responses in a zero-resource setting when one has full access to output distributions.[2] This thesis will use 'factual' to define when statements are grounded in valid information, i.e. when hallucinations are avoided, and 'zero-resource' when no external database is used.

Uncertainty-based Assessment

To understand how the factuality of a generated response can be determined in a zero-resource setting, we consider LLM pre-training. During pre-training, the model is trained with next-word prediction over massive corpora of textual data. This gives the model a strong understanding of language [122, 244], powerful contextual reasoning [346], as well as world knowledge [175]. Consider the input "`Lionel Messi is a _`". Since Messi is a world-famous athlete who may have appeared multiple times in pre-training, the LLM is likely to know who Messi is. Therefore given the context, the token "`footballer`" may be assigned a high probability while other professions such as "`carpenter`" may be considered improbable. However, for a different input such as "`John Smith is a _`", the system will be unsure of the continuation which may result in a flat probability distribution. During inference, this is likely to lead to a non-factual word being generated.

[2] Alternate white-box approaches such as that of Azaria and Mitchell [1] require access to full internal states, and are less practical and so not considered in this work.

This insight allows us to understand the connection between uncertainty metrics and factuality. Factual sentences are likely to contain tokens with higher likelihood and lower entropy, while hallucinations are likely to come from positions with flat probability distributions with high uncertainty.

Token-level Probability

Notation:

- $\boldsymbol{\theta}$ is a (large) language model
- X is the LLM's response (to be assessed)
- x_i is the i-th sentence
- $x_{i,j}$ is the j-th token in x_i, and the number of tokens in x_i is J
- $p_{i,j} = P(x_{i,j}|x_{1,1},...,x_{i,j-1};\boldsymbol{\theta})$ is the probability of the token generated by the LLM at the j-th token of the i-th sentence.

For i-th sentence, two probability metrics are defined as follows:

$$\text{Avg}(-\log p)_i = -\frac{1}{J}\sum_j \log p_{i,j} \tag{8.2}$$

$$\text{Max}(-\log p)_i = \max_j \left(-\log p_{i,j}\right) \tag{8.3}$$

where $\text{Avg}(-\log p)_i$ is the average of token-level probabilities, and $\text{Max}(-\log p)_i$ measures the sentence's likelihood by assessing the *least* likely token in the sentence.

Entropy

Entropy measures can utilize more information in the output distribution. The entropy of the output distribution at the token level is defined as follows:

$$\mathcal{H}_{i,j} = -\sum_{w\in\mathcal{W}} P(w|x_{1,1},...,x_{i,j-1};\boldsymbol{\theta}) \log P(w|x_{1,1},...,x_{i,j-1};\boldsymbol{\theta}) \tag{8.4}$$

where $P(w|x_{1,1},...,x_{i,j-1};\boldsymbol{\theta})$ is the probability of the token w being generated at the j-th token of the i-th sentence and $\mathcal{W}$ is the set of all possible tokens in the vocabulary. Similar to the probability-based metrics, two entropy-based metrics are defined for sentence-level

measures as follows:

$$\text{Avg}(\mathcal{H})_i = \frac{1}{J}\sum_j \mathcal{H}_{i,j} \tag{8.5}$$

$$\text{Max}(\mathcal{H})_i = \max_j \left(\mathcal{H}_{i,j}\right) \tag{8.6}$$

The average token-level entropy in Equation 8.5 is a sum of conditional entropies via the entropy chain rule. This measure has been shown to be a Monte-Carlo approximation (with 1 sample) of the total uncertainty in an autoregressive prediction by Malinin and Gales [181].

8.1.3 Black-Box Factuality Assessment

A drawback of grey-box methods is that they require output token-level probabilities. Though this may seem a reasonable requirement, for massive LLMs only available through limited API calls, such token-level information may not be available (such as with ChatGPT). Therefore, this work considers black-box approaches which remain applicable even when only text-based responses are available.

Proxy LLMs

A simple approach to approximate the grey-box approaches is by using a proxy LLM $\tilde{\boldsymbol{\theta}}$, i.e. another LLM that we have full access to, such as LLaMA [286]. A proxy LLM can be used to approximate the output token-level probabilities of the black-box LLM $\boldsymbol{\theta}$ that generates the text. Hence, the measures previously defined in the grey-box section is approximated by:

$$P(.|x_{1,1},...,x_{i,j-1};\boldsymbol{\theta}) \approx P(.|x_{1,1},...,x_{i,j-1};\tilde{\boldsymbol{\theta}}) \tag{8.7}$$

As this method only requires access to the text X, the probability and entropy measures can then be computed as in Equations 8.2, 8.3, 8.5, 8.6 without the probability distribution of $\boldsymbol{\theta}$. It should be noted that the proxy LLM method is based on the assumption that the proxy LLM $\tilde{\boldsymbol{\theta}}$ has a distribution similar to the original LLM $\boldsymbol{\theta}$. If the two LLMs are different, this method is expected to perform poorly as different LLMs have different generating patterns, i.e., poor approximation in Equation 8.7.

8.2 SelfCheckGPT

SelfCheckGPT is a family of sampling-based methods that are designed to detect whether responses generated by LLMs are hallucinated or factual. The motivating idea of SelfCheck-

GPT is that when an LLM has been trained on a given concept, the sampled responses are likely to be similar and contain consistent facts. However, for hallucinated facts, stochastically sampled responses are likely to diverge and may contradict one another. By sampling multiple responses from an LLM, one can measure information consistency between the different responses and determine if statements are factual or hallucinated. The variants of SelfCheckGPT derive from different methods that can be used for measuring the information consistency between two texts. These methods (or variants of the SelfCheckGPT approach) include BERTScore (in Section 8.2.1), MQAG (in Section 8.2.2), n-gram language model (in Section 8.2.3), natural language inference (NLI) model (in Section 8.2.4), and LLM prompting (in Section 8.2.5). As SelfCheckGPT only leverages sampled responses, it has the added benefit that it can be used for black-box models, and it requires no external database. To the best of our knowledge, SelfCheckGPT is the first work to analyze model hallucination of general LLM responses, and is the first zero-resource hallucination detection solution that can be applied to black-box systems.

Notation: Following the notation used in the grey-box section,

- X is the LLM's response (to be assessed) from a given query
- x_i is the i-th sentence in X
- $x_{i,j}$ is the j-th token in x_i
- $\{S^1, S^2, ..., S^N\}$ are stochastic responses from the same LLM using the same query
- s_i^n is the i-th sentence in S^n
- $\mathcal{S}(x_i)$ is the hallucination score of x_i

SelfCheckGPT draws N stochastic LLM response responses using the same query. It then measures the consistency between the response and the stochastic responses. SelfCheckGPT is designed to predict the hallucination score of the i-th sentence, $\mathcal{S}(x_i)$, such that $\mathcal{S}(x_i) \in [0.0, 1.0]$,[3] where $\mathcal{S}(x_i) \to 0.0$ if the i-th sentence is grounded in valid information and $\mathcal{S}(x_i) \to 1.0$ if the i-th sentence is hallucinated. The following sections will describe each of the SelfCheckGPT variants.

8.2.1 SelfCheckGPT with BERTScore

Having described a general framework of SelfCheckGPT, its first variant is SelfCheckGPT with BERTScore. This method compares the response X against the samples $\{S^1, S^2, ..., S^N\}$

[3]With the exception of SelfCheckGPT with n-gram as the score of n-gram language model is not bounded.

using BERTScore [343]. Let $\mathcal{B}(.,.)$ denote the BERTScore between two sentences. This computes the average BERTScore of x_i with the most similar sentence from each drawn sample as follows:

$$\mathcal{S}_{\text{BERT}}(x_i) = 1.0 - \frac{1}{N}\sum_{n=1}^{N} \max_k \left(\mathcal{B}(x_i, s_k^n)\right) \tag{8.8}$$

This way if the information in a sentence appears in many drawn samples, one may assume that the information is factual, whereas if the statement appears in no other sample, it is likely a hallucination. In this work, RoBERTa-Large [170] is used as the backbone of BERTScore.

8.2.2 SelfCheckGPT with Question Answering

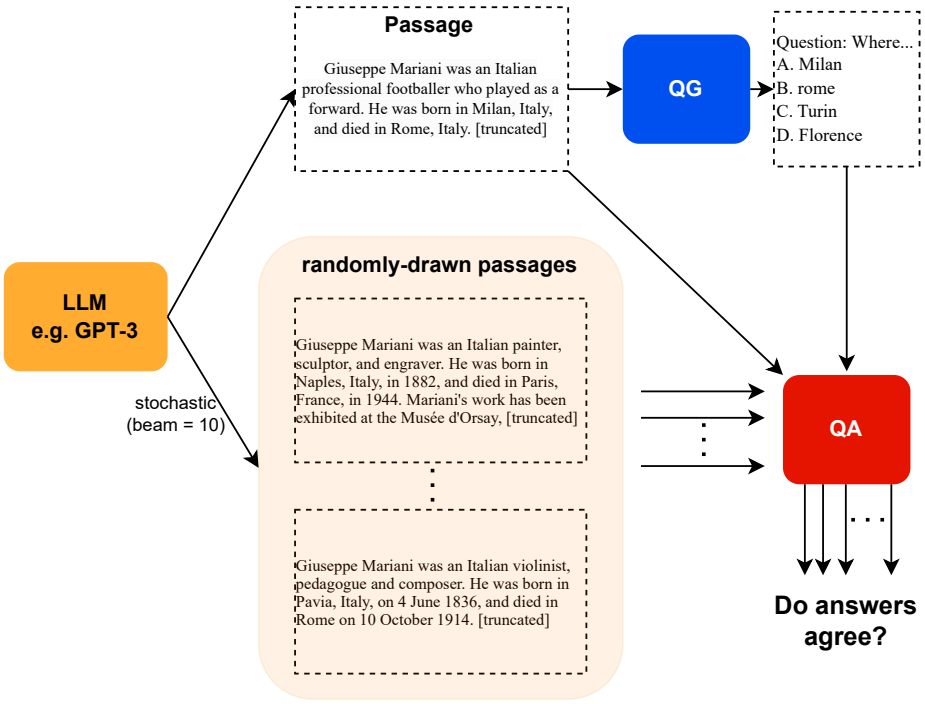


Fig. 8.2 SelfCheckGPT with MQAG.

We consider using the MQAG framework (proposed in Section 7.2.2) to measure consistency for SelfCheckGPT. MQAG assesses consistency by generating multiple-choice questions over the main generated response, which an independent answering system can attempt to answer while conditioned on the other sampled responses. First, in the generation stage $\mathtt{G}$, for the sentence x_i, we draw questions Q and options O:

$$Q, O \sim P_{\mathtt{G}}(Q, O|x_i) \tag{8.9}$$

For each drawn question Q and associated options O, the answering stage $\mathtt{A}$ selects two answers:[4] A_X is the answer grounded on the response to be assessed X and A_{S^n} is the answer grounded on the sample S^n,

$$A_X = \underset{o}{\text{argmax}} \left[P_{\mathtt{A}}(o|Q, X, O) \right] \tag{8.10}$$

$$A_{S^n} = \underset{o}{\text{argmax}} \left[P_{\mathtt{A}}(o|Q, S^n, O) \right] \tag{8.11}$$

We compare A_X is to A_{S^n} for sample in $\{S^1, ..., S^N\}$, yielding the number of matches $N_{\mathtt{m}}$ and the number of not-matches $N_{\mathtt{n}}$:

$$N_{\mathtt{m}} = \sum_{n=1}^{N} \sum_{j=1}^{J} \mathbb{I}[A_X = A_{S^n}] \tag{8.12}$$

$$N_{\mathtt{n}} = \sum_{n=1}^{N} \sum_{j=1}^{J} \mathbb{I}[A_X \neq A_{S^n}] \tag{8.13}$$

where $\mathbb{I}[.] = 1.0$ if the statement is true; otherwise $\mathbb{I}[.] = 0.0$. A simple (inconsistency) score for x_i and question Q based on the match and not-match counts is defined as:

$$\mathcal{S}_{\text{QA}}(x_i, Q) = \frac{N_{\mathtt{n}}}{N_{\mathtt{m}} + N_{\mathtt{n}}} \tag{8.14}$$

Furthermore, to take into account the answerability of generated questions, we apply the **Bayes Theorem** as follows. Let $P(\text{F})$ denote the probability of the i-th sentence being non-factual, and $P(\text{T})$ denote the probability of the i-th sentence being factual. For a question Q, the probability of i-th sentence being non-factual given a set of matched answers $L_{\mathtt{m}}$ and a set of not-matched answers $L_{\mathtt{n}}$ is:

$$\begin{aligned} P(\text{F}|L_{\mathtt{m}}, L_{\mathtt{n}}) &= \frac{P(L_{\mathtt{m}}, L_{\mathtt{n}}|\text{F})P(\text{F})}{P(L_{\mathtt{m}}, L_{\mathtt{n}}|\text{F})P(\text{F}) + P(L_{\mathtt{m}}, L_{\mathtt{n}}|\text{T})P(\text{T})} \\ &= \frac{P(L_{\mathtt{m}}, L_{\mathtt{n}}|\text{F})}{P(L_{\mathtt{m}}, L_{\mathtt{n}}|\text{F}) + P(L_{\mathtt{m}}, L_{\mathtt{n}}|\text{T})} \end{aligned} \tag{8.15}$$

[4] This is equivalent is the one-best method of MQAG defined in Equation 7.9 and Equation 7.13.

where we assume the sentence is equally likely to be False or True, i.e. $P(\mathrm{F}) = P(\mathrm{T})$. The probability of observing $L_{\mathtt{m}}, L_{\mathtt{n}}$ when the sentence is False (i.e., non-factual):

$$\begin{aligned} P(L_{\mathtt{m}}, L_{\mathtt{n}}|\mathrm{F}) &= \prod_{A \in L_{\mathtt{m}}} P(A = A_R|F) \prod_{A' \in L_{\mathtt{n}}} P(A' \neq A_R|F) \\ &= (1-\beta_1)^{N_{\mathtt{m}}} (\beta_1)^{N_{\mathtt{n}}} \end{aligned} \tag{8.16}$$

and probability of observing $L_{\mathtt{m}}, L_{\mathtt{n}}$ when the sentence is True (i.e., factual):

$$\begin{aligned} P(L_{\mathtt{m}}, L_{\mathtt{n}}|\mathrm{T}) &= \prod_{A \in L_{\mathtt{m}}} P(A = A_R|T) \prod_{A' \in L_{\mathtt{n}}} P(A' \neq A_R|T) \\ &= (\beta_2)^{N_{\mathtt{m}}} (1-\beta_2)^{N_{\mathtt{n}}} \end{aligned} \tag{8.17}$$

where $N_{\mathtt{m}}$ and $N_{\mathtt{n}}$ are the number of matched answers and the number of not-matched answers, respectively. Hence, we can simplify Equation 8.15:

$$P(\mathrm{F}|L_{\mathtt{m}}, L_{\mathtt{n}}) = \frac{\gamma_2^{N_{\mathtt{n}}}}{\gamma_1^{N_{\mathtt{m}}} + \gamma_2^{N_{\mathtt{n}}}} \tag{8.18}$$

where $\gamma_1 = \frac{\beta_2}{1-\beta_1}$ and $\gamma_2 = \frac{\beta_1}{1-\beta_2}$. Lastly, instead of rejecting samples having an answerability score below a threshold,[5] we find empirically that soft-counting (defined below) improves the detection performance. We set both β_1 and β_2 to 0.8.

$$N'_{\mathtt{m}} = \sum_{n \text{ s.t. } A_n \in L_{\mathtt{m}}} \alpha_n; \;\; N'_{\mathtt{n}} = \sum_{n \text{ s.t. } A_n \in L_{\mathtt{n}}} \alpha_n \tag{8.19}$$

where $\alpha_n = P_{\mathtt{U}}(\text{answerable}|Q, S^n)$. Therefore, the SelfCheckGPT-QA score with the Bayes theorem that considers soft-counting is

$$\mathcal{S}_{\mathrm{QA}}(x_i, Q) = P(\mathrm{F}|L_{\mathtt{m}}, L_{\mathtt{n}}) = \frac{\gamma_2^{N'_{\mathtt{n}}}}{\gamma_1^{N'_{\mathtt{m}}} + \gamma_2^{N'_{\mathtt{n}}}} \tag{8.20}$$

Ultimately, SelfCheckGPT with QA is the average of inconsistency scores across q,

$$\mathcal{S}_{\mathrm{QA}}(x_i) = \mathbb{E}_Q \left[\mathcal{S}_{\mathrm{QA}}(x_i, Q)\right] \tag{8.21}$$

[5] α is between 0.0 (unanswerable) and 1.0 (answerable). Standard-counting $N_{\mathtt{m}}$ and $N_{\mathtt{n}}$ can be considered as a special case of soft-counting where α is set to 1.0 if α is greater than the answerability threshold and otherwise α is 0.0.

