

Hanoi University of Science and Technology

Semester: 2024.2

Course: Project II

Supervisor: Ths. Le Duc Trung



English Text Summarization Using Deep Learning Methods

Name	Doan Anh Vu
Program	IT-E10 - Data Science & AI
Student's ID	20225465
Class ID	750647

Contents

1	Introduction	1
2	Dataset	1
3	Methods and Models	4
3.1	Preprocessing and Tokenization	4
3.1.1	Simple Tokenization	4
3.1.2	Modern tokenization	6
3.2	Models	8
3.2.1	Custom Transformer	9
3.2.2	T5-small	10
3.2.3	BART-base	11
3.2.4	Pegasus-custom	12
3.2.5	Model Ensemble via ROUGE-based Selection	13
4	Experimental Setup	13
5	Evaluation and Result	15
6	Challenges	18
7	Conclusion	19

1 Introduction

Text summarization is an important task in the field of Natural Language Processing (NLP), with the goal of generating concise summaries that still retain the core content of the original text. This technique is particularly useful in the context of the growing volume of information, helping users save time in reading and grasping key ideas. There are two main types of text summarization: **extractive summarization** and **abstractive summarization**, according to [3].

Extractive summarization: This method selects sentences or segments directly from the original text and combines them to create a summary. Extractive methods typically rely on algorithms that assess the importance of each sentence or segment in the text.

Abstractive summarization: In contrast, abstractive summarization generates a new summary by rephrasing the content of the original text in a more concise form, using new words and sentence structures. This method is more complex and requires the model to deeply understand the text's content.

This project focuses on **abstractive summarization**, with the goal of generating summaries that not only extract key sentences from the original text but also restructure the information, creating more natural, concise, and coherent summaries.

The objective of this project is to build and compare the effectiveness of deep learning models for English text summarization. Initially, I experimented with a self-implemented sequence-to-sequence (seq2seq) model using the **Transformer** architecture as a foundation, in order to understand the end-to-end encoding-decoding process. Then, I fine-tuned three pre-trained large language models **T5-small**, **Pegasus**, and **BART-base** on a specific summarization dataset.

Each model offers its own strengths: **T5** provides high flexibility in input-output formatting; **Pegasus** is specifically designed for summarization tasks; and **Bart** combines the characteristics of both autoencoding and autoregressive models, resulting in more fluent and accurate outputs [6]. Through experimentation and evaluation, I compare the summarization quality across these models to draw conclusions about their effectiveness, accuracy, and suitability for different use cases.

2 Dataset

In this project, I utilized a preprocessed version of the CNN/DailyMail dataset [2]. The dataset originally consists of three columns: `id`, `article`, and `highlights`. For consistency and ease of use, I renamed `article` to `articles` and `highlights` to `summaries`.

The original dataset was divided into three subsets as follows:

- **Training set:** 287,113 samples
- **Validation set:** 13,368 samples
- **Test set:** 11,490 samples

Each sample consists of a full news article paired with its corresponding abstractive summary. The large scale and diversity of the dataset make it well-suited for training and evaluating deep learning models for text summarization.

The length statistics and distributions of the articles and summaries before filtering are illustrated in Figures 1 to 4.

