|  |
| --- |
| **NLP(News title) Report** |

**Jincheng Jiang**

College of Engineering

Northeastern University

Toronto, ON

*Jiang.jinc@northeastern.edu*

**Abstract**

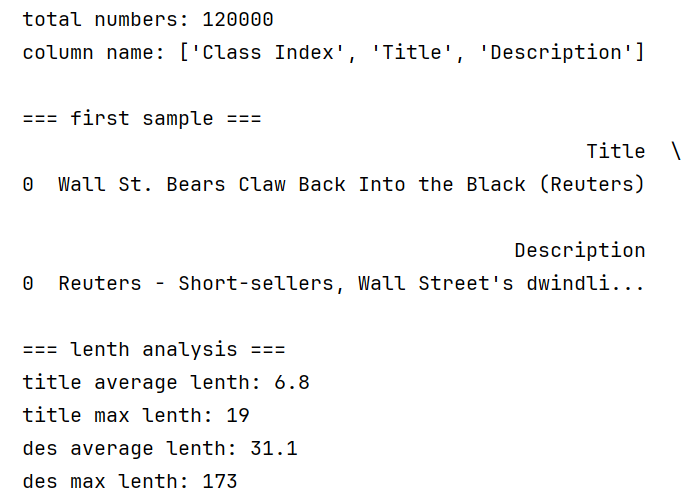
This study explored various deep-learning methods for automated news-title generation, beginning with a comprehensive data-processing pipeline using the AG’s News dataset. Initially, a robust vocabulary was constructed by employing tokenization strategies with spaCy, followed by meticulous data cleaning and normalization. After vocabulary preparation, exploratory N-gram analysis highlighted common linguistic patterns and informed the subsequent development of predictive language models. Multiple text-generation architectures, including GRU-based Seq2Seq, vanilla Transformers, T5-small, and BART, were implemented and fine-tuned on the prepared dataset. Evaluation across these models indicated significant performance differences, with transformer-based architectures, particularly BART—a hybrid leveraging BERT's bidirectional context modeling and GPT's autoregressive generation—demonstrating superior capability in capturing semantic context and generating coherent, contextually relevant news titles.

**1 Datasets**

<https://www.kaggle.com/datasets/amananandrai/ag-news-classification-dataset/data>

**1.2 Why is this an interesting dataset?**

The AG's news topic classification dataset is ideally suited for the news title summarization task due to its extensive, well-balanced, and richly labeled collection of news articles across four major topics. With 120,000 training samples and 7,600 testing samples, the dataset offers a robust foundation for training various text generation architectures such as RNNs, LSTMs, and Transformers, enabling them to capture the nuances of news content effectively. Additionally, the inclusion of both titles and detailed descriptions provides a natural setup for summarization, where models can learn to condense comprehensive news descriptions into engaging, concise headlines. This dataset not only facilitates rigorous performance evaluation but also presents an interesting challenge by requiring the generated summaries to be coherent, original, and fluent, while also highlighting the ethical considerations inherent in automated content generation.



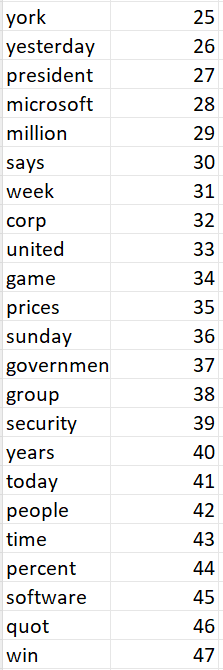
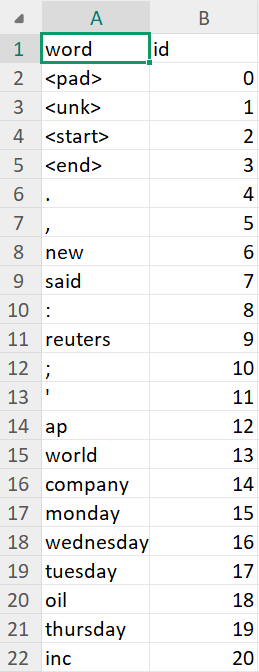
**2 Data preprocessing (details in preprocess.ipynb)**