SelfCheckGPT with MQAG: Implementation

MQAG developed in the previous section (in Section 7.2.2) is used in SelfCheckGPT-QA. MQAG has two generators: `G1` generates the question and associated answer, and `G2` generates distractors. The two-stage generation is,

$$Q,A \sim P_{\texttt{G1}}(Q,A|x_i) \quad \text{and} \quad O_{\backslash A} \sim P_{\texttt{G2}}(O_{\backslash A}|Q,A,X) \tag{8.22}$$

where $O = \{A, O_{\backslash A}\} = \{o_1, ..., o_4\}$. In addition, to filter out bad (unanswerable) questions, we define an answerability score [245]:

$$\alpha = P_{\texttt{U}}(\text{answerable}|Q, \text{context}) \tag{8.23}$$

where the context is either the response X or sampled passages S^n, and $\alpha \to 0.0$ for unanswerable and $\alpha \to 1.0$ for answerable. We use α to filter out unanswerable questions which have α lower than a threshold or use α in the soft-counting measure discussed previously.

8.2.3 SelfCheckGPT with n-gram

Given samples $\{S^1, ..., S^N\}$ generated by an LLM, one can use the samples to create a new language model that approximates the LLM. In the limit, as N gets sufficiently large, the new language model will converge to the LLM that generated the responses. We can therefore approximate the LLM's token probabilities using the new language model.

In practice, due to time and/or cost constraints, there can only be a limited number of samples N. Consequently, we train a simple n-gram model using the samples $\{S^1, ..., S^N\}$ and the main response X. This is similar to the proxy LLM method described in Section 8.1.3, but this method instead trains an n-gram model on X and $\{S^1, ..., S^N\}$ to obtain an approximated probability distribution. For simplicity of notation, this notation for n-gram will omit i in $x_{i,j}$, and let w_j denote $x_{i,j}$ the j-th token of the i-th sentence. An n-gram model makes the following approximation,

$$P(w_j|w_{1:j-1}) \approx P(w_j|w_{j-n+1:j-1}) \tag{8.24}$$

and the probability of $x_{i,j}$ is approximated by counting the occurrences of n-grams,

$$\tilde{p}_{i,j} = P(w_j|w_{j-n+1:j-1}) = \frac{\text{Count}(w_{j-n+1:j})}{\text{Count}(w_{j-n+1:j-1})} \tag{8.25}$$

where Count$(w_{j-n+1:j})$ is the occurrence of the n-gram $w_{j-n+1:j}$ in $\{X, S^1, ..., S^N\}$. Note that including X in training can be considered as a *smoothing* method where the occurrence of each n-gram in X is increased by 1. SelfCheckGPT with n-gram computes the average of the log-probabilities of the sentence in response X,

$$\mathcal{S}^{\text{Avg}}_{n\text{-gram}}(x_i) = -\frac{1}{J}\sum_j \log \tilde{p}_{i,j} \tag{8.26}$$

where $\tilde{p}_{i,j}$ is the probability (of the j-th token of the i-th sentence) computed using the n-gram model. Similar to the grey-box approach, we can also use the maximum of the negative log probabilities,

$$\mathcal{S}^{\text{Max}}_{n\text{-gram}}(x_i) = \max_j \left(-\log \tilde{p}_{i,j}\right) \tag{8.27}$$

8.2.4 SelfCheckGPT with NLI

Natural Language Inference (NLI) determines whether a hypothesis follows a premise, classified into either entailment/neutral/contradiction. NLI measures have been used to measure faithfulness in summarization, where Maynez et al. [193] use a textual entailment classifier trained on MNLI [313] to determine if a summary contradicts a context or not. Inspired by the NLI-based summary assessment method, we consider using the NLI contradiction score as a SelfCheckGPT score.

For SelfCheck-NLI, we use DeBERTa-v3-large [102] fine-tuned to MNLI as the NLI model. The input for NLI classifiers is typically the `premise` concatenated to the `hypothesis`, which for this methodology is the sampled passage S^n concatenated to the sentence to be assessed x_i. Only the logits associated with the 'entailment' and 'contradiction' classes are considered,

$$P(\text{contradiction}|x_i, S^n) = \frac{\exp(z_c)}{\exp(z_e) + \exp(z_c)} \tag{8.28}$$

where z_e and z_c are the logits of the 'entailment' and 'contradiction' classes, respectively. We use this normalization instead of taking the probability of P(contradiction) out of the 3 classes directly to bound the score within 0.0 and 1.0.

$$\mathcal{S}_{\text{NLI}}(x_i) = \frac{1}{N}\sum_{n=1}^{N} P(\text{contradiction}|x_i, S^n) \tag{8.29}$$

8.2.5 SelfCheckGPT with Prompt

LLMs have recently been shown to be effective in assessing information consistency between a document and its summary in zero-shot settings such as in Chapter 7 and Luo et al. [179].

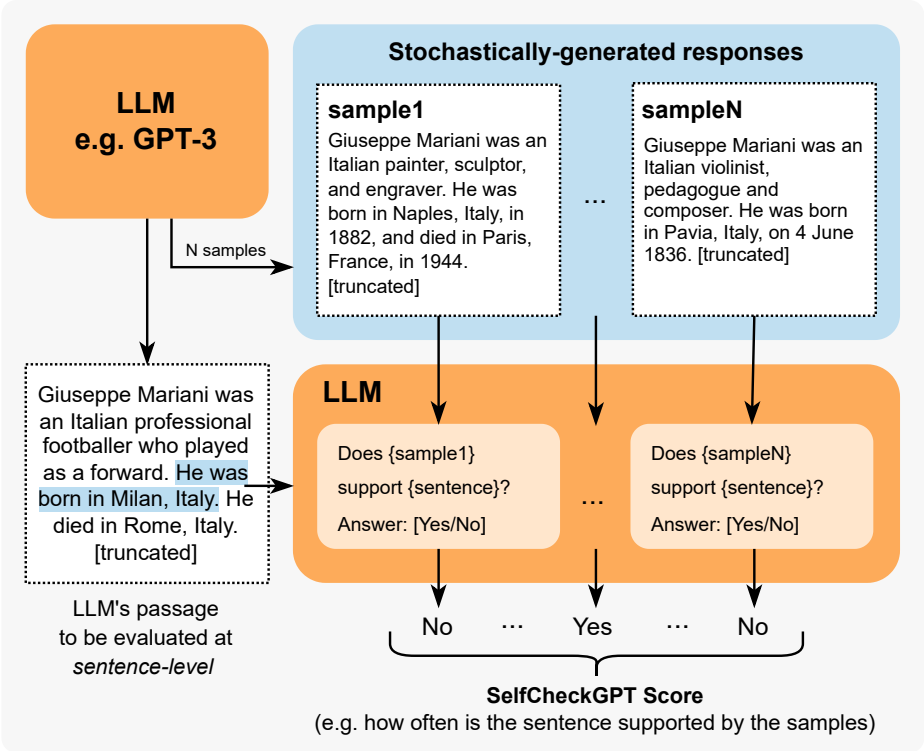


Fig. 8.3 SelfCheckGPT with Prompt. Each LLM-generated sentence is compared against stochastically generated responses with no external database. A comparison method can be, for example, through LLM prompting as shown above.

Based on these findings, we propose to query an LLM to assess whether the i-th sentence is supported by sample S^n (as the context) using the following prompt.

```
---------------------------------------------
Context: {}
Sentence: {}
Is the sentence supported by the context above?
Answer Yes or No:
---------------------------------------------
```

Initial investigation showed that GPT-3 (text-davinci-003) will output either `Yes` or `No` 98% of the time, while any remaining outputs can be set to `N/A`. The output from prompting when comparing the i-th sentence, x_i, against sample S^n is converted to score z_i^n through the mapping {`Yes`: 0.0, `No`: 1.0, `N/A`: 0.5}. The final inconsistency score is then calculated as:

$$\mathcal{S}_{\text{Prompt}}(x_i) = \frac{1}{N}\sum_{n=1}^{N} z_i^n \tag{8.30}$$

SelfCheckGPT-Prompt is illustrated in Figure 8.3. Note that our initial investigations found that less capable models such as GPT-3 (text-curie-001) or LLaMA-7B failed to effectively perform consistency assessment via such prompting. The reason why LLaMA-7B cannot perform LLM prompting is likely because it has not been adapted to follow human instructions. In contrast, as we will show in the experimental section (in Section 8.4), the chat variants of the newer Llama-2 are capable of performing LLM prompting.

8.3 Resources for Evaluating Hallucination in LMs

Prior to this work, resources for studying hallucination detection were limited to NLG tasks such as summarization and machine translation, or artificially perturbed texts. For example, as used for summary assessment in the previous chapter, QAG-CNNDM/XSum [297], XSum-Hallucination [193], SummEval [67] are datasets where summaries have consistency or factuality annotations. In machine translation, Guerreiro et al. [97] recently released a dataset consisting of 3,415 annotated for hallucination detection. In general text language model generation, HADES [166] was constructed by perturbing factual texts. and thus may not reflect true open-domain LLM hallucination.

Although these datasets are developed for hallucination detection, none of them were developed for assessing real language model generation. As a result, this work introduces a new dataset for benchmarking hallucination detection approaches. This new dataset, based on real GPT-3 generation, is constructed as follows:

1. Generate synthetic Wikipedia articles using GPT-3 (text-davinci-003) on the individuals/concepts from WikiBio [156]
2. Manually annotate the factuality of the passage at a sentence level
3. Evaluate the system's ability to detect hallucinations[6]

WikiBio is a dataset where each input contains the first paragraph (along with tabular information) of Wikipedia articles on a specific concept. We rank the WikiBio test set in terms of paragraph length and randomly sample 238 articles from the top 20% of longest articles (to ensure no very obscure concept is selected). GPT-3 (text-davinci-003) is then used to generate Wikipedia articles on a concept, using the prompt "`This is a Wikipedia passage about {concept}:`". Table 8.1 provides the statistics of GPT-3 generated passages.

[6]In the experiments described in Section 8.4.1, we consider two scenarios of non-factuality. First, 'NonFact' is when both major-inaccurate and minor-inaccurate labels are grouped into the non-factual class. Second, 'NonFact*' is a more challenging task of detecting major-inaccurate sentences in passages that are not total hallucination passages.

#Passages	#Sentences	#Tokens/passage
238	1908	184.7±36.9

Table 8.1 The statistics of WikiBio GPT-3 dataset where the number of tokens is based on the OpenAI GPT-2 tokenizer. ± indicates standard deviation.

We then annotate the sentences of the generated passages using the guidelines shown in Figure 8.4 such that each sentence is classified as either:

- **Major Inaccurate** (Non-Factual, **1**): The sentence is entirely hallucinated, i.e. the sentence is unrelated to the topic.
- **Minor Inaccurate** (Non-Factual, **0.5**): The sentence consists of some non-factual information, but the sentence is related to the topic.
- **Accurate** (Factual, **0**): The information presented in the sentence is accurate.

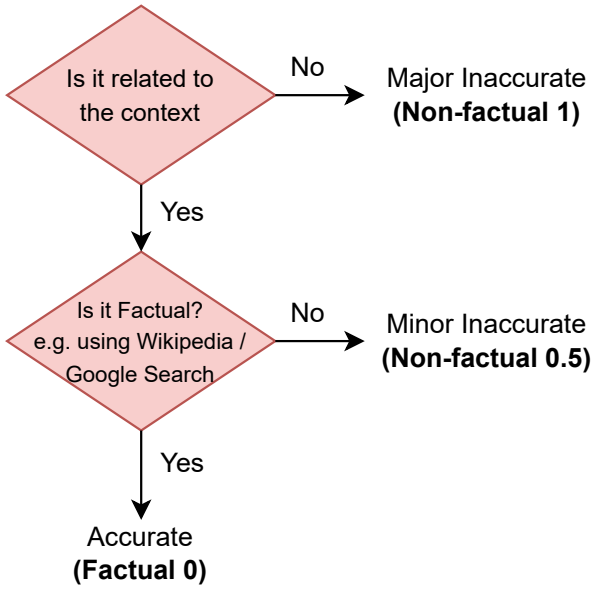


Fig. 8.4 Flowchart of the annotation process

Of the 1908 annotated sentences, 761 (39.9%) of the sentences were labelled major-inaccurate, 631 (33.1%) minor-inaccurate, and 516 (27.0%) accurate. 201 sentences in the dataset had annotations from two different annotators. To obtain a single label for this subset, if both annotators agree, then the agreed label is used. However, if there is disagreement, then the worse-case label is selected (e.g. {minor inaccurate, major inaccurate} is mapped to major inaccurate). The inter-annotator agreement, as measured by Cohen's κ [40], has κ values of

0.595 and 0.748, indicating *moderate* and *substantial* agreement [291] for the 3-class and 2-class scenarios, respectively.[7]

Annotation	3-label	2-label
Cohen's κ	0.595	0.748

Table 8.2 Inter-annotator agreement where 3-label means selecting from accurate, minor inaccurate, major inaccurate. 2-label is calculated by combining minor/major into one label.

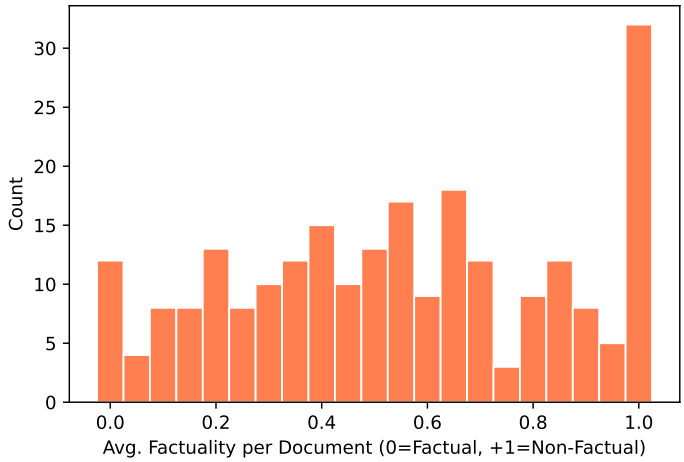


Fig. 8.5 Document factuality scores histogram plot

Passage-level scores can then be obtained by averaging the sentence-level labels in each passage. These scores are essentially the consistency scores for passages. The distribution of passage-level scores is shown in Figure 8.5, where we observe a large peak at +1.0. We refer to the points at this peak as *total hallucination*, which occurs when the information of the response is unrelated to the real concept and is entirely fabricated by the LLM.

Lastly, to increase the reproducibility of this work and to provide a resource for hallucination detection, this dataset is open-sourced at https://huggingface.co/datasets/potsawee/wiki_bio_gpt3_hallucination.

[7] 3-class refers to when selecting between accurate, minor inaccurate, major inaccurate. 2-class refers to when minor/major inaccuracies are combined into one label.

8.4 Experiments

The generative LLM used to generate passages for the dataset is **GPT-3** (text-davinci-003)[8], the state-of-the-art system at the time of creating and annotating the dataset. To obtain the main response, we set the temperature to 0.0, and set the maximum number of tokens to the model's limit. For the stochastically generated samples, we set the temperature to 1.0 and generate N=20 samples. For the proxy LLM approach, we use LLaMA [286], one of the best-performing open-source LLMs currently available at the time of conducting the experiments. For SelfCheckGPT-Prompt, we consider both GPT-3 (which is the same LLM that is used to generate passages) as well as ChatGPT (gpt-3.5-turbo) which was recently released in March 2023.