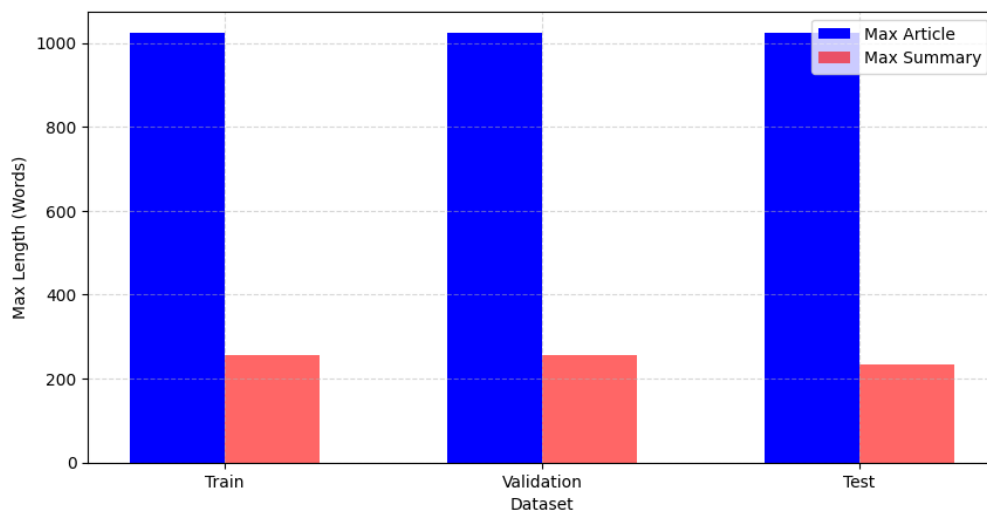


Figure 1: Maximum lengths of articles and summaries

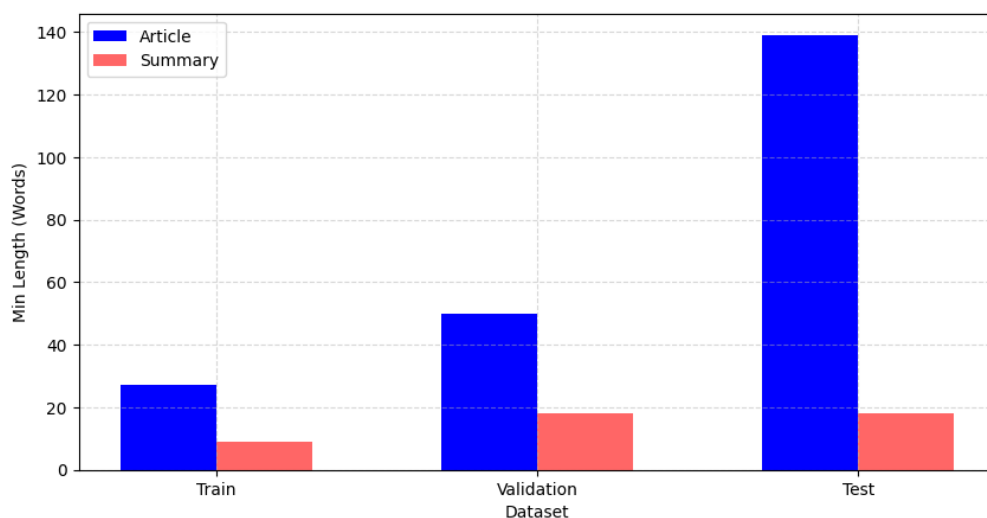


Figure 2: Minimum lengths of articles and summaries

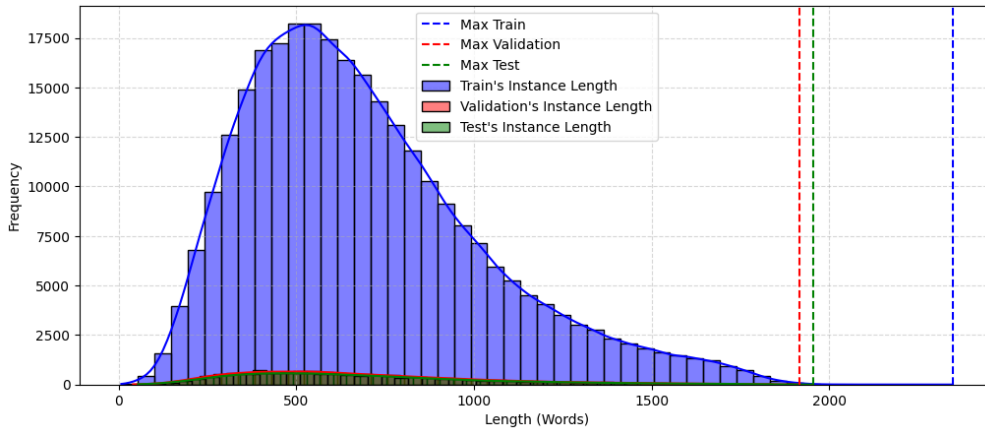


Figure 3: Distribution of article lengths

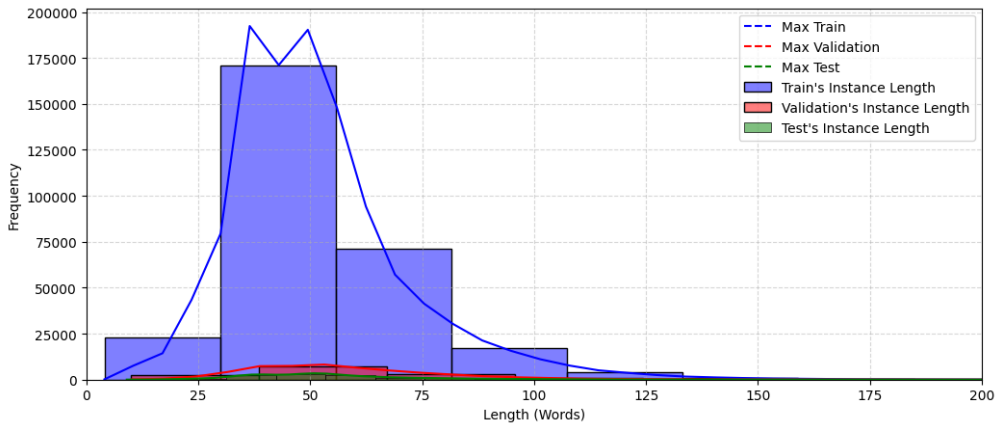


Figure 4: Distribution of summary lengths

To reduce training time while preserving dataset diversity, I applied filtering based on the lengths of the articles and summaries. Specifically, I selected only entries where articles contain a maximum of 512 words, and summaries 128 words. After filtering, the final dataset was significantly smaller, but still sufficient for fine-tuning and evaluation:

- **Training set:** 19,198 samples
- **Validation set:** 994 samples
- **Test set:** 4224 samples

3 Methods and Models

3.1 Preprocessing and Tokenization

3.1.1 Simple Tokenization

For a custom sequence-to-sequence (seq2seq) model from scratch, The input preparation process for this model involved several stages:

Tokenization I used a simple whitespace-based tokenizer by applying Python's `split()` function. In addition, I manually added the following special tokens:

- **<PAD>**: Padding token, used to ensure all sequences in a batch have the same length.
- **<SOS>**: Start-of-sequence token, added at the beginning of the decoder input sequence.
- **<EOS>**: End-of-sequence token, used as a signal to stop decoding.
- **<UNK>**: Unknown token, used to represent words not present in the vocabulary.

Word Embedding To represent each token in a dense vector space, I experimented with two pre-trained GloVe embeddings: `glove.6B.50d` [8] and `glove-wiki-gigaword-100`: an older 100-dimensional version of GloVe trained on Wikipedia and Gigaword. This version was obtained from a prior release and may no longer be publicly available.

Each word in the input text was mapped to its corresponding GloVe vector. For words not present in the embedding dictionary, vectors were randomly initialized from a uniform distribution. The resulting embedding matrix was used to initialize the embedding layers of both the encoder and decoder.

Vocabulary Coverage Analysis To assess the suitability of the embeddings, I measured the vocabulary coverage rate—defined as the percentage of unique words in the dataset that are present in the embedding dictionary—for each subset of the data, for `glove-wiki-gigaword-100`:

- **Training set coverage rate:** 5.85%
- **Validation set coverage rate:** 24.56%
- **Test set coverage rate:** 14.18%

Although the `glove.6B.50d` embedding has 400,000 words, the results are still disappointing. With such low coverage rates, even large or complex models will inevitably face difficulties in learning meaningful representations and generating coherent summaries.

- **Training set coverage rate:** 15.59%
- **Validation set coverage rate:** 33.25%
- **Test set coverage rate:** 24.57%

Upon examining the words that were not present in the embedding dictionary, I observed that many of them were either abbreviations, inflected forms, or tokens with special characters directly attached—for example: "Ariel,", *issue*, *out.Small*, *can't*, or variations in apostrophe symbols such as *'mainly*. This highlights that simple whitespace-based tokenization (`split()`) is insufficient for robust preprocessing.

Preprocessing To address this issue, I adopted a more advanced tokenization method using the Natural Language Toolkit (NLTK) library - a more sophisticated tokenization strategy using the `TreebankWordTokenizer` provided by the Natural Language Toolkit (NLTK). Unlike simple whitespace-based splitting, this tokenizer applies a set of linguistically informed rules based on the Penn Treebank annotation guidelines [8]. It handles contractions (e.g., "*can't*" → "*ca*", "*n't*"), separates punctuation (e.g., "*issue,*" → "*issue*", ","), and avoids attaching special characters directly to tokens.

I chose not to convert the text to lowercase because retaining the case allows distinguishing between words with different meanings, such as "Apple" (the company) and "apple" (the fruit). This preserves important semantic information and enhances the model's ability to generalize.