**2.1 Data cleaning**

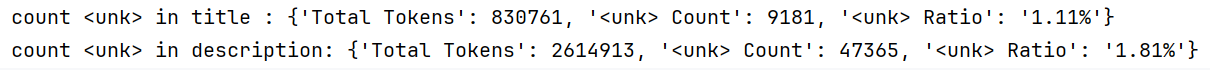
In the data cleaning process, I removed unwanted HTML tags and entities, then selectively retains only basic punctuation while standardizing and merging excess whitespace through the clean\_text function. Then I applied different normalization rules for titles and descriptions using normalize\_text: titles are converted to title case while preserving any existing all-caps words (such as acronyms), and descriptions are converted entirely to lowercase. Finally, the script filters out entries that fall below a specified word count threshold for both the normalized titles (fewer than three words) and descriptions (fewer than ten words), ensuring that only sufficiently informative text samples remain for subsequent modeling steps.

**2.2 Tokenization&Vocab**

I leveraged spaCy with GPU support and a lightweight English model (with the parser, NER, and lemmatizer disabled) to tokenize the text. The tokenization function converts the text to lowercase and then processes it token by token. It specifically retains only tokens that match a pattern of at least two characters consisting solely of lowercase letters and numbers, while also keeping certain punctuation marks (".", ",", "!", "?", ":", ";", "'"). Additionally, it removes spaCy’s built-in stop words and any tokens containing underscores, which helps in filtering out meaningless words. After tokenization, I aggregated tokens from both titles and descriptions, counted their frequency, and filtered out low-frequency tokens (with a minimum frequency of 3). The vocabulary is then constructed by sorting tokens by frequency and appending four special tokens (‘<pad>’, ‘<unk>’, ‘<start>’, ‘<end>’) to manage padding, unknown words, and sentence boundaries. Finally, I converted the tokenized text into numerical sequences using this vocabulary and saved the processed text data in Parquet format for efficient columnar storage and the vocabulary in CSV format for ease of portability and review.

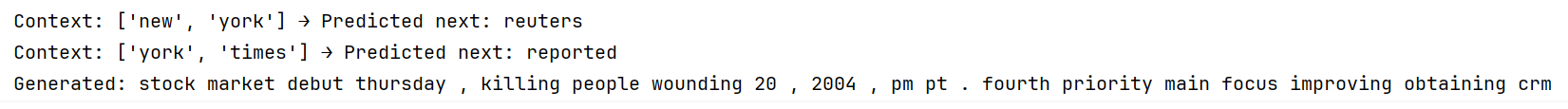


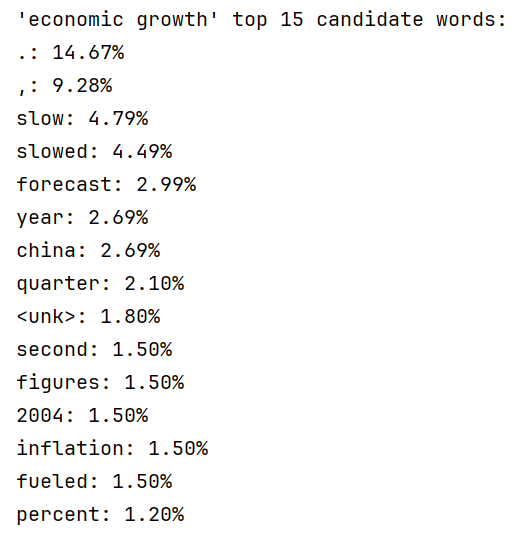
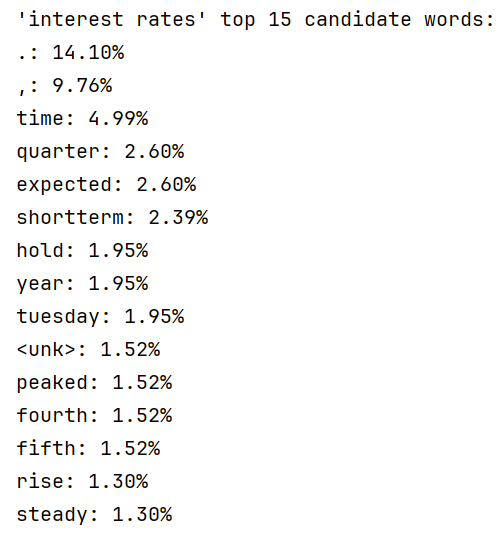
Additionally, the proportion of <unk> tokens in the titles is about 1.11%, and in the descriptions, it's about 1.81%. To avoid a higher <unk> ratio, I decided not to set the minimum frequency threshold to 3 or more, ensuring that the vocabulary remains sufficiently comprehensive and keeps the unknown token usage relatively low.