8.4.1 Sentence-level Hallucination Detection

First, we investigate whether SelfCheckGPT hallucination detection methods can identify the factuality of sentences. In detecting non-factual sentences, both major-inaccurate labels and minor-inaccurate labels are grouped together into the *non-factual* class, while the *factual* class refers to accurate sentences. In this setup, there are 1392 (out of 1908) sentences with the non-factual label and 516 sentences with the factual label. In addition, we consider a more challenging task of detecting major-inaccurate sentences in passages that are *not* total hallucination passages, which we refer to as *non-factual**. In this setup, there remain 206 non-factual* passages (or 1632 sentences). Out of 1632 sentences, 485 sentences have the non-factual label. Figure 8.6 and Table 8.3 show the performance of SelfCheckGPT methods, where the following observations can be made:

1) LLM's probabilities p correlate well with factuality. The results show that probability measures (from the LLM generating the texts) are strong baselines for assessing factuality. Factual sentences can be identified with an AUC-PR of 53.97, significantly better than the random baseline of 27.04, with the AUC-PR for hallucination detection also increasing from 72.96 to 83.21. This supports the hypothesis that when the LLMs are uncertain about generated information, generated tokens often have higher uncertainty, paving a promising direction for hallucination detection approaches. Also, the probability p measure performs better than the entropy $\mathcal{H}$ measure of top-5 tokens.

2) Proxy LLM perform noticeably worse than LLM (GPT-3). The results of proxy LLM (based on LLaMA) show that the entropy $\mathcal{H}$ measures outperform the probability measures. This suggests that using richer uncertainty information can improve factuality/hallucination

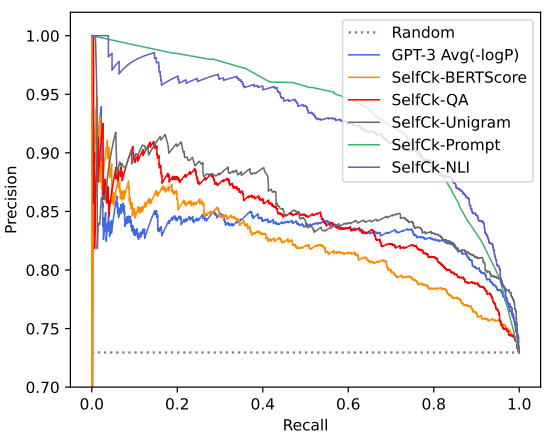
[8]This series of GPT models is now more commonly referred to as GPT-3.5.

Method	Sentence-level (AUC-PR)			Passage-level (Corr.)	
	NonFact	NonFact*	Factual	Pearson	Spearman
Random	72.96	29.72	27.04	-	-
`GPT-3 (text-davinci-003)`'s probabilities (*LLM, grey-box*)					
Avg($-\log p$)	83.21	38.89	53.97	57.04	53.93
Avg($\mathcal{H}$)†	80.73	37.09	52.07	55.52	50.87
Max($-\log p$)	87.51	35.88	50.46	57.83	55.69
Max($\mathcal{H}$)†	85.75	32.43	50.27	52.48	49.55
`LLaMA-30B`'s probabilities (*Proxy LLM, black-box*)					
Avg($-\log p$)	75.43	30.32	41.29	21.72	20.20
Avg($\mathcal{H}$)	80.80	39.01	42.97	33.80	39.49
Max($-\log p$)	74.01	27.14	31.08	-22.83	-22.71
Max($\mathcal{H}$)	80.92	37.32	37.90	35.57	38.94
SelfCheckGPT (*black-box*)					
w/ BERTScore	81.96	45.96	44.23	58.18	55.90
w/ QA	84.26	40.06	48.14	61.07	59.29
w/ Unigram (max)	85.63	41.04	58.47	64.71	64.91
w/ NLI	92.50	45.17	66.08	74.14	73.78
w/ Prompt (Llama2-chat-7B)	89.05	44.01	63.06	61.52	72.65
w/ Prompt (Llama2-chat-13B)	91.91	57.10	64.34	75.44	75.54
w/ Prompt (ChatGPT)	**93.42**	**53.19**	**67.09**	**78.32**	**78.30**

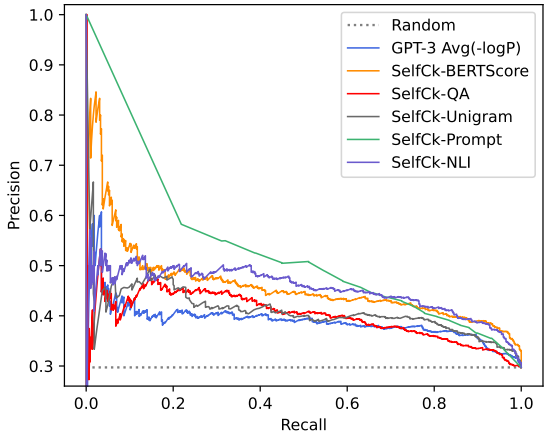
Table 8.3 Sentence-level detection performances are measured by AUC-PR where the PR plot is shown in Figure 8.6. Passage-level ranking performances are measured by Pearson correlation coefficient and Spearman's rank correlation coefficient w.r.t. human judgements, and the scatter plots are provided in Figure 8.9 and Figure 8.10. †GPT-3 API returns the top-5 tokens' probabilities, which are used to compute entropy.

detection performance, and that previously the entropy of top-5 tokens is likely to be insufficient. However, when using other proxy LLMs such as GPT-NeoX or OPT-30B, the performance is near that of the random baseline. We believe this poor performance occurs as different LLMs have different generating patterns, and so even common tokens may have a low probability in situations where the response is dissimilar to the generation style of the proxy LLM. We note that a weighted conditional LM score such as BARTScore [337] could be incorporated in future investigations.

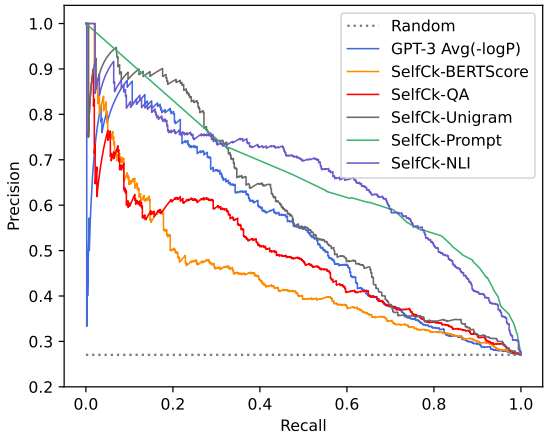
3) SelfCheckGPT outperforms grey-box approaches. It can be seen that SelfCheckGPT-Prompt *considerably* outperforms the grey-box approaches (including GPT-3's output probabilities) as well as other black-box approaches. Even other variants of SelfCheckGPT, including BERTScore, QA, *n*-gram, and NLI outperform the grey-box approaches in most setups. Interestingly, despite being the least computationally expensive method, SelfCheck-



(a) Non-Factual Sentences



(b) Non-Factual* Sentences



(c) Factual Sentences

Fig. 8.6 PR-curve of sentence-level detection tasks on the GPT-3 generated WikiBio passages.

GPT with unigram (max) works well across different setups. Essentially, when assessing a sentence, this method picks up the token with the *lowest* occurrence given all the samples. This suggests that if a token only appears a few times (or once) within the generated samples ($N = 20$), it is likely non-factual.

4) SelfCheckGPT with *n*-gram. In Table 8.4, when investigating the n-gram performance from 1-gram to 5-gram, the results show that simply finding the least likely token/n-gram is more effective than computing the average n-gram score. Additionally, as n increases, the performance of SelfCheckGPT with n-gram (max) drops. This is because as n increases, although the n-gram model can take into account more information, it requires more data to train the n-gram model and this could increase exponentially with n, e.g., for unigram, the size of all possible unigram is the vocabulary size $|V|$, and this size becomes $|V|^n$ for n-gram. Figure 8.7 shows ablation results where the 20 samples are randomly drawn to investigate

the impact of N on the performance. These results show that the performance of the bigram model starts to match the unigram model when N is near 20.

n-gram	Sent-level (AUC-PR)			Passage-level	
	NonFact	NonFact*	Fact	Pear.	Spear.
Avg($-\log p$)					
1-gram	81.52	40.33	41.76	40.68	39.22
2-gram	82.94	44.38	52.81	58.84	58.11
3-gram	83.56	44.64	53.99	62.21	63.00
4-gram	83.80	43.55	54.25	61.98	63.64
5-gram	83.45	42.31	53.98	60.68	62.96
Max($-\log p$)					
1-gram	85.63	41.04	58.47	64.71	64.91
2-gram	85.26	39.29	58.29	62.48	66.04
3-gram	84.97	37.10	57.08	57.34	60.49
4-gram	84.49	36.37	55.96	55.77	57.25
5-gram	84.12	36.19	54.89	54.84	55.97

Table 8.4 The performance using different n-gram models in the SelfCheckGPT with n-gram method.

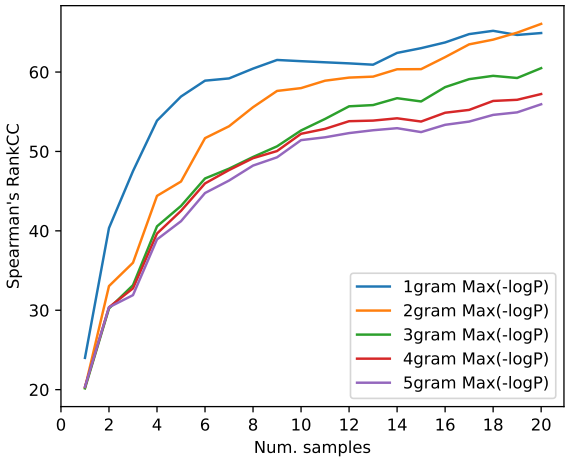


Fig. 8.7 Ablation on the number of samples N and passage-level correlation obtained by drawing samples randomly from 20 samples with replacement.

5) SelfCheckGPT with NLI. The NLI method outperforms all black-box and grey-box baselines, and its performance is close to the performance of the Prompt method. As SelfCheckGPT with Prompt can be computationally heavy, SelfCheckGPT with NLI could

be the most practical method as it provides a good trade-off between performance and computation.

Threshold for sentence-level hallucination detection

A threshold can be tuned on a development set for the best performance to find the optimal operating threshold. For example, we can optimize for the best F1 score on the WikiBio GP3 Hallucination dataset. By varying the threshold between 0.0 and 1.0 as shown in Figure 8.8, the optimal threshold for SelfCheckGPT-NLI for detecting non-factual sentences is 0.5397 and for detecting factual sentences is 0.2948. We note that the optimal threshold varies depending on the task, variant, and criterion, e.g., in this particular example, we use the F1 score. Alternative criteria are, for instance, balanced accuracy or any other metric that might align better with the task of interest.

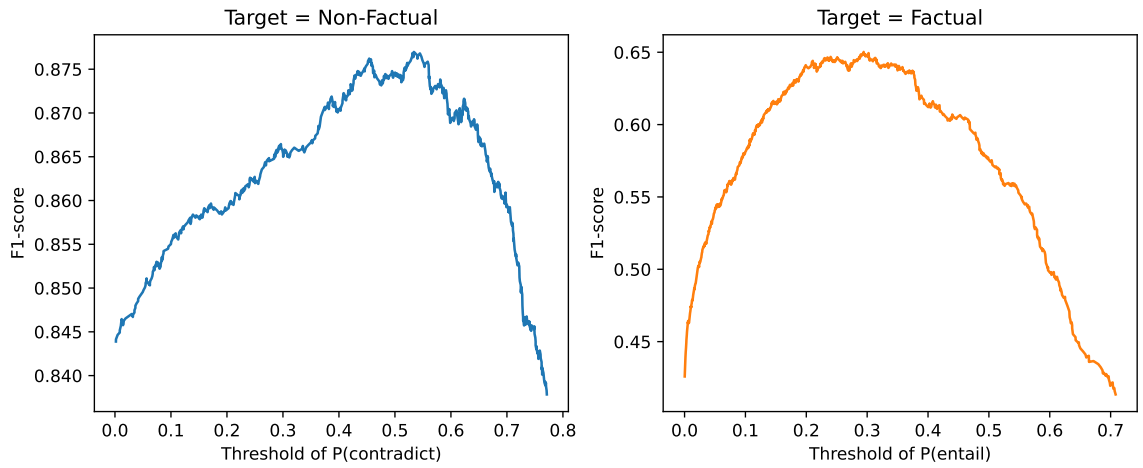


Fig. 8.8 Case study on finding the optimal threshold for SelfCheckGPT-NLI on the WikiBio-GP3 Hallucination dataset based on the F1-score as the criterion.

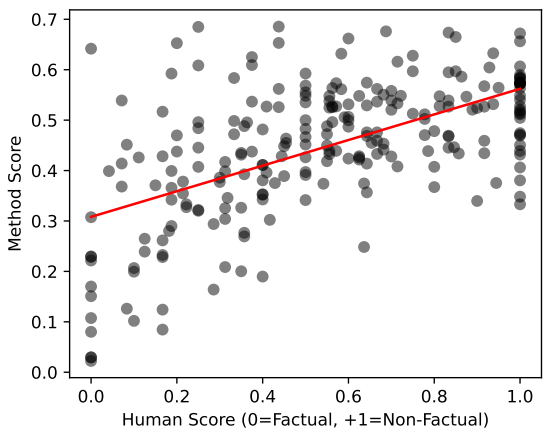
8.4.2 Passage-level Factuality Ranking

Previous results demonstrate that SelfCheckGPT is an effective approach for predicting sentence-level factuality. An additional consideration is whether SelfCheckGPT can also be used to determine the overall factuality of passages. Passage-level factuality scores are calculated by averaging the sentence-level scores over all sentences.

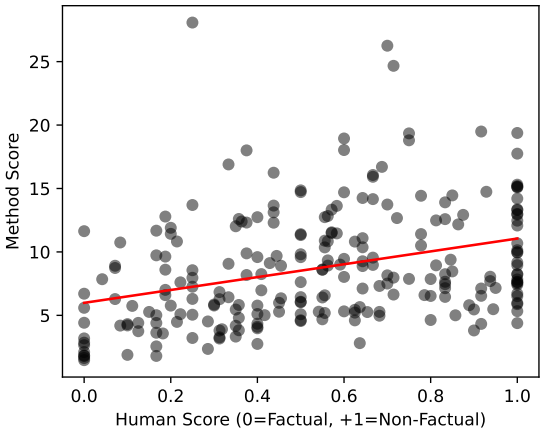
$$\mathcal{S}_{\text{passage}} = \frac{1}{N}\sum_{i}\mathcal{S}(x_i) \tag{8.31}$$

where $\mathcal{S}(x_i)$ is the sentence-level score, and N is the number of sentences in the passage. Since human judgement is somewhat subjective, averaging the sentence-level labels would lead to ground truths with less noise. Note that for Avg$(-\log p)$ and Avg$(\mathcal{H})$, we compute the average over all tokens in a passage. Whereas for Max$(-\log p)$ and Max$(\mathcal{H})$, we first take the maximum operation over tokens at the sentence level, and we then average over all sentences following Equation 8.31.

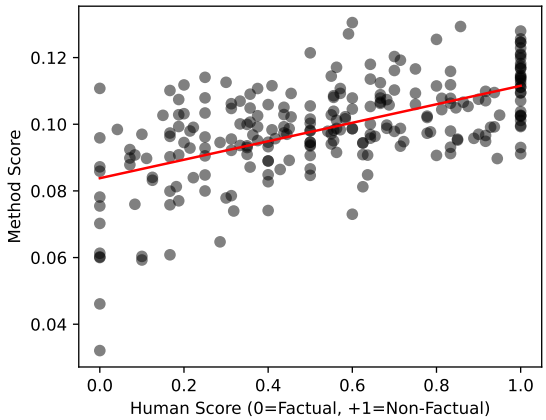
Our results in Table 8.3 and Figure 8.9 show that all SelfCheckGPT methods correlate far better with human judgements than the other baselines, including the grey-box probability and entropy methods. SelfCheckGPT-Prompt is the best-performing method, achieving the highest Pearson correlation of 78.32. Unsurprisingly, the proxy LLM approach again achieves considerably lower correlations.