However, certain edge cases remain unresolved. For example, special characters are sometimes attached directly to words, such as in *—but* or *'Skinny*, which are not consistently separated by tokenization. Attempts to remove such characters risk altering meaningful content, especially in cases where punctuation is embedded in URLs or domain-like strings,... such as *eredivisie.oakland* or *Australiana-themed*.

Furthermore, tokens such as *child.Royal* illustrate another limitation where a punctuation mark (e.g., a period) incorrectly merges two words. These cases are particularly challenging because naive separation may lead to the loss of semantic or structural information, especially when dealing with web addresses, abbreviations, or proper nouns. Thus, I opted not to apply overly aggressive preprocessing rules in order to preserve critical information in the dataset.

As a result on, the quality and consistency of word representations are enhanced, but still low.

- **Training set coverage rate:** 31.02%
- **Validation set coverage rate:** 53.84%
- **Test set coverage rate:** 43.60%

3.1.2 Modern tokenization

Modern neural language models no longer rely on traditional word-level tokenization, which is prone to out-of-vocabulary (OOV) issues and suffers from poor generalization to rare or morphologically rich words. Instead, they adopt **subword-level** tokenization strategies, enabling them to handle arbitrary input text without fixed vocabularies.

Three popular subword tokenization methods are used in current models each has own vocabulary training algorithm and encoding scheme:

- **WordPiece**

WordPiece tokenizes text based on a greedy longest-match-first algorithm over a fixed vocabulary of subwords. It requires input to be segmented by whitespace.



Figure 5: Word Piece vs Split on whitespaces tokenization [11]

- **Byte-Pair Encoding (BPE)**

BPE is a frequency-based compression algorithm. Figure 6 and 10 show the way BPE works. In training phase, it merges the most frequent adjacent pairs of characters or tokens to build subwords iteratively.

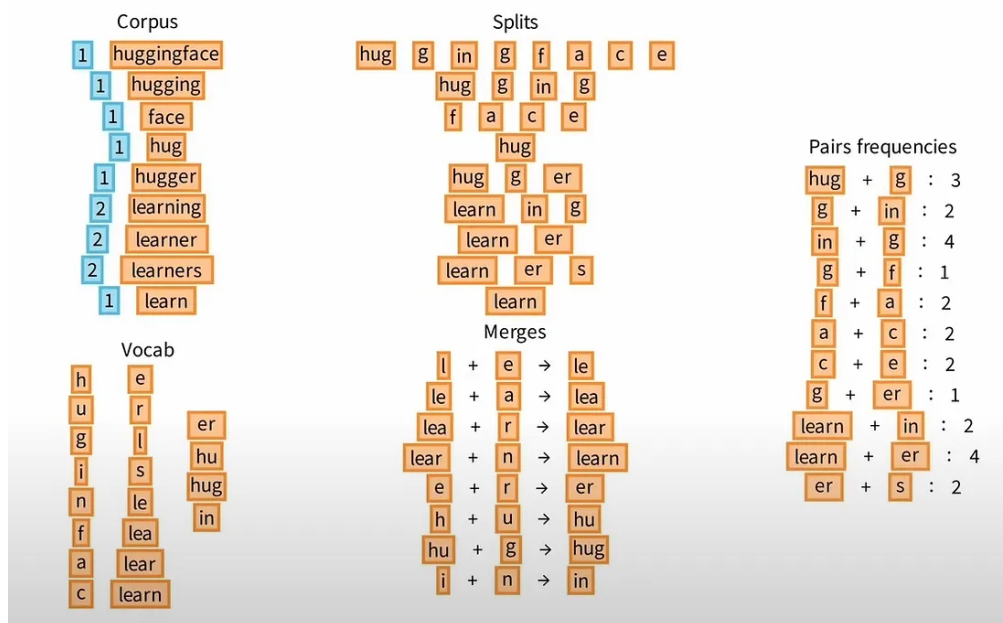


Figure 6: BPE training[9]

After training, BPE produces a fixed vocabulary of subword units. When encoding a new input sequence, the algorithm greedily applies the longest possible merges from this vocabulary to segment the input into subword tokens.

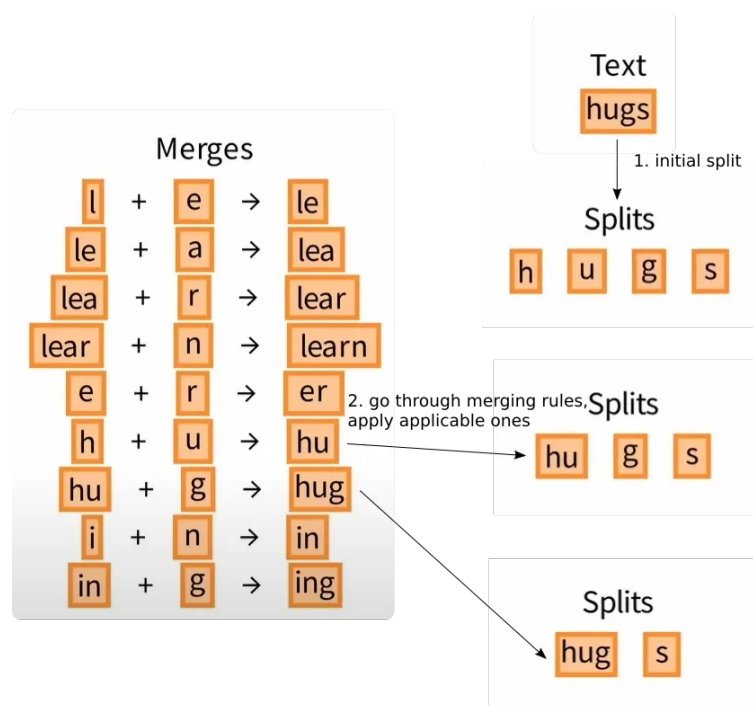



Figure 7: Tokenize word "hugs" by trained BPE [9]

- **SentencePiece**

SentencePiece, operates directly on raw text without requiring whitespace-based tokenization.

It treats the input as a stream of characters and learns subwords using either BPE or a probabilistic unigram language model.

$$P_{\text{train}}(\text{unigram}) = \frac{n_{\text{train}}(\text{unigram}) + k}{N_{\text{train}} + kV}$$



 unigram vocabulary size
 (number of unique unigrams in training text)

Figure 8: Unigram with Laplace smoothing [7]

In this work, the three fine-tuned models—`t5-small`, `pegasus`, and `bart-base`—use subword tokenization tailored to their architecture:

- `Bart-base`: uses Byte-Pair Encoding (BPE), a merge-based tokenization algorithm that iteratively replaces the most frequent pair of symbols with a new token.
- `t5-small` and `Pegasus`: both use SentencePiece with a unigram language model, which learns a probabilistic model over subword units and selects the most likely segmentation.