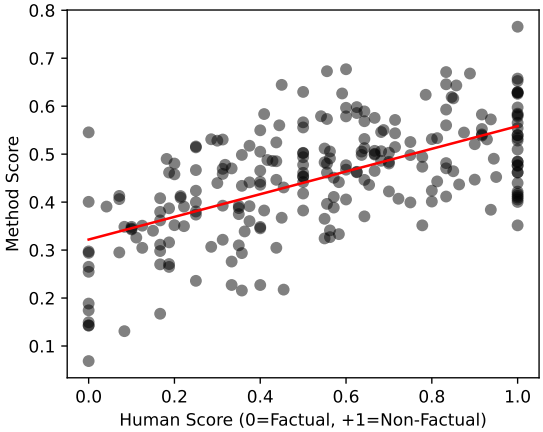
(a) GPT-3 Avg($-\log p$)



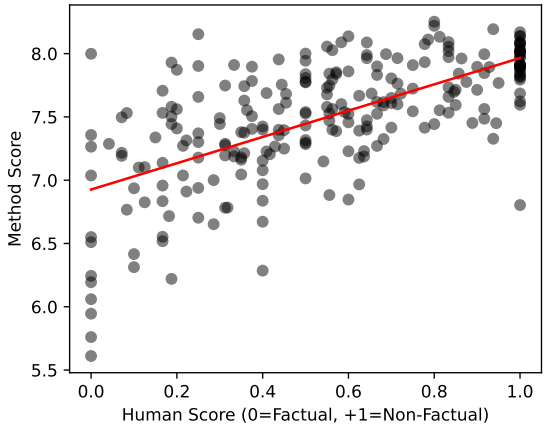
(b) LLaMA-30B Avg($\mathcal{H}$)



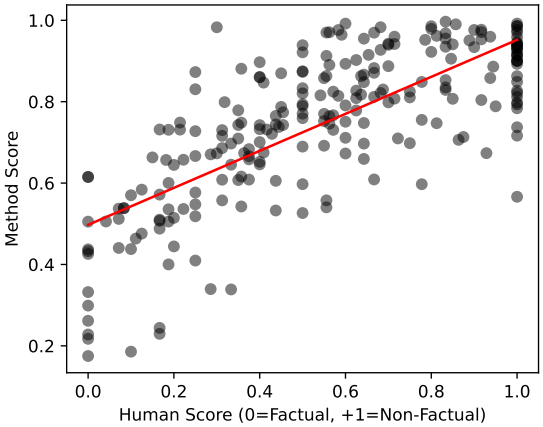
(c) SelfCheckGPT-BERTScore



(d) SelfCheckGPT-MQAG

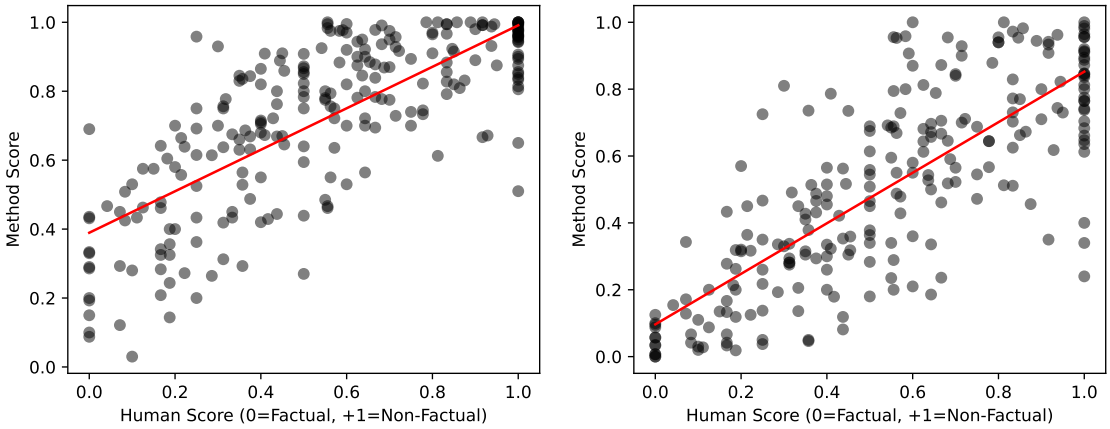


(e) SelfCheckGPT-Unigram(max)



(f) SelfCheckGPT-NLI

Fig. 8.9 Scatter plot of passage-level scores where Y-axis = Method scores, X-axis = Human scores. Correlations are reported in Table 8.3.



(a) SelfCheckGPT-Prompt (Llama2-13B) **(b)** SelfCheckGPT-Prompt (GPT-3.5)

Fig. 8.10 Scatter plot of passage-level scores where Y-axis = Method scores, X-axis = Human scores, in addition to Figure 8.9

8.4.3 Ablation Studies

Given various components in the SelfCheckGPT method, we perform ablation studies to investigate: (1) the zero-resource constraint where we compare SelfCheckGPT against each variant with external knowledge; (2) the number of samples as SelfCheckGPT is based on sampling; (3) the model choice in proxy LLM baselines; (4) the model choice in the SelfCheckGPT with Prompt.

External Knowledge (instead of SelfCheck)

If external knowledge is available, one can measure the informational consistency between the LLM response and the information source. In this experiment, we use the first paragraph of each concept that is available in WikiBio.[9] Our findings in Table 8.5 show the following. First, SelfCheckGPT, with BERTScore/QA using self-samples, can yield comparable or even better performance than when using the reference passage. Second, SelfCheckGPT with n-gram shows a large performance drop when using the WikiBio passages instead of self-samples. This failure is attributed to the fact that the WikiBio reference text alone is not sufficient to train an n-gram model. Third, in contrast, SelfCheckGPT with NLI/Prompt can benefit considerably when access to retrieved information is available. These results suggest that information consistency methods such as NLI and LLM-prompting are effective when there is a reference. While, other methods such as BERTScore, QA, and n-gram are more sensitive to the reference. Therefore, if there is an external database for the use case, it could

[9]This method is no longer zero-resource as it requires retrieving relevant knowledge from external data.

be more effective to adopt retrieval and compare (via NLI or LLM-prompting). Whereas if there is no database, SelfCheckGPT can also be effective.

Method	Sent-lvl AUC-PR			Passage-lvl	
	NoFac	NoFac*	Fact	Pear.	Spear.
SelfCheck-BERTScore	81.96	45.96	44.23	58.18	55.90
WikiBio+BERTScore	81.32	40.62	49.15	58.71	55.80
SelfCheck-QA	84.26	40.06	48.14	61.07	59.29
WikiBio+QA	84.18	45.40	52.03	57.26	53.62
SelfCheck-unigram	85.63	41.04	58.47	64.71	64.91
WikiBio+unigram	80.43	31.47	40.53	28.67	26.70
SelfCheck-NLI	92.50	45.17	66.08	74.14	73.78
WikiBio+NLI	91.18	48.14	71.61	78.84	80.00
SelfCheck-Prompt	93.42	53.19	67.09	78.32	78.30
WikiBio+Prompt	93.59	65.26	73.11	85.90	86.11

Table 8.5 The performance when using SelfCheckGPT samples versus external stored knowledge.

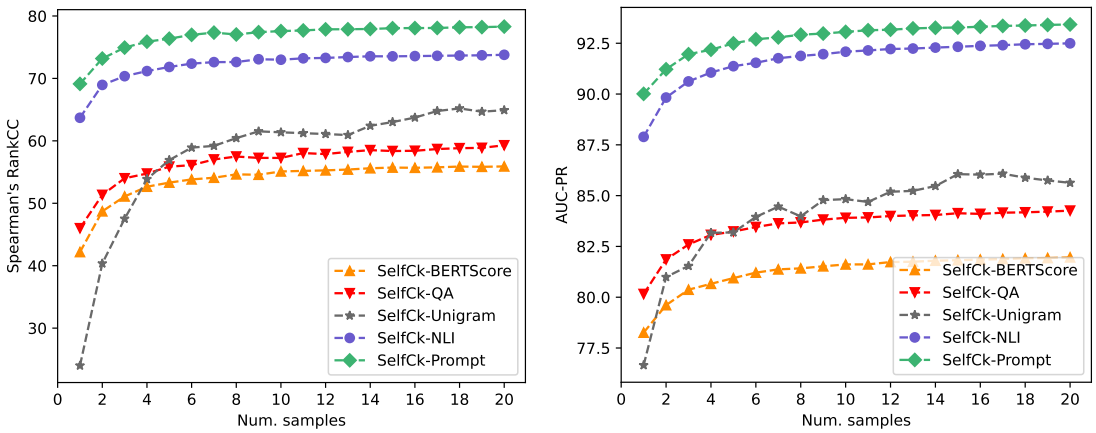
The Impact of the Number of Samples

Although sample-based methods are expected to perform better when more samples are drawn, drawing more samples has higher computational costs. Thus, we investigate performance as the number of samples varied. Our results in Figure 8.11 show that: (1) the performance of SelfCheckGPT increases smoothly as more samples are used, with performance gain diminishing as we generate more samples. (2) SelfCheckGPT with n-gram requires the highest number of samples before its performance reaches a plateau. (3) SelfCheckGPT requires at least around 5 samples to perform well while having more than 10 samples only yields a diminishing return in performance.

Model Choice for the Proxy LLM Method

Figure 8.12 illustrates that LLaMA is far better than other LLMs, and the performance of the proxy LLM method generally increases with model size. Similarly, the average probability, Avg(p), is closer to that of GPT-3 when using a larger proxy LLM as shown in Table 8.6.

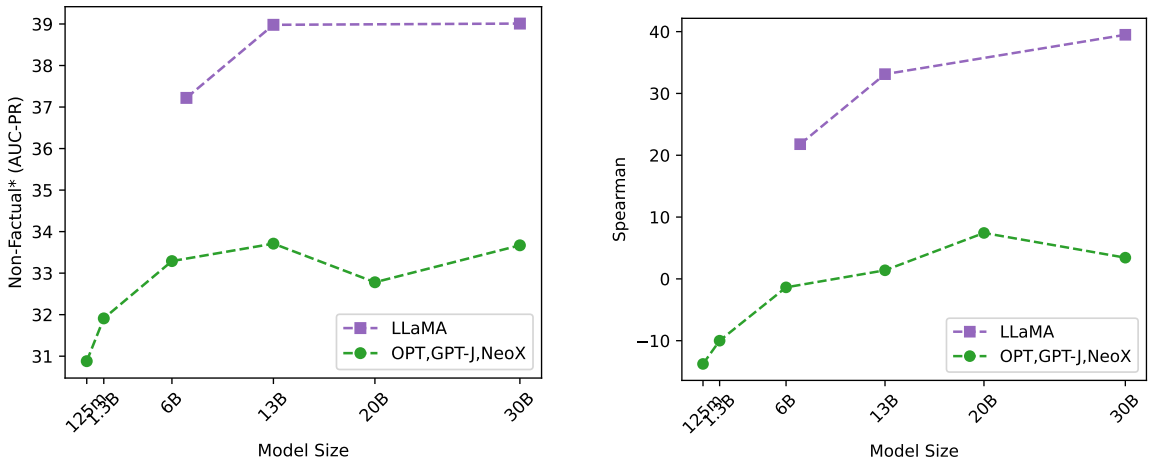
The implications of these results are that: (1) the model size is not the only factor and the amount of pre-training data can be more important similar to other tasks where smaller models (such as LLaMA-7B) have been shown to outperform larger models [286]. (2)



(a) Ranking passages (Spearman's) **(b)** Non-factual Detection (AUC-PR)

Fig. 8.11 The performance of SelfCheckGPT methods versus the number of samples.

the probability Avg(p) correlates with the detection performance. It is in line with the second point (#2) in Section 8.4.1 where models that have different generation patterns, corresponding to the model with low Avg(p), achieve low performance.



(a) Sentence-level Detection **(b)** Passage-level Ranking

Fig. 8.12 Performance of the Avg($\mathcal{H}$) method using a proxy LLM where the model sizes are: LLaMA={7B, 13B, 30B}, OPT={125m, 1.3B, 13B, 30B}, GPT-J=6B, and NeoX=20B.

Model Choice for the SelfCheckGPT-Prompt Method

Here, we investigate whether the LLM generating the text can self-check its own text. We use the prompt template provided in Section 8.2.5 for both GPT-3 (text-davinci-003) and ChatGPT (gpt-3.5-turbo). For ChatGPT, a standard system message "`You are a helpful`

LLM	Size	Avg(p)
GPT-3	175B	72.02
LLaMA	30B	65.25
LLaMA	13B	64.71
LLaMA	7B	63.70
OPT	30B	53.81
NeoX	20B	54.49
OPT	13B	52.80
GPT-J	6B	52.50
OPT	1.3B	49.20
OPT	125m	40.91

Table 8.6 Average token probability, Avg(p), over all tokens in GPT-3 generated passages.

assistant." is used in setting up the system. At the time of conducting experiments, the API costs per 1,000 tokens were $0.020 for GPT-3 and $0.002 for ChatGPT. The estimated costs for running the models to answer Yes/No on all 1908 sentences and 20 samples are around $200 for GPT-3 and $20 for ChatGPT. Given the cost, we conducted the experiments on 4 samples in the ablation study comparing GPT-3 and ChatGPT. Table 8.7 shows the breakdown of predictions made by GPT-3 and ChatGPT.

GPT-3 \ ChatGPT	Yes	No
Yes	3179	1038
No	367	3048

Table 8.7 Breakdown of the predictions made by GPT-3 and ChatGPT when prompted to answer Yes (supported) or No (not-supported).

The passage-ranking results in Table 8.8 show that GPT-3 can self-check its own text, and is better than Llama-2-chat-13B model. However, ChatGPT shows an improvement over GPT-3 in evaluating whether the sentence is supported by the context. Nevertheless, this finding shows that (1) GPT-3 can self-check itself, and (2) the LLMs considered are able to perform SelfCheckGPT-Prompt.

Text-Gen	SelfCheck-Prompt	N	Pear.	Spear.
GPT-3	GPT-3	4	73.11	74.69
GPT-3	ChatGPT	4	76.47	76.41
GPT-3	Llama-2-chat-13B	4	72.89	72.97
GPT-3	ChatGPT	20	78.32	78.30
GPT-3	Llama-2-chat-13B	20	75.44	75.54

Table 8.8 Comparison of GPT-3 (text-davinci-003) and ChatGPT (gpt-3.5.turbo) as the prompt-based text evaluator in SelfCheckGPT-Prompt. †Taken from Table 8.3 for comparison.

8.5 Chapter Summary

This chapter considers information consistency in the context of general language model generation. We propose SelfCheckGPT, a zero-resource hallucination detection approach that is applicable to any black-box LLM without the need for external resources. SelfCheckGPT is inspired by summary and NLG assessment methods, including BERTScore, MQAG, NLI, and zero-shot LLM Prompting. We collect, annotate, and release a new resource for real LLM hallucination detection such as GPT-3. Based on experimental results, SelfCheckGPT outperforms a range of considered black-box and grey-box baseline detection methods at both the sentence and passage levels as well as competitive against methods that require an external knowledge database.

Limitations and Future Work

Due to the rapid adoption of LLMs, hallucination detection has received growing interest from the NLP community, and a number of new research works in this area have been released in recent months concurrent with and following this work. For example, the survey paper by Pan et al. [221] taxonomizes self-correction strategies, which are predominantly proposed and developed in the past one-year span. Therefore, this section is meant to discuss the limitations and potential improvements of SelfCheckGPT as suggested by the relevant following work.

First, in this study, the generated texts were predominantly passages about individuals in the WikiBio dataset. To further investigate the nature of LLM's hallucination, a wider range of concepts could be explored. To this end, the following work by Mündler et al. [205], for

example, extended this idea and constructed a dataset based on Wikipedia comprising 30 diverse topics such as architecture, art, history, politics, sports, television, etc.

In addition, this work considers factuality at the sentence level, but as suggested by Min et al. [200], a single sentence may consist of multiple pieces of information and contain both factual and non-factual information. Hence, FActScore [200] considers a fine-grained factuality evaluation by decomposing sentences into *atomic facts* (i.e., pieces of information) by prompting an LLM to perform the decomposition in a zero-shot manner.

Lastly, we note that SelfCheckGPT is a *post-hoc correction* method, which addresses the model output after it has been generated, unlike *training-time* methods which update the model parameters or *inference-time* methods which use some feedback to guide the generation process. A training-time method could mitigate hallucination by using reinforcement learning with a reward function that measures the factuality of generated responses. Inference-time methods have the potential to mitigate hallucination, for example, self-consistency decoding [304] and tree-of-thought decoding [332]. SelfCheckGPT, as a post-hoc method, should be complementary to training-time and inference-time methods.

Chapter 9

Conclusion

First, this thesis studied abstractive summarization and information consistency, and their importance and challenges were outlined in Chapter 1. This thesis then introduced the background to deep learning in Chapter 2 and foundation models in Chapter 3, which have revolutionized artificial intelligence and NLP. These two chapters provided the basis for recent developments in the area. Chapter 4 then provided the background of summarization, which is the focus of this thesis.