3.2 Models

In my exploration of effective architectures for *Text Summarization*, I chose to exclude traditional Recurrent Neural Networks (RNNs). RNNs suffer from the **vanishing gradient problem** and inefficient sequential processing. While LSTMs and GRUs offer improvements, they still have limitations in scalability and contextual understanding.

Therefore, I opted to begin directly with the **Transformer** architecture from the *Attention is All You Need* [12] paper. The Transformer introduces a **self-attention mechanism**, enabling the model to attend to all input positions simultaneously, effectively capturing global dependencies and enhancing deeper contextual modeling.

The model takes raw text documents as input and learns to generate concise summaries that preserve the key information. The encoder processes tokenized input text and transforms it into contextualized representations. For the decoding stage, the Transformer generates summary tokens in parallel, unlike recurrent models that inherently maintain a sense of sequence order. To compensate for this lack of order awareness, positional encodings are added to the token embeddings. The positional encoding used sinusoidal functions similar to the one implemented in [12], which is defined as

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

where:

- pos is the position in the sequence.
- i is the dimension index.
- d_{model} is the dimensionality of the embedding vector.

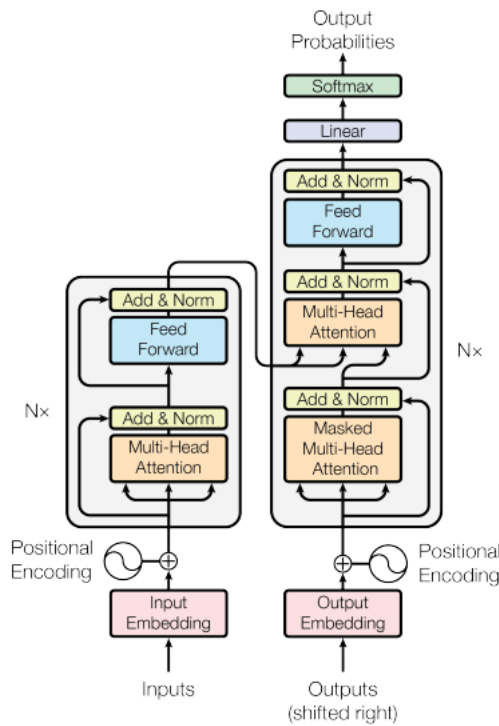


Figure 9: Original transformer encoder-decoder architecture [12]

3.2.1 Custom Transformer

Custom Transformer was implemented from scratch following the standard encoder-decoder architecture for text summarization. This implementation combines self-attention mechanisms with positional encoding to capture document context effectively. The architecture consists of separate encoder and decoder components:

- The encoder processes input documents through multi-head self-attention layers
- The decoder generates summaries using both self-attention and cross-attention to the encoded input
- Positional encoding provides sequence order information without recurrence

For summary generation, two decoding strategies were implemented:

- **Greedy Search:** Selects the highest probability token at each step
- **Beam Search:** Maintains multiple candidate sequences during generation

The model was trained using teacher forcing with cross-entropy loss. Implementation details include 3 encoder and decoder layers, 8 attention heads, 512-dimensional feed-forward networks, and 0.1 dropout rate for regularization. While lacking the benefit of pre-training on large corpora, this custom implementation offers insights into transformer mechanics for summarization and we can dive into the real architecture leveraged transformer to understand it.

3.2.2 T5-small

T5-small, a pretrained encoder-decoder model introduced in the *Text-To-Text Transfer Transformer (T5)* framework by [10]. The T5 model reformulates every NLP task into a text-to-text format, making it particularly suitable for abstractive summarization.

Compared to the original Transformer architecture, T5 integrates several enhancements:

- It is pretrained on a large corpus (C4 dataset) using a masked span denoising objective.
- It uses layer normalization before each sub-layer (pre-norm) and a modified positional encoding scheme.

I opted for the **T5-small** variant due to its computational efficiency, making it suitable for training and inference on limited hardware resources. During fine-tuning, I formatted the input with

```
"summarize: <input-text>"
```

exactly like Figure 6:

The prefix "summarize:" serves as an explicit instruction that guides the model to perform abstractive summarization. Unlike task-specific models like BART or PEGASUS, T5 uses a unified text-to-text framework, where the model interprets the prefix as a signal to determine the nature of the task. If the prefix is missing or altered (e.g., mistyped as "summary:"), the model may not perform summarization correctly, as it has not been trained on such variations.

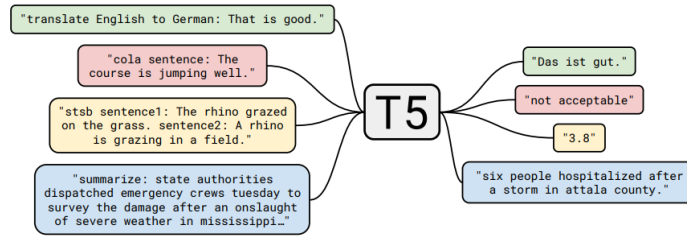


Figure 10: Standard prefix for T5's input [10]

Workflow. The fine-tuning process follows a standard encoder-decoder sequence-to-sequence structure:

1. The input text is prefixed with "summarize:" and tokenized.
2. The encoder transforms the prefixed input sequence into contextual embeddings using self-attention and feed-forward layers with relative positional bias.
3. The decoder, conditioned on the encoder output and previously generated tokens, autoregressively generates the summary sequence token by token.

3.2.3 BART-base

BART-base, proposed by [4], is a denoising autoencoder for pretraining sequence-to-sequence models. BART adopts a standard Transformer architecture but with key distinctions:

- The encoder is fully bidirectional, similar to BERT, enabling it to access the full context of the input.
- The decoder is autoregressive, like GPT, generating output one token at a time.
- BART is pretrained using a denoising objective, such as text infilling and sentence permutation, allowing it to learn robust representations for downstream tasks.