After providing the background, this thesis investigated four main topics: (1) abstractive summarization with hierarchical RNNs in Chapter 5, (2) abstractive summarization with foundation models in Chapter 6, (3) information consistency in summarization in Chapter 7, and (4) information consistency in broader generative AI in Chapter 8. Regarding the first three topics, this thesis examines the existing technology and proposes techniques for improvements. On the other hand, regarding the last topic, this thesis proposes a new approach to address hallucination in generative language model generation, which is a relatively unexplored area prior to this work. While the proposed method for addressing hallucination in generative language model generation may not be perfect, as one of the first attempts, it is anticipated that it will inspire future research and start new advancements in the field. Lastly, this final chapter is intended to summarize the contributions made in each of the previous main chapters and to present known limitations and potential future work.

9.1 Summary of Contributions

• Chapters 2, 3, and 4, provided a relevant literature review. The link between MBR training and reinforcement learning was shown in Section 2.2.2, and hierarchical ensemble of generative models was proposed in Section 2.4.

• Chapter 5 proposed explicit utterance-level attention diversity model and unigram bias decoding, and showed that both techniques improve the hierarchical RNN system on limited data. Furthermore, It demonstrated that a multi-task learning method, which incorporates dialogue act and salient utterance information, is useful for summarization. The hierarchical model with multi-task and diversity objectives with unigram bias decoding was found to be the best configuration for the meeting summarization dataset. For a real-world spoken document application based on an ASR system with WER of around 20%, summarization performance close to that obtained from manual transcription can be achieved.

• Chapter 6 contributed to improving abstractive summarization with foundation models for long-span and efficient summarization in three directions. *First*, on content selection via sentence filtering, this chapter distinguished between training time and inference time methods, and provided a good practice for both phases. At the training time, this work showed that the oracle method with random sentences padded yields the best results. At the inference time, this work proposed multi-task content selection (MCS) that shows an improvement over sentence filtering baselines. The experimental results demonstrated that content selection is essential, in particular for longer documents such as the articles in the arXiv dataset. *Second*, on efficient encoder via local self-attention, this work presented the design considerations for local self-attention BART, and investigated the feasibility and performance of different network configurations. By combining the local self-attention technique with MCS, our system achieved state-of-the-art results at the time in terms of ROUGE scores in all three long-span summarization tasks. *Third*, this work showed that the computational cost of the transformer-based decoder becomes more significant at inference. Towards reducing this cost, this work showed that there is sparsity in the encoder-decoder attention that enables a reduction in the computational cost with minimal degradation in performance. Next, by partitioning the sentence-level attention score and augmenting the standard decoder, the modified model can perform sentence selection with the performance converging to that of the full attention baseline, while achieving an improved complexity.

In addition to the three main contributions, Chapter 6 also applied ensemble methods, which are underexplored in summarization. This work also introduced hierarchical ensemble of summarization models (HESM) which combines the token-level ensemble method and MBR decoding. Section 6.5.7 demonstrated the effectiveness of ensemble techniques where our systems were ranked first in both the Spotify Podcast Summarization Challenge 2020 and the Medical Problem List Summarization Challenge (ProbSum at BioNLP) 2023.

• Chapter 7 proposed MQAG, a novel scheme for assessing information consistency between the source and the summary based on the statistical distance between multiple-choice answer

distributions instead of text-based answer spans in existing question-answering methods. The experiments demonstrated the potential of this alternative approach which outperforms existing techniques on summary assessment datasets. The realization of the framework exploits current multiple-choice question generation and answering systems. Its performance is expected to increase as backbone systems improve, for example, the diversity of questions generated and the selection of options. Also, the framework is highly interpretable, allowing more insight into summary assessment. In addition, Chapter 7 investigated zero-shot assessment methods and proposed LLM comparative assessment, a simple and highly flexible zero-shot approach to NLG assessment such as summary assessment. The experiments demonstrated that for moderately sized LLMs, comparative assessment outperforms standard LLM prompting absolute scoring, and is an effective automatic assessment, achieving near state-of-the-art performance. Furthermore, this work also assembled and released a new resource for summary assessment called Podcast Summary Assessment (PSA). The corpus is unique in that the data consists of podcast episodes, instead of news articles which have received more attention, and the documents are long which can be more challenging for assessment methods. The benchmark of this new resource is provided in Appendix A.

• Chapter 8 provided the first (or one of the first) work to consider the task of hallucination detection for general large language model responses. The chapter proposed SelfCheckGPT, a zero-resource approach that is applicable to any black-box LLM without the need for external resources, and demonstrated the efficacy of our method. The experimental results showed that SelfCheckGPT outperforms a range of grey-box and black-box baseline detection methods at both the sentence and passage levels. This work also released an annotated dataset for GPT-3 hallucination detection with sentence-level factuality labels.

9.2 Limitations and Future Work

Given that the work in this thesis is in a fast-moving field, some of the limitations in this work, especially in the earlier part, have been further studied and/or addressed by following works by other researchers. Thus, this section splits the limitations for the discussion into two groups at the time of writing: (1) limitations with recently proposed solutions, and (2) limitations for future work.

9.2.1 Limitations with Recently Proposed Solutions

• **Automatic Evaluation**. Given the use of ROUGE as the main evaluation metric in Chapter 5 and Chapter 6, it is known that they might not correlate well with human judgements for

strong generative models. Currently, LLM-as-a-judge [351] has become a common technique for automatic evaluation.

• **Context Length of Generative Models**. In this thesis, extending the context length of summarization still requires fine-tuning the model on a reasonable-size dataset, and its maximum length is still limited to the order of 10,000 tokens. Recent work such as Position Interpolation (PI) [21] demonstrates that the context length of Llama2 can be extended from 2,048 to 32,768 tokens with only around 1,000 fine-tuning steps. Furthermore, PoSE [357] shows that by learning the positional encoding of two arbitrary adjacent chunks, it can extend the context length of Llama2 to 128k tokens.

9.2.2 Limitations for Future Work

• **Exploiting the Sparsity in Encoder-Decoder Attention**. The proposed modified architecture requires additional modules to the original attention mechanism, and the real gain in speed will depend on the balance of the sparsity against the computational cost of the additional modules. Although the modified architecture should have an improved complexity, the current realization does not result in an improvement in wall-clock speed. This is because the model choice of using RNN for the sentence encoder in Equation 6.28 leads to a large additional computational cost (specifically large k_e) because the computational cost of RNN grows with $N_1 N_2 D^2$. Because the goal is to obtain a sentence-level representation, there is an opportunity to replace RNN with hierarchical attention that runs over sentences, which could instead lead to a computational cost that grows with $N_1 N_2 D$. Additional sentence-level query and key mappings in Equation 6.26 and Equation 6.27 also incur a large computational cost.

• **MQAG**. We anticipate that MQAG could be extended in two directions. First, it could be extended to other domains such as document paraphrasing. In document paraphrasing, in addition to being consistent, all information should be included in the output. This problem, for example, can be interesting, and we anticipate that MQAG as well as other summary assessment methods could be applied to document paraphrasing. Second, the current realization of MQAG can be slow with limited computational resources (e.g., it takes around 3 seconds per question on one NVIDIA P100 GPU). To address this issue, future work could explore a more efficient realization of MQAG, and also investigate if the same or similar performance can be achieved with as few generated questions as possible, for example, by generating a smaller but more diverse set of questions and options. Another interesting could be aspect-control summary assessment where the assessment can be tailored to specific concepts by controlling question generation.

• **LLM Comparative Assessment**. The framework requires the full set of comparisons resulting in $N \cdot (N-1)$ comparisons, which for large N can be computationally prohibitive and difficult to scale up. Therefore, an interesting question is whether a similar performance level can be achieved with a subset of comparison and how to select the subset for comparison efficiently; for example, dynamic selection schemes either by considering sorting algorithms or ELO competition schemes.

• **SelfCheckGPT**. Given that this is a more active area of research, the discussion about the limitations and potential improvements to SelfCheckGPT are dedicated in Section 8.5.

References

[1] Amos Azaria and Tom Mitchell. 2023. The internal state of an llm knows when its lying. *arXiv preprint arXiv:2304.13734*.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

[3] Sanghwan Bae, Taeuk Kim, Jihoon Kim, and Sang-goo Lee. 2019. Summary level training of sentence rewriting for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pages 10–20, Hong Kong, China. Association for Computational Linguistics.

[4] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[6] Randall Balestriero, Mark Ibrahim, Vlad Sobal, Ari Morcos, Shashank Shekhar, Tom Goldstein, Florian Bordes, Adrien Bardes, Gregoire Mialon, Yuandong Tian, et al. 2023. A cookbook of self-supervised learning. *arXiv preprint arXiv:2304.12210*.

[7] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

[8] Michele Banko, Vibhu O. Mittal, and Michael J. Witbrock. 2000. Headline generation based on statistical translation. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 318–325, Hong Kong. Association for Computational Linguistics.

[9] Forrest Sheng Bao, Hebi Li, Ge Luo, Cen Chen, Yinfei Yang, Youbiao He, and Minghui Qiu. 2020. End-to-end semantics-based summary quality assessment for single-document summarization. *arXiv preprint arXiv:2005.06377*.

[10] Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

[11] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1171–1179.

[12] Manik Bhandari, Pranav Narayan Gour, Atabak Ashfaq, and Pengfei Liu. 2020. Metrics also disagree in the low scoring range: Revisiting summarization evaluation metrics. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 5702–5711, Barcelona, Spain (Online). International Committee on Computational Linguistics.

[13] Manik Bhandari, Pranav Narayan Gour, Atabak Ashfaq, Pengfei Liu, and Graham Neubig. 2020. Re-evaluating evaluation in text summarization. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9347–9359, Online. Association for Computational Linguistics.

[14] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

[15] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

[16] Erik Brynjolfsson, Danielle Li, and Lindsey R Raymond. 2023. Generative ai at work. Working Paper 31161, National Bureau of Economic Research.

[17] Daniel O Cajueiro, Arthur G Nery, Igor Tavares, Maísa K De Melo, Silvia A dos Reis, Li Weigang, and Victor RR Celestino. 2023. A comprehensive review of automatic text summarization techniques: method, data, evaluation and coding. *arXiv preprint arXiv:2301.03403*.

[18] Jaime Carbonell and Jade Goldstein. 1998. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336.

[19] Jean Carletta, Simone Ashby, Sebastien Bourban, Mike Flynn, Mael Guillemot, Thomas Hain, Jaroslav Kadlec, Vasilis Karaiskos, Wessel Kraaij, Melissa Kronenthal, Guillaume Lathoud, Mike Lincoln, Agnes Lisowska, Iain McCowan, Wilfried Post, Dennis Reidsma, and Pierre Wellner. 2005. The ami meeting corpus: A pre-announcement. In *Proceedings of the Second International Conference on Machine Learning for Multimodal Interaction*. Springer-Verlag.

[20] Danqi Chen, Jason Bolton, and Christopher D. Manning. 2016. A thorough examination of the CNN/Daily Mail reading comprehension task. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2358–2367, Berlin, Germany. Association for Computational Linguistics.

[21] Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*.

[22] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*.

[23] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR.

[24] Yen-Chun Chen and Mohit Bansal. 2018. Fast abstractive summarization with reinforce-selected sentence rewriting. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 675–686, Melbourne, Australia. Association for Computational Linguistics.

[25] Jianpeng Cheng and Mirella Lapata. 2016. Neural summarization by extracting sentences and words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 484–494, Berlin, Germany. Association for Computational Linguistics.

[26] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

[27] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

[28] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

[29] Sumit Chopra, Michael Auli, and Alexander M. Rush. 2016. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 93–98, San Diego, California. Association for Computational Linguistics.

[30] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. 2021. Rethinking attention with performers. In *International Conference on Learning Representations*.

[31] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

[32] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

[33] Christopher Cieri, David Miller, and Kevin Walker. 2004. The fisher corpus: a resource for the next generations of speech-to-text. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*, Lisbon, Portugal. European Language Resources Association (ELRA).

[34] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. 2012. Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649.

[35] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.

[36] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: pre-training text encoders as discriminators rather than generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

[37] James Clarke and Mirella Lapata. 2010. Discourse constraints for document compression. *Computational Linguistics*, 36(3):411–441.

[38] Ann Clifton, Aasish Pappu, Sravana Reddy, Yongze Yu, Jussi Karlgren, Ben Carterette, and Rosie Jones. 2020. The spotify podcasts dataset. *arXiv preprint arXiv:2004.04270*.

[39] Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. 2018. A discourse-aware attention model for abstractive summarization of long documents. In *Proceedings of the 2018 Conference*

of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers), pages 615–621, New Orleans, Louisiana. Association for Computational Linguistics.

[40] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20:37 – 46.

[41] John M. Conroy and Dianne P. O'leary. 2001. Text summarization via hidden markov models. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, page 406–407, New York, NY, USA. Association for Computing Machinery.

[42] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

[43] Hoa Dang and Karolina Owczarzak. 2009. Overview of the tac 2009 summarization track. In *Proceedings of the Text Analysis Conference (TAC 2009)*.

[44] Hoa Trang Dang. 2005. Overview of duc 2005.

[45] Hoa Trang Dang and Karolina Owczarzak. 2008. Overview of the tac 2008 update summarization task. *Theory and Applications of Categories*.

[46] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

[47] Dorottya Demszky, Kelvin Guu, and Percy Liang. 2018. Transforming question answering datasets into natural language inference datasets. *arXiv preprint arXiv:1809.02922*.

[48] Shrey Desai, Jiacheng Xu, and Greg Durrett. 2020. Compressive summarization with plausibility and salience modeling. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6259–6274, Online. Association for Computational Linguistics.

[49] Daniel Deutsch. 2022. *Methods for Text Summarization Evaluation*. Ph.D. thesis, University of Pennsylvania.

[50] Daniel Deutsch, Rotem Dror, and Dan Roth. 2022. Re-examining system-level correlations of automatic summarization evaluation metrics. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 6038–6052, Seattle, United States. Association for Computational Linguistics.

[51] Daniel Deutsch and Dan Roth. 2022. Benchmarking answer verification methods for question answering-based summarization evaluation metrics. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 3759–3765, Dublin, Ireland. Association for Computational Linguistics.

[52] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

[53] William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

[54] Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. In *Advances in Neural Information Processing Systems*, pages 13063–13075.

[55] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey for in-context learning. *arXiv preprint arXiv:2301.00234*.

[56] Yue Dong, Shuohang Wang, Zhe Gan, Yu Cheng, Jackie Chi Kit Cheung, and Jingjing Liu. 2020. Multi-fact correction in abstractive text summarization. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9320–9331, Online. Association for Computational Linguistics.

[57] Yue Dong, John Wieting, and Pat Verga. 2022. Faithful to the document or to the world? mitigating hallucinations via entity-linked knowledge in abstractive summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 1067–1082, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

[58] Qingyun Dou, Yiting Lu, Joshua Efiong, and Mark JF Gales. 2019. Attention forcing for sequence-to-sequence model training. *arXiv preprint arXiv:1909.12289*.

[59] Zi-Yi Dou, Pengfei Liu, Hiroaki Hayashi, Zhengbao Jiang, and Graham Neubig. 2021. GSum: A general framework for guided neural abstractive summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4830–4842, Online. Association for Computational Linguistics.

[60] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159.

[61] Esin Durmus, He He, and Mona Diab. 2020. FEQA: A question answering evaluation framework for faithfulness assessment in abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5055–5070, Online. Association for Computational Linguistics.

[62] Wafaa S. El-Kassas, Cherif R. Salama, Ahmed A. Rafea, and Hoda K. Mohamed. 2021. Automatic text summarization: A comprehensive survey. *Expert Systems with Applications*, 165:113679.

[63] Ahmed El-Refaiy, A.R. Abas, and I. Elhenawy. 2018. Review of recent techniques for extractive text summarization. *Journal of Theoretical and Applied Information Technology*, 96:7739–7759.

[64] Günes Erkan and Dragomir R Radev. 2004. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of artificial intelligence research*, 22:457–479.

[65] Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S Weld. 2008. Open information extraction from the web. *Communications of the ACM*, 51(12):68–74.

[66] Matan Eyal, Tal Baumel, and Michael Elhadad. 2019. Question answering as an automatic evaluation metric for news article summarization. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3938–3948, Minneapolis, Minnesota. Association for Computational Linguistics.