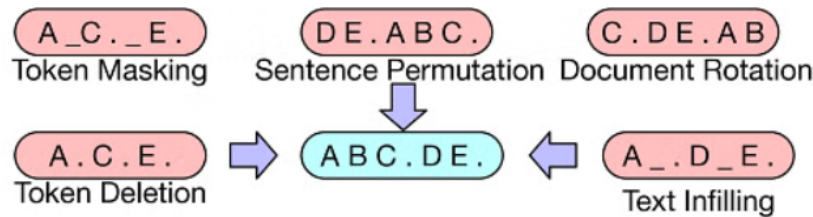


Figure 11: Transformations for noising the input [4]

Compared to the vanilla Transformer:

- BART applies masking and corruption strategies during pretraining, whereas the original Transformer does not include pretraining.
- The encoder in BART is bidirectional (like BERT), while the standard Transformer’s encoder is typically left-to-right in generative setups.
- The model is trained end-to-end to reconstruct corrupted inputs, enhancing performance on summarization tasks.

I used the **BART-base** variant for fine-tuning on text summarization. It offers a good trade-off between performance and computational efficiency. Input documents are tokenized using a byte-level **BPE tokenizer** and passed to the encoder. The decoder then autoregressively generates the summary, conditioned on the encoder’s hidden states and previously generated tokens.

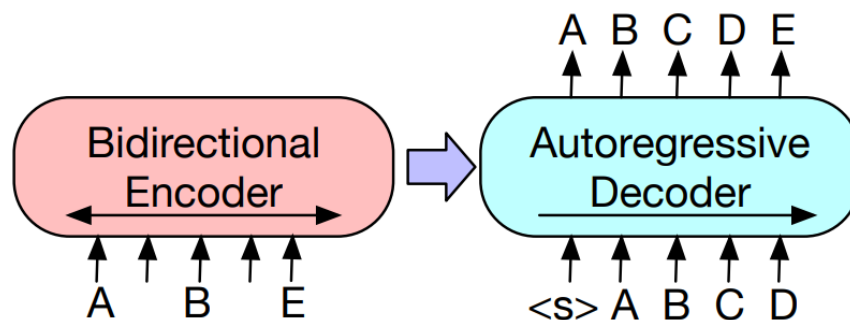


Figure 12: BART architecture combining a BERT-style encoder and GPT-style decoder [4]

3.2.4 Pegasus-custom

PEGASUS (Pre-training with Extracted Gap-sentences for Abstractive Summarization), proposed by [13], is a Transformer-based encoder-decoder model specifically designed for abstractive text summarization. Unlike generic pretraining objectives, PEGASUS uses a novel gap-sentence generation (GSG) task, where important sentences are masked and the model is trained to generate them.

Due to the high computational demands of the original pegasus-large model (16 layers, 1024 hidden units), I implemented a **custom reduced-size variant** named Pegasus-custom. The reduction was applied to make the model suitable for training and inference on limited hardware (e.g., single-GPU environments) without significantly compromising performance.

The key configuration changes include:

- Reduced encoder and decoder layers from 16 to 6.
- Decreased hidden size from 1024 to 512.

- Reduced number of attention heads to 8.
- Lowered feedforward dimension from 4096 to 2048.

The model was initialized using: `PegasusTokenizer.from_pretrained('google/pegasus-large')` and a custom `PegasusConfig` matching the tokenizer’s vocabulary size (`vocab_size=96103`) to ensure compatibility.

3.2.5 Model Ensemble via ROUGE-based Selection

To further improve the summary quality, I implemented an ensemble approach that leverages the strengths of multiple pretrained and fine-tuned models. Instead of averaging their outputs or logits, I employed a model selection strategy based on ROUGE-L scores.

For a given input article, I first generate summaries independently using each model. Let S_{bart} , S_{t5} , and $S_{pegasus}$ denote these outputs. I then evaluate each candidate summary against the others using the ROUGE-L F1 score. Each summary receives a weighted average of its similarity to the other two, using pre-defined weights $\{w_{bart} = 0.40, w_{t5} = 0.50, w_{pegasus} = 0.10\}$ to reflect their individual performance contributions.

Formally, for a candidate summary S_i , the score is computed as:

$$\text{score}(S_i) = w_i \cdot \frac{1}{|C_i|} \sum_{S_j \in C_i} \text{ROUGE-L}(S_i, S_j)$$

where C_i is the set of other candidate summaries ($C_i = \{S_j \mid j \neq i\}$). The summary with the highest weighted score is selected as the final output.

Finally, I apply a simple post-processing step to remove duplicate sentences and truncate the summary to a maximum token length using sentence tokenization and re-encoding.

This method combines both diversity and robustness, while remaining computationally efficient at inference time.

4 Experimental Setup

Before training on the full dataset, we first experimented with a small subset of the data. We tested different learning rate schedulers to find the most effective one. Specifically, we explored three options: no learning rate scheduler, a linear scheduler with warmup, and a cosine annealing scheduler with warmup. The results shown in Figure ?? indicate that the cosine scheduler with warmup generally achieves better loss performance.