[67] Alexander R. Fabbri, Wojciech Kryściński, Bryan McCann, Caiming Xiong, Richard Socher, and Dragomir Radev. 2021. SummEval: Re-evaluating summarization evaluation. *Transactions of the Association for Computational Linguistics*, 9:391–409.

[68] Tobias Falke, Leonardo F. R. Ribeiro, Prasetya Ajie Utama, Ido Dagan, and Iryna Gurevych. 2019. Ranking generated summaries by correctness: An interesting but challenging application for natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2214–2220, Florence, Italy. Association for Computational Linguistics.

[69] Katja Filippova. 2010. Multi-sentence compression: Finding shortest paths in word graphs. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 322–330, Beijing, China. Coling 2010 Organizing Committee.

[70] J.G. Fiscus. 1997. A post-processing system to yield reduced word error rates: Recognizer output voting error reduction (rover). In *1997 IEEE Workshop on Automatic Speech Recognition and Understanding Proceedings*, pages 347–354.

[71] Ronald N. Forthofer, Eun Sul Lee, and Mike Hernandez. 2007. 3 - descriptive methods. In Ronald N. Forthofer, Eun Sul Lee, and Mike Hernandez, editors, *Biostatistics (Second Edition)*, second edition edition, pages 21–69. Academic Press, San Diego.

[72] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

[73] Markus Freitag, David Grangier, Qijun Tan, and Bowen Liang. 2022. High quality rather than high model probability: Minimum Bayes risk decoding with neural metrics. *Transactions of the Association for Computational Linguistics*, 10:811–825.

[74] Jinlan Fu, See-Kiong Ng, Zhengbao Jiang, and Pengfei Liu. 2023. Gptscore: Evaluate as you desire. *arXiv preprint arXiv:2302.04166*.

[75] Maria Fuentes, Enrique Alfonseca, and Horacio Rodríguez. 2007. Support vector machines for query-focused summarization trained and evaluated on pyramid data. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 57–60, Prague, Czech Republic. Association for Computational Linguistics.

[76] Yarin Gal et al. 2016. Uncertainty in deep learning.

[77] Mikel Galar, Alberto Fernandez, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. 2011. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):463–484.

[78] M.J.F. Gales, X. Liu, R. Sinha, P.C. Woodland, K. Yu, S. Matsoukas, T. Ng, K. Nguyen, L. Nguyen, J-L Gauvain, L. Lamel, and A. Messaoudi. 2007. Speech recognition system combination for machine translation. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, volume 4, pages IV–1277–IV–1280.

[79] Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

[80] Yanjun Gao, Dmitriy Dligach, Timothy Miller, and Majid Afshar. 2023. Overview of the problem list summarization (ProbSum) 2023 shared task on summarizing patients' active diagnoses and problems from electronic health record progress notes. In *The 22nd Workshop on Biomedical Natural Language Processing and BioNLP Shared Tasks*, pages 461–467, Toronto, Canada. Association for Computational Linguistics.

[81] Matt Gardner, Yoav Artzi, Victoria Basmov, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, Nitish Gupta, Hannaneh Hajishirzi, Gabriel Ilharco, Daniel Khashabi, Kevin Lin, Jiangming Liu, Nelson F. Liu, Phoebe Mulcaire, Qiang Ning, Sameer Singh, Noah A. Smith, Sanjay Subramanian, Reut Tsarfaty, Eric Wallace, Ally Zhang, and Ben Zhou. 2020. Evaluating models' local decision boundaries via contrast sets. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1307–1323, Online. Association for Computational Linguistics.

[82] Sebastian Gehrmann, Yuntian Deng, and Alexander Rush. 2018. Bottom-up abstractive summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4098–4109, Brussels, Belgium. Association for Computational Linguistics.

[83] F.A. Gers and E. Schmidhuber. 2001. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340.

[84] Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit

reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

[85] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. 2018. Unsupervised representation learning by predicting image rotations.

[86] Alexios Gidiotis and Grigorios Tsoumakas. 2020. A divide-and-conquer approach to the summarization of long documents. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:3029–3040.

[87] D. Gillick, K. Riedhammer, B. Favre, and D. Hakkani-Tur. 2009. A global optimization framework for meeting summarization. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4769–4772.

[88] Dimitra Gkatzia and Saad Mahamood. 2015. A snapshot of NLG evaluation practices 2005 - 2014. In *Proceedings of the 15th European Workshop on Natural Language Generation (ENLG)*, pages 57–60, Brighton, UK. Association for Computational Linguistics.

[89] Chih-Wen Goo and Yun-Nung Chen. 2018. Abstractive dialogue summarization with sentence-gated modeling optimized by dialogue acts. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 735–742. IEEE.

[90] Ben Goodrich, Vinay Rao, Peter J. Liu, and Mohammad Saleh. 2019. Assessing the factual accuracy of generated text. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 166–175, New York, NY, USA. Association for Computing Machinery.

[91] Tanya Goyal, Junyi Jessy Li, and Greg Durrett. 2023. News summarization and evaluation in the era of gpt-3.

[92] Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

[93] Alex Graves, Douglas Eck, Nicole Beringer, and Jürgen Schmidhuber. 2004. Biologically plausible speech recognition with lstm neural nets. In *Biologically Inspired Approaches to Advanced Information Technology*.

[94] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 369–376, New York, NY, USA. Association for Computing Machinery.

[95] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. 2020. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284.

[96] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1631–1640, Berlin, Germany. Association for Computational Linguistics.

[97] Nuno M. Guerreiro, Elena Voita, and André Martins. 2023. Looking for a needle in a haystack: A comprehensive study of hallucinations in neural machine translation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 1059–1075, Dubrovnik, Croatia. Association for Computational Linguistics.

[98] Min Gui, Junfeng Tian, Rui Wang, and Zhenglu Yang. 2019. Attention optimization for abstractive document summarization. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1222–1228, Hong Kong, China. Association for Computational Linguistics.

[99] Zhijiang Guo, Michael Schlichtkrull, and Andreas Vlachos. 2022. A survey on automated fact-checking. *Transactions of the Association for Computational Linguistics*, 10:178–206.

[100] R. Hadsell, S. Chopra, and Y. LeCun. 2006. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742.

[101] Junxian He, Wojciech Kryscinski, Bryan McCann, Nazneen Rajani, and Caiming Xiong. 2022. CTRLsum: Towards generic controllable text summarization. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5879–5915, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

[102] Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2023. DeBERTav3: Improving deBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. In *The Eleventh International Conference on Learning Representations*.

[103] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

[104] Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1693–1701.

[105] Salah Hihi and Yoshua Bengio. 1995. Hierarchical recurrent neural networks for long-term dependencies. *Advances in neural information processing systems*, 8.

[106] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*.

[107] Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

[108] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[109] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

[110] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

[111] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

[112] John J Hopfield. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558.

[113] David M. Howcroft, Anya Belz, Miruna-Adriana Clinciu, Dimitra Gkatzia, Sadid A. Hasan, Saad Mahamood, Simon Mille, Emiel van Miltenburg, Sashank Santhanam, and Verena Rieser. 2020. Twenty years of confusion in human evaluation: NLG needs evaluation sheets and standardised definitions. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 169–182, Dublin, Ireland. Association for Computational Linguistics.

[114] Wan-Ting Hsu, Chieh-Kai Lin, Ming-Ying Lee, Kerui Min, Jing Tang, and Min Sun. 2018. A unified model for extractive and abstractive summarization using inconsistency loss. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 132–141, Melbourne, Australia. Association for Computational Linguistics.

[115] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. 2021. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3451–3460.

[116] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

[117] Minghao Hu, Furu Wei, Yuxing Peng, Zhen Huang, Nan Yang, and Dongsheng Li. 2019. Read+ verify: Machine reading comprehension with unanswerable questions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6529–6537.

[118] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Xu Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 103–112.

[119] Yichong Huang, Xiachong Feng, Xiaocheng Feng, and Bing Qin. 2021. The factual inconsistency problem in abstractive text summarization: A survey.

[120] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr.

[121] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713.

[122] Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3651–3657, Florence, Italy. Association for Computational Linguistics.

[123] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12).

[124] Renlong Jie, Xiaojun Meng, Lifeng Shang, Xin Jiang, and Qun Liu. 2023. Prompt-based length controlled generation with reinforcement learning.

[125] Rosie Jones, Ben Carterette, Ann Clifton, Maria Eskevich, Gareth J. F. Jones, Jussi Karlgren, Aasish Pappu, Sravana Reddy, and Yongze Yu. 2020. Trec 2020 podcasts track overview. In *The 29th Text Retrieval Conference (TREC) notebook*.

[126] Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield Dodds, Nova DasSarma, Eli Tran-Johnson, et al. 2022. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*.

[127] Masahiro Kaneko, Masato Mita, Shun Kiyono, Jun Suzuki, and Kentaro Inui. 2020. Encoder-decoder models can benefit from pre-trained masked language models in grammatical error correction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4248–4254, Online. Association for Computational Linguistics.

[128] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

[129] Hannes Karlbom and Ann Clifton. 2020. Abstract podcast summarization using BART with longformer attention. In *Proceedings of the 29th Text REtrieval Conference (TREC)*.

[130] Andrej Karpathy. 2015. Multi-layer recurrent neural networks (lstm, gru, rnn) for character-level language models in torch. https://github.com/karpathy/char-rnn?tab=readme-ov-file.

[131] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR.

[132] M. G. KENDALL. 1938. A NEW MEASURE OF RANK CORRELATION. *Biometrika*, 30(1-2):81–93.

[133] Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. 2019. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*.

[134] Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. 2018. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 252–262, New Orleans, Louisiana. Association for Computational Linguistics.

[135] Daniel Khashabi, Tushar Khot, and Ashish Sabharwal. 2020. More bang for your buck: Natural perturbation for robust question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 163–170, Online. Association for Computational Linguistics.

[136] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[137] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

[138] Kevin Knight and Daniel Marcu. 2002. Summarization beyond sentence extraction: A probabilistic approach to sentence compression. *Artificial Intelligence*, 139(1):91–107.

[139] K. M. Knill, M. J. F. Gales, P. P. Manakul, and A. P. Caines. 2019. Automatic grammatical error detection of non-native spoken learner english. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8127–8131.

[140] Tom Kocmi and Christian Federmann. 2023. Large language models are state-of-the-art evaluators of translation quality. *arXiv preprint arXiv:2302.14520*.

[141] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

[142] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

[143] Wojciech Kryscinski, Bryan McCann, Caiming Xiong, and Richard Socher. 2020. Evaluating the factual consistency of abstractive text summarization. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9332–9346, Online. Association for Computational Linguistics.

[144] Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

[145] Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. 2023. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *The Eleventh International Conference on Learning Representations*.

[146] Shankar Kumar and William Byrne. 2004. Minimum Bayes-risk decoding for statistical machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 169–176, Boston, Massachusetts, USA. Association for Computational Linguistics.

[147] Souvik Kundu and Hwee Tou Ng. 2018. A nil-aware answer extraction framework for question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4243–4252, Brussels, Belgium. Association for Computational Linguistics.

[148] Julian Kupiec, Jan Pedersen, and Francine Chen. 1995. A trainable document summarizer. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 68–73.

[149] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. 2015. From word embeddings to document distances. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 957–966, Lille, France. PMLR.

[150] Faisal Ladhak, Esin Durmus, He He, Claire Cardie, and Kathleen McKeown. 2022. Faithful or extractive? on mitigating the faithfulness-abstractiveness trade-off in abstractive summarization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1410–1421, Dublin, Ireland. Association for Computational Linguistics.

[151] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794, Copenhagen, Denmark. Association for Computational Linguistics.

[152] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[153] Alex M Lamb, Anirudh Goyal Alias Parth Goyal, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. 2016. Professor forcing: A new algorithm for training recurrent networks. In *Advances In Neural Information Processing Systems*, pages 4601–4609.

[154] Guillaume Lample and Alexis Conneau. 2019. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*.

[155] Phong Le and Willem Zuidema. 2016. Quantifying the vanishing gradient and long distance dependency problem in recursive neural networks and recursive LSTMs. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 87–93, Berlin, Germany. Association for Computational Linguistics.

[156] Rémi Lebret, David Grangier, and Michael Auli. 2016. Generating text from structured data with application to the biography domain. *CoRR*, abs/1603.07771.

[157] Eric Lehman, Evan Hernandez, Diwakar Mahajan, Jonas Wulff, Micah J Smith, Zachary Ziegler, Daniel Nadler, Peter Szolovits, Alistair Johnson, and Emily Alsentzer. 2023. Do we still need clinical language models? In *Proceedings of the Conference on Health, Inference, and Learning*, volume 209 of *Proceedings of Machine Learning Research*, pages 578–597. PMLR.

[158] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. 1993. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867.

[159] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

[160] Manling Li, Lingyu Zhang, Heng Ji, and Richard J. Radke. 2019. Keep meeting summaries on topic: Abstractive multi-modal meeting summarization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2190–2196, Florence, Italy. Association for Computational Linguistics.

[161] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Alexander Cosgrove, Christopher D Manning, Christopher Re, Diana Acosta-Navas, Drew Arad Hudson,

Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue WANG, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri S. Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Andrew Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2023. Holistic evaluation of language models. *Transactions on Machine Learning Research*. Featured Certification, Expert Certification.

[162] Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

[163] Chin-Yew Lin and Franz Josef Och. 2004. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 605–612, Barcelona, Spain.

[164] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. 2023. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*.

[165] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating wikipedia by summarizing long sequences. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[166] Tianyu Liu, Yizhe Zhang, Chris Brockett, Yi Mao, Zhifang Sui, Weizhu Chen, and Bill Dolan. 2022. A token-level reference-free hallucination detection benchmark for free-form text generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6723–6737, Dublin, Ireland. Association for Computational Linguistics.

[167] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-eval: NLG evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore. Association for Computational Linguistics.

[168] Yang Liu and Mirella Lapata. 2019. Hierarchical transformers for multi-document summarization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5070–5081, Florence, Italy. Association for Computational Linguistics.

[169] Yang Liu and Mirella Lapata. 2019. Text summarization with pretrained encoders. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3730–3740, Hong Kong, China. Association for Computational Linguistics.

[170] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

[171] Yixin Liu, Alex Fabbri, Pengfei Liu, Yilun Zhao, Linyong Nan, Ruilin Han, Simeng Han, Shafiq Joty, Chien-Sheng Wu, Caiming Xiong, and Dragomir Radev. 2023. Revisiting the gold standard: Grounding summarization evaluation with robust human evaluation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4140–4170, Toronto, Canada. Association for Computational Linguistics.

[172] Yixin Liu and Pengfei Liu. 2021. SimCLS: A simple framework for contrastive learning of abstractive summarization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 1065–1072, Online. Association for Computational Linguistics.

[173] Adian Liusie, Potsawee Manakul, and Mark J. F. Gales. 2024. Llm comparative assessment: Zero-shot nlg evaluation through pairwise comparisons using large language models.

[174] Adian Liusie, Potsawee Manakul, and Mark JF Gales. 2023. Mitigating word bias in zero-shot prompt-based classifiers. *arXiv preprint arXiv:2309.04992*.

[175] Adian Liusie, Vatsal Raina, and Mark Gales. 2023. "world knowledge" in multiple choice reading comprehension. In *Proceedings of the Sixth Fact Extraction and VERification Workshop (FEVER)*, pages 49–57, Dubrovnik, Croatia. Association for Computational Linguistics.

[176] Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. 2023. The flan collection: Designing data and methods for effective instruction tuning. *arXiv preprint arXiv:2301.13688*.

[177] Y. Lu, Mark J.F. Gales, Kate M. Knill, P. Manakul, L. Wang, and Y. Wang. 2019. Impact of ASR Performance on Spoken Grammatical Error Detection. In *Proc. Interspeech 2019*, pages 1876–1880.

[178] Yiting Lu, Mark JF Gales, Katherine Knill, Potsawee Manakul, and Yu Wang. 2019. Disfluency detection for spoken learner english. In *SLaTE*, pages 74–78.

[179] Zheheng Luo, Qianqian Xie, and Sophia Ananiadou. 2023. Chatgpt as a factual inconsistency evaluator for abstractive text summarization. *arXiv preprint arXiv:2303.15621*.