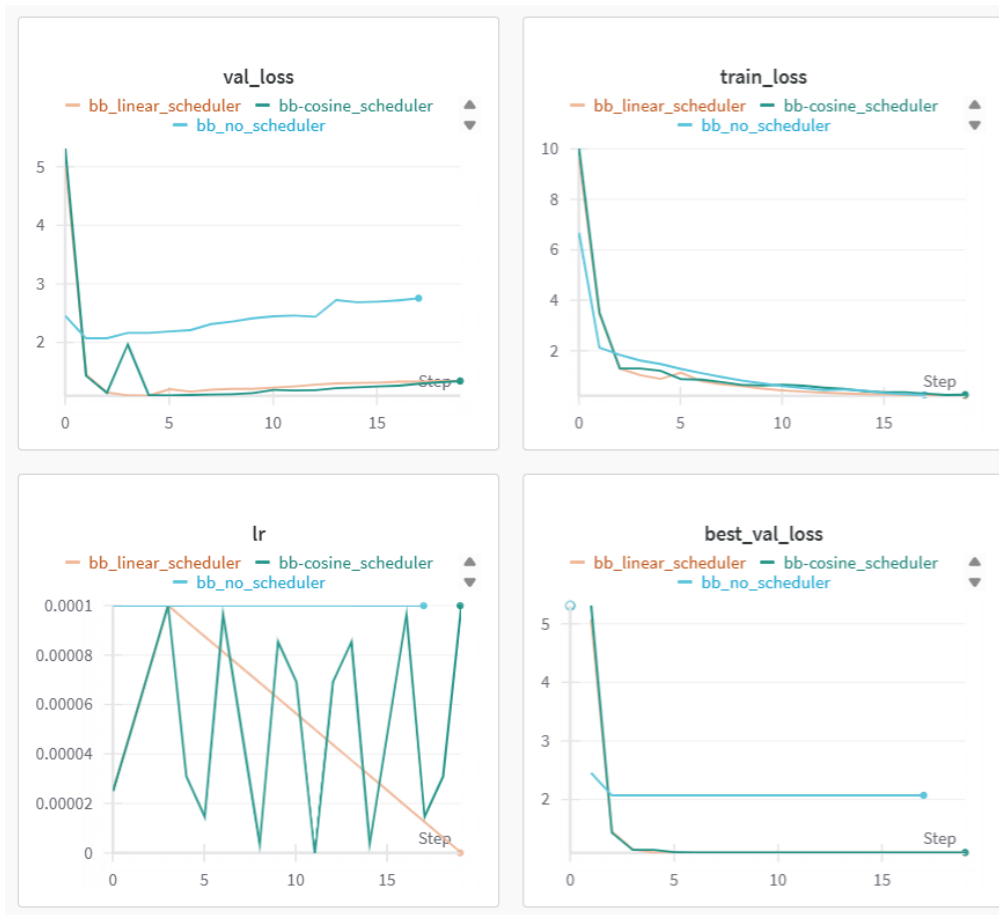


Figure 13: Three types of learning rate scheduler

Next, we tested the impact of the maximum learning rate on the training process. It is unsurprising that a learning rate that is too high causes the model to overfit quickly, while a rate that is too low prevents effective learning, leading to plateauing losses. Ultimately, a learning rate of 10^{-4} was selected as the optimal value.

We used common settings for training across all models. The cross-entropy loss function was used, which measured the difference between predicted token probabilities and ground-truth target tokens. The Adam optimizer was applied with a weight decay (L2 penalty) of $\lambda = 10^{-4}$. Training was primarily conducted on Kaggle, a cloud-based platform. Due to limited GPU resources, a small batch size (32) was used to avoid exceeding memory limits.

All four models were trained using the cross-entropy loss, a standard objective for sequence-to-sequence models. This loss function measures the difference between the predicted token probabilities and the true token distribution, effectively guiding the model to generate outputs that match the ground truth summaries token by token.

5 Evaluation and Result

After all, we get the chart from 4 models. As shown in Figure 14 and Figure 15, the best validation loss of the *Transformer (trained from scratch)* and the *PEGASUS (custom-trained)* models both remain under 5.0, around 4.7. However, such values are still not sufficient for high-quality text summarization tasks. This can be easily predicted because the models were significantly simplified in terms of architecture and trained for a relatively small number of epochs. Despite this, signs of overfitting were already observable, indicating that further training without regularization may not yield better generalization.

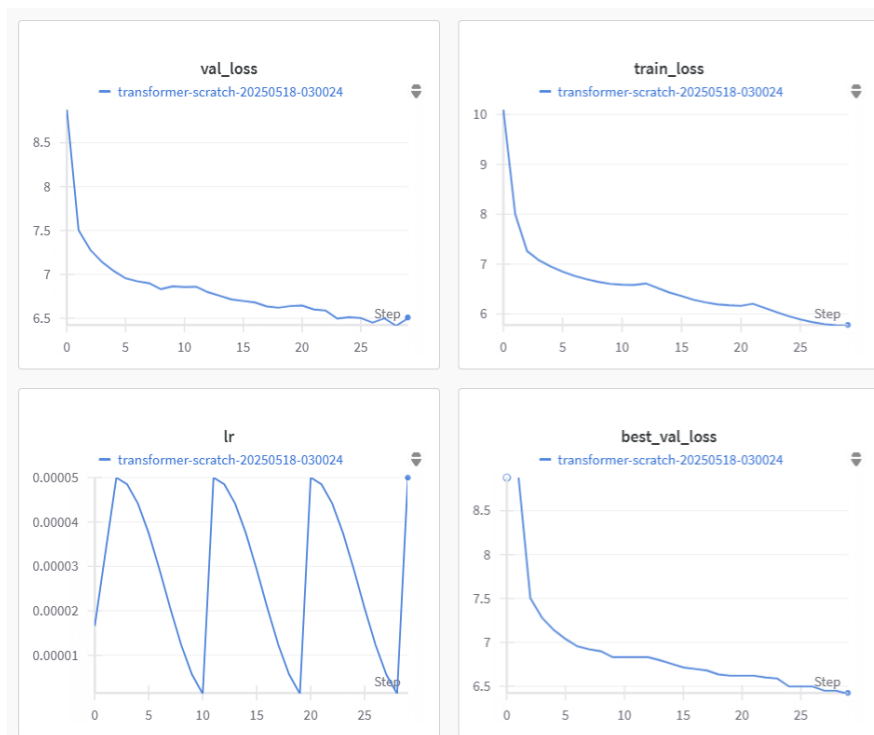


Figure 14: Transformer Custom 's charts

On the other hand, Figure 16 presents the validation loss comparison between *T5-small* and *BART-base*, both fine-tuned from pretrained checkpoints. Their best validation losses are similar, both falling below 1.0, with *BART-base* achieving slightly better performance. This result is expected, as *BART-base* is generally more complex than *T5-small* in terms of parameter count and model capacity.

The training loss and validation trends across the four models provide useful insights into how architectural complexity and pretraining influence summarization performance. While cross-entropy is effective for optimizing token-level accuracy during training, it does not directly capture broader qualities such as semantic coherence, informativeness, or fluency of the generated summaries.

In fact, relying solely on cross-entropy loss can be limiting, as a model might achieve a low loss but still produce outputs that are grammatically correct yet semantically misaligned or lacking in

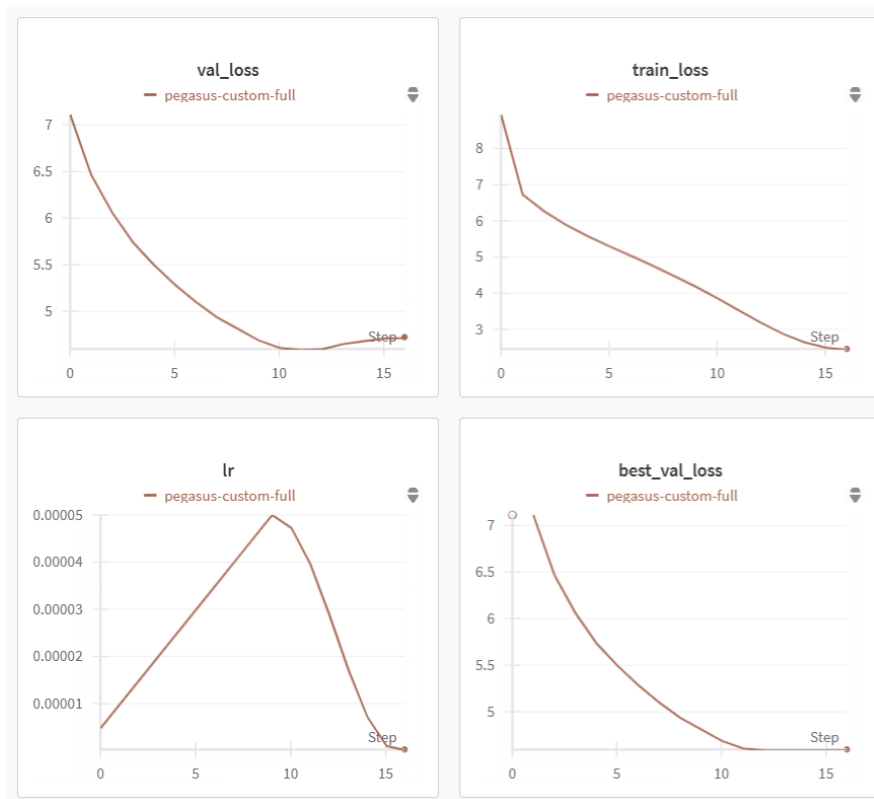


Figure 15: Pegasus Custom 's charts

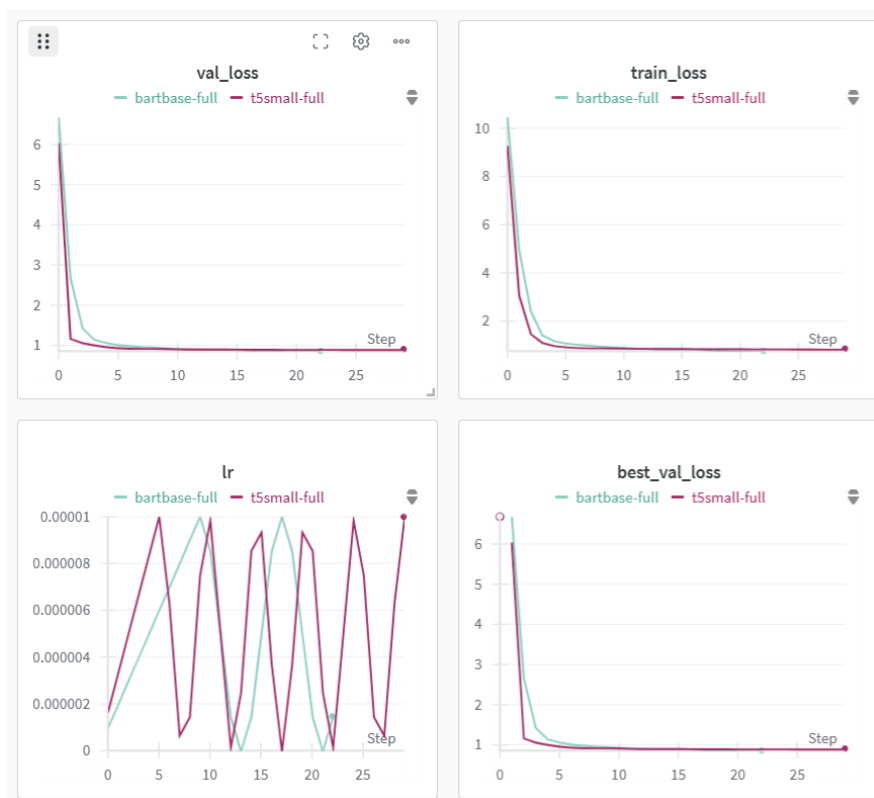


Figure 16: T5-small vs BART-base 's charts

essential information. This is especially true in abstractive summarization, where understanding and restructuring the input content is crucial. For actual quality assessment of the generated summaries, more robust and interpretable evaluation metrics are required.

To evaluate the quality of the generated summaries, I used a combination of lexical and semantic-based metrics. The evaluation was conducted on the test set (4500 samples), where each predicted summary was compared against the corresponding ground truth. Specifically, we measured performance using ROUGE, BERTScore, and METEOR:

- **ROUGE** [5] evaluates n-gram and longest common subsequence overlaps, and is a standard metric for summarization benchmarks.
- **BERTScore** [14] uses contextual embeddings from BERT to compute semantic similarity, and has shown higher correlation with human judgment in abstractive tasks.
- **METEOR** [1] considers synonym matching and stemming, originally developed for machine translation but often adapted for summarization evaluation.

Among the ROUGE metrics, I reported ROUGE-1, ROUGE-2, and ROUGE-L scores. Each of these captures a different aspect of summary quality: ROUGE-1 reflects unigram (word-level) overlap and is indicative of content coverage; ROUGE-2 measures bigram overlap, capturing fluency and local coherence; while ROUGE-L evaluates the longest common subsequence between prediction and reference, emphasizing the preservation of overall sentence structure and information flow.

While ROUGE remains the standard for benchmarking summarization models, I complement it with BERTScore to assess deeper semantic similarity beyond surface-level n-gram overlap. METEOR was included to account for synonymy and minor lexical variations, offering additional robustness to the evaluation.

The results are summarized in Table 1, where we compare the performance of different models. We report the F1 scores for ROUGE and BERTScore, which provide a balanced view between precision and recall.

Model	ROUGE-1	ROUGE-2	ROUGE-L	BERTScore (F1)	METEOR
Custom-Transformer	0.1815	0.0280	0.1741	0.8209	0.1318
Pegasus-custom	0.2576	0.0573	0.2415	0.8377	0.1820
T5-small	0.3928	0.1776	0.3711	0.8814	0.3149
BART-base	0.4061	0.1919	0.3845	0.8871	0.3803
Ensemble	0.4071	0.1925	0.3854	0.8816	0.3173

Table 1: Evaluation results on the CNN/DailyMail test set.

The **Custom Transformer**, trained from scratch without any pretraining, performs the weakest across all metrics. Its ROUGE-1 score of 0.1815 and ROUGE-2 of 0.0280 indicate poor content overlap with reference summaries. Additionally, its low BERTScore (0.8209) and METEOR (0.1318) confirm limited semantic and linguistic quality. These results are expected, given the model’s simple architecture, lack of pretraining, limited data, and short training time. While it serves as a useful baseline, it demonstrates the challenges of training summarization models from scratch.

The **Pegasus-custom** model achieves slightly better scores but still underperforms compared to pretrained alternatives. The pruning of encoder and decoder layers reduced computational demands but led to degraded performance. Its outputs often lack fluency and coherence, with a tendency to omit key information.