[180] Andrey Malinin and Mark Gales. 2018. Predictive uncertainty estimation via prior networks. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[181] Andrey Malinin and Mark Gales. 2021. Uncertainty estimation in autoregressive structured prediction. In *International Conference on Learning Representations*.

[182] Andrey Malinin, Bruno Mlodozeniec, and Mark Gales. 2020. Ensemble distribution distillation. In *International Conference on Learning Representations*.

[183] Potsawee Manakul, Yassir Fathullah, Adian Liusie, Vyas Raina, Vatsal Raina, and Mark Gales. 2023. CUED at ProbSum 2023: Hierarchical ensemble of summarization models. In *The 22nd Workshop on Biomedical Natural Language Processing and BioNLP Shared Tasks*, pages 516–523, Toronto, Canada. Association for Computational Linguistics.

[184] Potsawee Manakul and Mark Gales. 2020. CUED_SPEECH at TREC 2020 Podcast Summarisation Track. In *Proceedings of the 29th Text REtrieval Conference (TREC 2020)*, Online. National Institute of Standards and Technology (NIST).

[185] Potsawee Manakul and Mark Gales. 2021. Long-span summarization via local attention and content selection. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6026–6041, Online. Association for Computational Linguistics.

[186] Potsawee Manakul and Mark Gales. 2021. Sparsity and sentence structure in encoder-decoder attention of summarization systems. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9359–9368, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

[187] Potsawee Manakul and Mark JF Gales. 2022. Podcast Summary Assessment: A resource for evaluating summary assessment methods. *arXiv preprint arXiv:2208.13265*.

[188] Potsawee Manakul, Mark J.F. Gales, and Linlin Wang. 2020. Abstractive Spoken Document Summarization Using Hierarchical Model with Multi-Stage Attention Diversity Optimization. In *Proc. Interspeech 2020*, pages 4248–4252.

[189] Potsawee Manakul, Adian Liusie, and Mark Gales. 2023. MQAG: Multiple-choice question answering and generation for assessing information consistency in summarization. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 39–53, Nusa Dua, Bali. Association for Computational Linguistics.

[190] Potsawee Manakul, Adian Liusie, and Mark Gales. 2023. SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Singapore. Association for Computational Linguistics.

[191] Inderjeet Mani, David House, Gary Klein, Lynette Hirschman, Therese Firmin, and Beth Sundheim. 1999. The TIPSTER SUMMAC text summarization evaluation. In *Ninth Conference of the European Chapter of the Association for Computational Linguistics*, pages 77–85, Bergen, Norway. Association for Computational Linguistics.

[192] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: the penn treebank. *Comput. Linguist.*, 19(2):313–330.

[193] Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. On faithfulness and factuality in abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1906–1919, Online. Association for Computational Linguistics.

[194] Ryan McDonald. 2007. A study of global inference algorithms in multi-document summarization. In *Advances in Information Retrieval*, pages 557–564, Berlin, Heidelberg. Springer Berlin Heidelberg.

[195] Clara Meister, Tim Vieira, and Ryan Cotterell. 2020. Best-first beam search. *Transactions of the Association for Computational Linguistics*, 8:795–809.

[196] Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 14014–14024.

[197] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed precision training. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

[198] Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing order into text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 404–411, Barcelona, Spain. Association for Computational Linguistics.

[199] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space.

[200] Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2023. Factscore: Fine-grained atomic evaluation of factual precision in long form text generation. *arXiv preprint arXiv:2305.14251*.

[201] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11048–11064, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

[202] Andrew H. Morris, George M. Kasper, and Dennis A. Adams. 1992. The effects and limitations of automated text condensing on reading comprehension performance. *Information Systems Research*, 3(1):17–35.

[203] M. F. Mridha, Aklima Akter Lima, Kamruddin Nur, Sujoy Chandra Das, Mahmud Hasan, and Muhammad Mohsin Kabir. 2021. A survey of automatic text summarization: Progress, process and challenges. *IEEE Access*, 9:156043–156070.

[204] Khalil Mrini, Can Liu, and Markus Dreyer. 2021. Rewards with negative examples for reinforced topic-focused abstractive summarization. In *Proceedings of the Third Workshop on New Frontiers in Summarization*, pages 33–38, Online and in Dominican Republic. Association for Computational Linguistics.

[205] Niels Mündler, Jingxuan He, Slobodan Jenko, and Martin Vechev. 2023. Self-contradictory hallucinations of large language models: Evaluation, detection and mitigation. *arXiv preprint arXiv:2305.15852*.

[206] Masaaki Nagata. 1997. A self-organizing Japanese word segmenter using heuristic word identification and re-estimation. In *Fifth Workshop on Very Large Corpora*.

[207] Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. 2017. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[208] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Cauglar Gulcehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany. Association for Computational Linguistics.

[209] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium. Association for Computational Linguistics.

[210] Shashi Narayan, Joshua Maynez, Jakub Adamek, Daniele Pighin, Blaz Bratanic, and Ryan McDonald. 2020. Stepwise extractive summarization and planning with structured transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4143–4159, Online. Association for Computational Linguistics.

[211] N. Nazari and M. A. Mahdavi. 2019. A survey on automatic text summarization. *Journal of AI and Data Mining*, 7(1):121–135.

[212] Ani Nenkova and Kathleen McKeown. 2012. A survey of text summarization techniques. *Mining text data*, pages 43–76.

[213] Ani Nenkova and Rebecca Passonneau. 2004. Evaluating content selection in summarization: The pyramid method. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 145–152, Boston, Massachusetts, USA. Association for Computational Linguistics.

[214] Mohammad Norouzi, Samy Bengio, Navdeep Jaitly, Mike Schuster, Yonghui Wu, Dale Schuurmans, et al. 2016. Reward augmented maximum likelihood for neural structured prediction. In *Advances In Neural Information Processing Systems*, pages 1723–1731.

[215] Jekaterina Novikova, Ondřej Dušek, Amanda Cercas Curry, and Verena Rieser. 2017. Why we need new evaluation metrics for NLG. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2241–2252, Copenhagen, Denmark. Association for Computational Linguistics.

[216] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

[217] Nedjma Ousidhoum, Zhangdie Yuan, and Andreas Vlachos. 2022. Varifocal question generation for fact-checking. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2532–2544, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

[218] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

[219] Paul Owoicho and Jeff Dalton. 2020. Glasgow Representation and Information Learning Lab (GRILL) at TREC 2020 Podcasts Track. In *Proceedings of the 29th Text REtrieval Conference (TREC)*.

[220] Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The Proposition Bank: An Annotated Corpus of Semantic Roles. *Computational Linguistics*, 31(1):71–106.

[221] Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. 2023. Automatically correcting large language models: Surveying the landscape of diverse self-correction strategies.

[222] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

[223] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4052–4061. PMLR.

[224] Romain Paulus, Caiming Xiong, and Richard Socher. 2018. A deep reinforced model for abstractive summarization. In *International Conference on Learning Representations*.

[225] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The refinedweb dataset for falcon llm: Outperforming curated corpora with web data, and web data only.

[226] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Leon Derczynski, Xingjian Du, Matteo Grella, Kranthi Gv, Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartłomiej Koptyra, Hayden Lau, Jiaju Lin, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Johan Wind, Stanisław Woźniak, Zhenyuan Zhang, Qinghua Zhou, Jian Zhu, and Rui-Jie Zhu. 2023. RWKV: Reinventing RNNs for the transformer era. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14048–14077, Singapore. Association for Computational Linguistics.

[227] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

[228] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

[229] Maxime Peyrard. 2019. Studying summarization evaluation metrics in the appropriate scoring range. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5093–5100, Florence, Italy. Association for Computational Linguistics.

[230] Maxime Peyrard, Teresa Botschen, and Iryna Gurevych. 2017. Learning to score system summaries for better content selection evaluation. In *Proceedings of the Workshop on New Frontiers in Summarization*, pages 74–84, Copenhagen, Denmark. Association for Computational Linguistics.

[231] Jonathan Pilault, Raymond Li, Sandeep Subramanian, and Chris Pal. 2020. On extractive and abstractive neural document summarization with transformer language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9308–9319, Online. Association for Computational Linguistics.

[232] Dan Povey. 2003. *Discriminative Training for Large Vocabulary Speech Recognition*. Ph.D. thesis, Cambridge University.

[233] Daniel Povey, Gaofeng Cheng, Yiming Wang, Ke Li, Hainan Xu, Mahsa Yarmohammadi, and Sanjeev Khudanpur. 2018. Semi-orthogonal low-rank matrix factorization for deep neural networks. In *Interspeech*.

[234] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. 2011. The kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*. IEEE Signal Processing Society.

[235] Rohit Prabhavalkar, Tara N Sainath, Yonghui Wu, Patrick Nguyen, Zhifeng Chen, Chung-Cheng Chiu, and Anjuli Kannan. 2018. Minimum word error rate training for attention-based sequence-to-sequence models. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4839–4843. IEEE.

[236] Ofir Press, Noah Smith, and Mike Lewis. 2022. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*.

[237] Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, et al. 2023. Large language models are effective text rankers with pairwise ranking prompting. *arXiv preprint arXiv:2306.17563*.

[238] Jiezhong Qiu, Hao Ma, Omer Levy, Wen-tau Yih, Sinong Wang, and Jie Tang. 2020. Blockwise self-attention for long document understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2555–2565, Online. Association for Computational Linguistics.

[239] Dragomir R. Radev, Eduard Hovy, and Kathleen McKeown. 2002. Introduction to the special issue on summarization. *Computational Linguistics*, 28(4):399–408.

[240] Dragomir R. Radev, Simone Teufel, Horacio Saggion, Wai Lam, John Blitzer, Hong Qi, Arda Çelebi, Danyu Liu, and Elliott Drabek. 2003. Evaluation challenges in large-scale document summarization. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 375–382, Sapporo, Japan. Association for Computational Linguistics.

[241] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

[242] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

[243] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *ICML 2023 Workshop The Many Facets of Preference-Based Learning*.

[244] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*.

[245] Vatsal Raina and Mark Gales. 2022. Answer uncertainty and unanswerability in multiple-choice machine reading comprehension. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1020–1034, Dublin, Ireland. Association for Computational Linguistics.

[246] Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

[247] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

[248] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence level training with recurrent neural networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

[249] Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020. COMET: A neural framework for MT evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online. Association for Computational Linguistics.

[250] Ehud Reiter. 2022. Summarisation datasets should contain summaries!

[251] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. 2017. Self-critical sequence training for image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7008–7024.

[252] Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal. Association for Computational Linguistics.

[253] David Saad. 1998. Online algorithms and stochastic approximations. *Online Learning*, 5(3):6.

[254] Hasim Sak, Matt Shannon, Kanishka Rao, and Françoise Beaufays. 2017. Recurrent neural aligner: An encoder-decoder neural network model for sequence to sequence mapping. In *INTERSPEECH*.

[255] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

[256] Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M Rush. 2022. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*.

[257] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

[258] Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. 2023. Are emergent abilities of large language models a mirage? In *Thirty-seventh Conference on Neural Information Processing Systems*.

[259] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.

[260] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[261] Thomas Scialom, Paul-Alexis Dray, Sylvain Lamprier, Benjamin Piwowarski, Jacopo Staiano, Alex Wang, and Patrick Gallinari. 2021. QuestEval: Summarization asks for fact-based evaluation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6594–6604, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

[262] Thomas Scialom, Sylvain Lamprier, Benjamin Piwowarski, and Jacopo Staiano. 2019. Answers unite! unsupervised metrics for reinforced summarization models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3246–3256, Hong Kong, China. Association for Computational Linguistics.

[263] Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

[264] Thibault Sellam, Dipanjan Das, and Ankur Parikh. 2020. BLEURT: Learning robust metrics for text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7881–7892, Online. Association for Computational Linguistics.

[265] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

[266] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

[267] Guokan Shang, Wensi Ding, Zekun Zhang, Antoine Tixier, Polykarpos Meladianos, Michalis Vazirgiannis, and Jean-Pierre Lorré. 2018. Unsupervised abstractive meeting

summarization with multi-sentence compression and budgeted submodular maximization. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 664–674, Melbourne, Australia. Association for Computational Linguistics.

[268] Matt Shannon. 2017. Optimizing expected word error rate via sampling for speech recognition. *arXiv preprint arXiv:1706.02776*.

[269] Ori Shapira, David Gabay, Yang Gao, Hadar Ronen, Ramakanth Pasunuru, Mohit Bansal, Yael Amsterdamer, and Ido Dagan. 2019. Crowdsourcing lightweight pyramids for manual summary evaluation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 682–687, Minneapolis, Minnesota. Association for Computational Linguistics.

[270] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

[271] Tian Shi, Yaser Keneshloo, Naren Ramakrishnan, and Chandan K Reddy. 2021. Neural abstractive text summarization with sequence-to-sequence models. *ACM Transactions on Data Science*, 2(1):1–37.

[272] Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. 2021. Retrieval augmentation reduces hallucination in conversation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3784–3803, Punta Cana, Dominican Republic. Association for Computational Linguistics.

[273] K. C. Sim, W. J. Byrne, M. J. F. Gales, H. Sahbi, and P. C. Woodland. 2007. Consensus network decoding for statistical machine translation system combination. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, volume 4, pages IV–105–IV–108.

[274] Kaiqiang Song, Chen Li, Xiaoyang Wang, Dong Yu, and Fei Liu. 2020. Automatic summarization of open-domain podcast episodes. *arXiv preprint arXiv:2011.04132*.

[275] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*.

[276] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Training very deep networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

[277] Josef Steinberger and Karel Ježek. 2009. Evaluation measures for text summarization. *Computing and Informatics*, 28(2):251–275.

[278] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2021. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*.

[279] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[280] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

[281] Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. 2020. Sparse sinkhorn attention. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 9438–9447. PMLR.

[282] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2021. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*.

[283] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*.

[284] James Thorne, Andreas Vlachos, Oana Cocarascu, Christos Christodoulopoulos, and Arpit Mittal. 2018. The Fact Extraction and VERification (FEVER) shared task. In *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)*.

[285] Louis L Thurstone. 1927. A law of comparative judgment. *Psychological review*, 34(4):273.

[286] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

[287] Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. 2017. NewsQA: A machine comprehension dataset. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 191–200, Vancouver, Canada. Association for Computational Linguistics.

[288] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–85, Berlin, Germany. Association for Computational Linguistics.

[289] Hans van Halteren and Simone Teufel. 2003. Examining the consensus between human summaries: initial experiments with factoid analysis. In *Proceedings of the HLT-NAACL 03 Text Summarization Workshop*, pages 57–64.

[290] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

[291] Anthony J Viera, Joanne M Garrett, et al. 2005. Understanding interobserver agreement: the kappa statistic. *Fam med*, 37(5):360–363.

[292] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in neural information processing systems*, pages 2692–2700.

[293] Andrew Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269.

[294] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.

[295] Rohit Voleti, Julie M Liss, and Visar Berisha. 2019. Investigating the effects of word substitution errors on sentence embeddings. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7315–7319. IEEE.

[296] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85.

[297] Alex Wang, Kyunghyun Cho, and Mike Lewis. 2020. Asking and answering questions to evaluate the factual consistency of summaries. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5008–5020, Online. Association for Computational Linguistics.

[298] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. *SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems*. Curran Associates Inc., Red Hook, NY, USA.

[299] Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

[300] Chaojun Wang and Rico Sennrich. 2020. On exposure bias, hallucination and domain shift in neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3544–3552, Online. Association for Computational Linguistics.

[301] Jiaan Wang, Yunlong Liang, Fandong Meng, Haoxiang Shi, Zhixu Li, Jinan Xu, Jianfeng Qu, and Jie Zhou. 2023. Is chatgpt a good nlg evaluator? a preliminary study. *arXiv preprint arXiv:2303.04048*.

[302] Shibo Wang and Pankaj Kanwar. 2019. BFloat16: The secret to high performance on Cloud TPUs.

[303] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

[304] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

[305] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. 2022. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. *arXiv preprint arXiv:2204.07705*.

[306] Yufei Wang, Wanjun Zhong, Liangyou Li, Fei Mi, Xingshan Zeng, Wenyong Huang, Lifeng Shang, Xin Jiang, and Qun Liu. 2023. Aligning large language models with human: A survey.

[307] Zhiguo Wang, Wael Hamza, and Radu Florian. 2017. Bilateral multi-perspective matching for natural language sentences. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 4144–4150.

[308] Jonathan J. Webster and Chunyu Kit. 1992. Tokenization as the initial phase in NLP. In *COLING 1992 Volume 4: The 14th International Conference on Computational Linguistics*.

[309] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

[310] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

[311] Lilian Weng. 2018. Attention? attention! *lilianweng.github.io/lil-log*.

[312] P. J. Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

[313] Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

[314] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

[315] Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280.

[316] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

[317] Pak-kwong Wong and Chorkin Chan. 1996. Chinese word segmentation based on maximum matching and word binding force. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*.

[318] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. 2022. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, pages 23965–23998. PMLR.

[319] Hanlu Wu, Tengfei Ma, Lingfei Wu, Tariro Manyumwa, and Shouling Ji. 2020. Unsupervised reference-free summary quality evaluation via contrastive learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3612–3621, Online. Association for Computational Linguistics.

[320] Yuxiang Wu and Baotian Hu. 2018. Learning to extract coherent summary via deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

[321] Menglin Xia, Ekaterina Kochmar, and Ted Briscoe. 2019. Automatic learner summary assessment for reading comprehension. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2532–2542, Minneapolis, Minnesota. Association for Computational Linguistics.

[322] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR.

[323] Wen Xiao and Giuseppe Carenini. 2019. Extractive summarization of long documents by combining global and local context. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3011–3021, Hong Kong, China. Association for Computational Linguistics.

[324] Wen Xiao and Giuseppe Carenini. 2020. Systematically exploring redundancy reduction in summarizing long documents. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 516–528, Suzhou, China. Association for Computational Linguistics.

[325] Yijun Xiao and William Yang Wang. 2021. On hallucination and predictive uncertainty in conditional language generation. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2734–2744, Online. Association for Computational Linguistics.

[326] Jiacheng Xu and Greg Durrett. 2019. Neural extractive text summarization with syntactic compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3292–3303, Hong Kong, China. Association for Computational Linguistics.

[327] Jiacheng Xu, Zhe Gan, Yu Cheng, and Jingjing Liu. 2020. Discourse-aware neural extractive text summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5021–5031, Online. Association for Computational Linguistics.

[328] Xinnuo Xu, Ondřej Dušek, Jingyi Li, Verena Rieser, and Ioannis Konstas. 2020. Fact-based content weighting for evaluating abstractive summarisation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5071–5081, Online. Association for Computational Linguistics.

[329] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

[330] Geoffrey Hinton Yann LeCun, Yoshua Bengio. 2015. Deep learning. *Nature*, 521:436–444.

[331] Jin-ge Yao, Xiaojun Wan, and Jianguo Xiao. 2017. Recent advances in document summarization. *Knowledge and Information Systems*, 53:297–336.

[332] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

[333] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183.

[334] Wenpeng Yin, Dragomir Radev, and Caiming Xiong. 2021. DocNLI: A large-scale dataset for document-level natural language inference. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4913–4922, Online. Association for Computational Linguistics.

[335] Weihao Yu, Zihang Jiang, Yanfei Dong, and Jiashi Feng. 2020. Reclor: A reading comprehension dataset requiring logical reasoning. In *International Conference on Learning Representations*.

[336] Lin Yuan and Zhou Yu. 2019. Abstractive dialog summarization with semantic scaffolds. *arXiv preprint arXiv:1910.00825*.

[337] Weizhe Yuan, Graham Neubig, and Pengfei Liu. 2021. Bartscore: Evaluating generated text as text generation. In *Advances in Neural Information Processing Systems*, volume 34, pages 27263–27277. Curran Associates, Inc.

[338] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33.

[339] Matthew D. Zeiler. 2012. Adadelta: An adaptive learning rate method. *ArXiv*, abs/1212.5701.

[340] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. 2020. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR.

[341] Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, et al. 2023. Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792*.

[342] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

[343] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with BERT. In *International Conference on Learning Representations*.

[344] Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B Hashimoto. 2023. Benchmarking large language models for news summarization. *arXiv preprint arXiv:2301.13848*.

[345] Xingxing Zhang, Furu Wei, and Ming Zhou. 2019. HIBERT: Document level pre-training of hierarchical bidirectional transformers for document summarization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5059–5069, Florence, Italy. Association for Computational Linguistics.

[346] Zhuosheng Zhang, Yuwei Wu, Hai Zhao, Zuchao Li, Shuailiang Zhang, Xi Zhou, and Xiang Zhou. 2020. Semantics-aware bert for language understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9628–9635.

[347] Wei Zhao, Maxime Peyrard, Fei Liu, Yang Gao, Christian M. Meyer, and Steffen Eger. 2019. MoverScore: Text generation evaluating with contextualized embeddings and earth mover distance. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 563–578, Hong Kong, China. Association for Computational Linguistics.

[348] Zheng Zhao, Shay B. Cohen, and Bonnie Webber. 2020. Reducing quantity hallucinations in abstractive summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2237–2249, Online. Association for Computational Linguistics.

[349] Zhou Zhao, Haojie Pan, Changjie Fan, Yan Liu, Linlin Li, Min Yang, and Deng Cai. 2019. Abstractive meeting summarization via hierarchical adaptive segmental network learning. In *The World Wide Web Conference*, WWW '19, page 3455–3461, New York, NY, USA. Association for Computing Machinery.

[350] Chujie Zheng, Kunpeng Zhang, Harry Jiannan Wang, and Ling Fan. 2020. A two-phase approach for abstractive podcast summarization. *arXiv preprint arXiv:2011.08291*.

[351] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*.

[352] Ming Zhong, Pengfei Liu, Yiran Chen, Danqing Wang, Xipeng Qiu, and Xuanjing Huang. 2020. Extractive summarization as text matching. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6197–6208, Online. Association for Computational Linguistics.

[353] Ming Zhong, Pengfei Liu, Danqing Wang, Xipeng Qiu, and Xuanjing Huang. 2019. Searching for effective neural extractive summarization: What works and what's next. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1049–1058, Florence, Italy. Association for Computational Linguistics.

[354] Ming Zhong, Yang Liu, Da Yin, Yuning Mao, Yizhu Jiao, Pengfei Liu, Chenguang Zhu, Heng Ji, and Jiawei Han. 2022. Towards a unified multi-dimensional evaluator for text generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2023–2038, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

[355] Wanjun Zhong, Jingjing Xu, Duyu Tang, Zenan Xu, Nan Duan, Ming Zhou, Jiahai Wang, and Jian Yin. 2020. Reasoning over semantic-level graph for fact checking. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6170–6180, Online. Association for Computational Linguistics.

[356] Chunting Zhou, Graham Neubig, Jiatao Gu, Mona Diab, Francisco Guzmán, Luke Zettlemoyer, and Marjan Ghazvininejad. 2021. Detecting hallucinated content in conditional neural sequence generation. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1393–1404, Online. Association for Computational Linguistics.

[357] Dawei Zhu, Nan Yang, Liang Wang, Yifan Song, Wenhao Wu, Furu Wei, and Sujian Li. 2024. PoSE: Efficient context window extension of LLMs via positional skip-wise training. In *The Twelfth International Conference on Learning Representations*.

[358] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2020. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76.

[359] Máté Ákos Tündik, Valér Kaszás, and György Szaszák. 2019. Assessing the Semantic Space Bias Caused by ASR Error Propagation and its Effect on Spoken Document Summarization. In *Proc. Interspeech 2019*, pages 1333–1337.

Appendix A

Podcast Summary Assessment (PSA)

The podcast summary assessment (PSA) dataset is a collection of podcast summaries generated by summarization systems at the Spotify Podcast Challenge at TREC 2020 [125]. The dataset consists of 179 podcast episodes (i.e., source documents). All episodes have summaries from 19 summarization systems and 1 creator-provided description. The human evaluation was performed by assessors at the National Institute of Standards and Technology (NIST) for the TREC 2020 challenge, resulting in 3580 annotated document-summary pairs in total. The evaluated summarization systems [125, 350, 184, 274, 219, 129] can be categorized into:

- **Reference**[1] = R1
- **Extractive systems** = E1, E2, E3
- **Abstractive systems** = A1, A2, A3,..., A16.

where the extractive summarization systems are based on TextRank [198], while abstractive summarization systems use a form of deep learning and pre-trained sequence-to-sequence models including BART [159] and T5 [244]. We note that the system descriptions are anonymized (as labelled by E1, E2, A1, etc) due to a data permission restriction. Nonetheless, without the summary generation system identity, the dataset can be used for evaluating summary assessment methods. Table A.1 shows the length statistics, and compared to SummEval (in Table 7.2) the documents in PSA are more than 10$\times$ longer, which makes this dataset challenging.

[1] Creator-provided description has been used as the reference summary in training summarization systems. It is worth noting that less than half of the descriptions were graded with the highest score (see Figure A.1b).

	#sentences	#words
Transcript	303±258	5950±5092
Summary	5.9±9.2	88.3±75

Table A.1 Length of Podcast Summary Assessment (Avg.±Std.) based on the NLTK tokenizer.

NIST Manual Evaluation

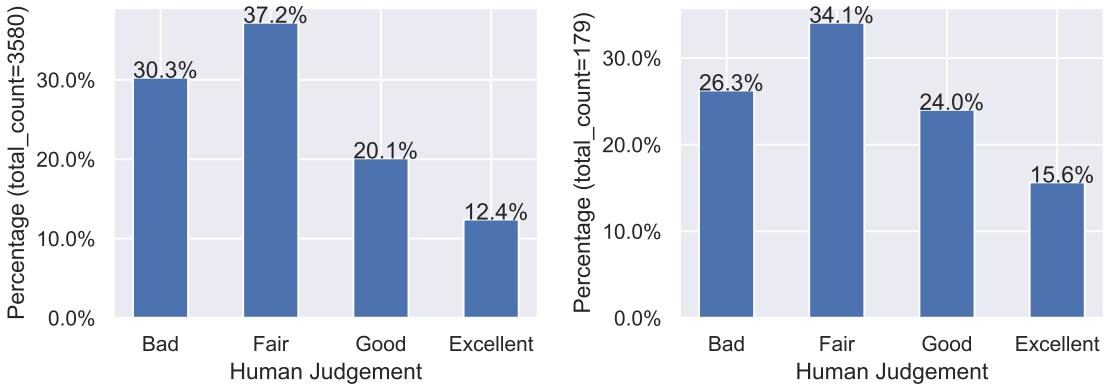
The summaries were judged by NIST assessors on a 4-point scale (Excellent/Good/Fair/Bad) considering both *content* and *text quality* as per the instructions (as described in Jones et al. [125]) as follows:

- **Excellent**: (*Content*) the summary accurately conveys all the most important attributes of the episode, which could include topical content, genre, and participants. In addition to giving an accurate representation of the content, it contains almost no redundant material which is not needed when deciding whether to listen. (*Text Quality*) It is also coherent, comprehensible, and has no grammatical errors.
- **Good**: (*Content*) the summary conveys most of the most important attributes and gives the reader a reasonable sense of what the episode contains with little redundant material which is not needed when deciding whether to listen. (*Text Quality*) Occasional grammatical or coherence errors are acceptable.
- **Fair**: (*Content*) the summary conveys some attributes of the content but gives the reader an imperfect or incomplete sense of what the episode contains. It may contain redundant material which is not needed when deciding whether to listen. (*Text Quality*) It may contain repetitions or broken sentences.
- **Bad**: (*Content*) the summary does not convey any of the most important content items of the episode or gives the reader an incorrect or incomprehensible sense of what the episode contains. It may contain a large amount of redundant information that is not needed when deciding whether to listen to the episode. If the content quality is considered bad, the summary is considered bad regardless of the text quality.

Figure A.1 shows the distribution of human judgements using the criteria above. It can be seen that around a quarter of creator-provided descriptions are graded *Bad*. This finding indicates that the training data for summarization can be quite noisy, and a potential way to

improve the summarization system development could involve filtering the training data to a smaller subset based on the scores predicted by an assessment system.[2]

Additionally, 8 binary attributes for each summary are included in addition to the overall scores. The annotators labelled 8 binary attributes by answering binary questions (i.e., answering 'yes' or 'no'): Q1) Are the names of the main people included?; Q2) Is there any additional information about the people mentioned?; Q3) Is the main topic included?; Q4) Is the format of the podcast mentioned?; Q5) Is there any context on the title?; Q6) Is there redundant information?; Q7) Is the summary in good written English? Q8) are the start and end points good?



(a) All Systems **(b)** Creator Description

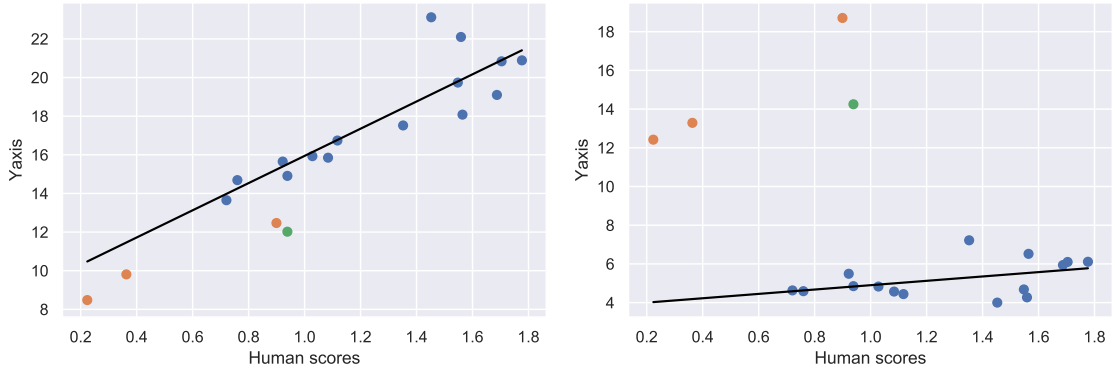
Fig. A.1 The distribution of human judgements in the PSA dataset.

Benchmarking Podcast Summary Assessment

Here, we provide a benchmark for existing summary assessment baselines on the new dataset. The summary assessment baselines are described in the main text of this thesis (in Section 4.3). In Table A.2, the baselines are categorized by whether they are reference-based (i.e., comparing against reference) or reference-free (i.e., comparing against document); unsupervised or supervised where supervised methods require gold-standard scores (i.e., human judgements). In total, there are 19 summarization systems evaluated ($M = 19$) excluding the creator description. However, given the nature of these summarization systems (e.g., some are extractive, while the rest are abstractive) extractive summaries by default will have high overlapping scores such as ROUGE-L against their source documents. This observation is, for example, shown in Figure A.2. As a result, we consider two sets of summarization systems: (1) 19-system where all summarization systems are included;

[2]Sentence filtering introduced in Section 6.2 reduced the length of the input sources, while this data filtering reduces the number of training instances.

(2) 15-system where the three extractive systems and vanilla BART-large fine-tuned to CNN/DailyMail (which is highly extractive) are excluded.



(a) ROUGE-L (summary, reference) **(b)** ROUGE-L (summary, document)

Fig. A.2 Scatter plots and best-fitted lines on abstractive systems. Blue = abstractive systems, Orange = extractive systems, Green = BART-large-cnn.

Method	Against		Type	System-level		Summary-level	
	Ref	Doc		19-sys	15-sys	19-sys	15-sys
ROUGE-L	✓		Unsupervised	0.905	0.864	0.350	0.246
TripleMatch	✓		Unsupervised	0.838	0.746	0.079	0.052
BERTScore	✓		Unsupervised	0.826	0.739	0.197	0.251
ROUGE-L		✓	Unsupervised	-0.200	0.364	-0.036	0.250
TripleMatch		✓	Unsupervised	-0.159	0.453	-0.123	0.143
BERTScore		✓	Unsupervised	-0.128	0.686	0.118	0.219
SpanQAG [B-512]		✓	Unsupervised	-0.112	0.517	-0.045	0.123
SpanQAG [L-4096]		✓	Unsupervised	-0.115	0.503	-0.071	0.118
Entailment [B-512]		✓	Unsupervised	0.356	0.114	0.102	0.021
Entailment [L-4096]		✓	Unsupervised	-0.192	0.392	-0.105	-0.059
CNN model		✓	Weakly Supervised	0.728	0.563	0.171	0.019
CNN model		✓	Supervised	0.901	0.902	0.299	0.183
BERT model		✓	Supervised	0.905	0.869	0.237	0.156
Longformer model		✓	Supervised	0.909	0.896	0.278	0.196

Table A.2 Spearman correlation coefficients (19 systems – excluding creator description). Inc./Exc. = Including/Excluding extractive summaries.