The **T5-small** model shows a strong improvement, with ROUGE-1 of 0.3928 and a BERTScore of 0.8814. Despite having fewer parameters than BART, its balanced architecture and full pretraining allow it to generate semantically accurate summaries. However, due to its smaller capacity, some outputs may still appear repetitive or lack stylistic richness.

The **BART-base** model achieves the highest overall scores, with ROUGE-1 at 0.4061, ROUGE-2 at 0.1919, and BERTScore of 0.8871. Its strong performance is due to both its higher capacity and its pretraining on the CNN/DailyMail dataset, which closely aligns with the current task. The model produces fluent, coherent, and information-rich summaries.

Finally, the simple **Ensemble** approach—averaging logits or predictions from T5-small, BART-base and Pegasus, yields marginal improvements, slightly surpassing BART in ROUGE metrics but trailing slightly in BERTScore. This suggests potential benefits from ensemble learning, though gains are modest without further optimization.

6 Challenges

Training large-scale transformer models poses significant computational challenges, especially under limited hardware conditions. All experiments in this project were conducted using a personal laptop with constrained GPU memory, which severely restricted the ability to utilize models with high parameter counts.

For example, **Pegasus-large**, with approximately 537 million parameters and a complex encoder-decoder architecture, was practically infeasible to train locally. A single epoch could take over 24 hours to complete. Similar constraints were observed with **BART-large** and larger variants of **T5**, which require significantly more memory and processing power than available.

To supplement local resources, I also utilized cloud platforms such as Kaggle, which offers free

access to GPUs. However, Kaggle imposes a weekly limit of 30 GPU hours per account, which still proved insufficient for comprehensive training and hyperparameter tuning of large models.

Due to these hardware and runtime limitations, the amount of training data had to be restricted. Although the selected subset retained key examples, the limited data volume reduced the model's exposure to diverse linguistic patterns. Moreover, input article lengths had to be capped at 512 tokens to prevent out-of-memory errors, even though real-world summarization tasks often require processing longer texts. To address this, I employed a chunking strategy in the demo phase, splitting long articles into smaller segments and summarizing each independently. While functional, this approach sacrifices global coherence and narrative flow compared to summarizing the full document in a single pass.

Finally, time constraints within the academic semester also posed a major challenge. With several concurrent courses and project deadlines, the time dedicated to this summarization project was limited. Given more time and access to high-performance hardware, I believe the results could be significantly improved, and more advanced models could be explored more thoroughly.

7 Conclusion

Through experimentation with models of varying size and complexity, I find that model architecture, pretraining, and fine-tuning strategies significantly affect summarization performance. Pretrained models like **BART-base** and **T5-small** clearly outperform simplified or custom-trained counterparts.

Among them, **BART-base** provides the best balance between fluency, accuracy, and informativeness—demonstrating the advantages of large-scale pretraining and domain alignment. Meanwhile, **T5-small** shows that smaller models can still be highly effective with proper fine-tuning. In contrast, models trained from scratch or aggressively pruned (like **Custom Transformer** and **Pegasus-custom**) highlight the limitations of reducing model capacity or skipping pretraining.

Moreover, an important yet often overlooked factor is the choice of *tokenizer*. Due to the variety of writing styles and complex word structures, simply splitting text by whitespace is not sufficiently accurate. Selecting an appropriate *tokenizer* significantly impacts the efficiency of processing and representing the input data. Tokenizers that are specifically designed for Vietnamese or trained on domain-relevant datasets typically yield noticeably better results. This further highlights the critical role of data preprocessing and the alignment between the *tokenizer*, the model, and the specific application domain.

These results reaffirm that, for abstractive summarization tasks, investing in pretrained trans-

former models remains the most effective strategy under current resource constraints.

All the code is in my GitHub repository <https://github.com/bluff-king/Text-Summarization>

References

- [1] S. Banerjee and A. Lavie. 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 MT and/or Summarization*, pages 65–72, 2005.
- [2] P. Gowrishankar. Newspaper text summarization - cnn/dailymail. <https://www.kaggle.com/datasets/gowrishankarp/newspaper-text-summarization-cnn-dailymail>, 2023. Accessed from Kaggle: <https://www.kaggle.com/datasets/gowrishankarp/newspaper-text-summarization-cnn-dailymail>.
- [3] M. Kirmani, G. Kaur, and M. Mohd. Analysis of abstractive and extractive summarization methods. *International Journal of Emerging Technologies in Learning (IJET)*, 19(01):86–96, January 2024. doi: 10.3991/ijet.v19i01.46079. URL <https://doi.org/10.3991/ijet.v19i01.46079>.
- [4] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019. doi: 10.48550/arXiv.1910.13461. URL <https://arxiv.org/abs/1910.13461>.
- [5] C.-Y. Lin. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81, 2004.
- [6] H. Nguyen, H. Chen, L. Pobbathi, and J. Ding. A comparative study of quality evaluation methods for text summarization, 2024. URL <https://arxiv.org/abs/2407.00747>. Submitted to EMNLP 2024.
- [7] K. Nguyen. N-gram language models. part 1: The unigram model, 2020. URL <https://medium.com/mti-technology/n-gram-language-model-b7c2fc322799>. Accessed: 2025-05-16.
- [8] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation (6b, 50d). <https://www.kaggle.com/datasets/watts2/glove6b50dtxt>, 2014. Pretrained on Wikipedia 2014 + Gigaword 5, 6B tokens, 400K vocab, 50d vectors. Accessed: 2025-05-14.
- [9] G. PIAO. What is bpe: Byte-pair encoding?, 2025. URL <https://medium.com/@parklize/what-is-bpe-byte-pair-encoding-5f1ea76ea01f>. Accessed: 2025-05-16.
- [10] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint*

- arXiv:1910.10683*, 2020. doi: 10.48550/arXiv.1910.10683. URL <https://arxiv.org/abs/1910.10683>.
- [11] X. Song and D. Zhou. A fast wordpiece tokenization system, 2021. URL <https://research.google/blog/a-fast-wordpiece-tokenization-system/>. Accessed: 2025-05-16.
- [12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017. doi: 10.48550/arXiv.1706.03762. URL <https://arxiv.org/abs/1706.03762>.
- [13] J. Zhang, Y. Zhao, M. Saleh, and P. J. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. *arXiv preprint arXiv:1912.08777*, 2020. doi: 10.48550/arXiv.1912.08777. URL <https://arxiv.org/abs/1912.08777>.
- [14] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations (ICLR)*, 2020.