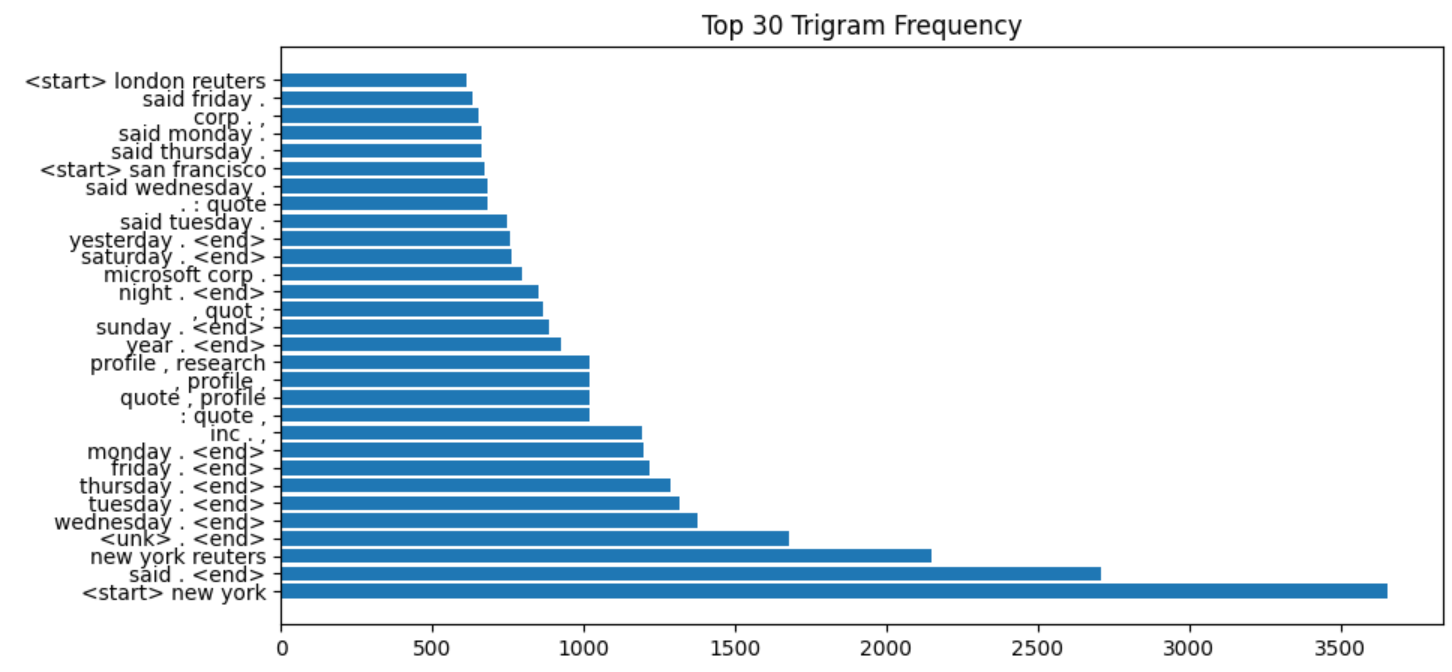


**3 Ngrams analysis (details in ngrams.ipynb)**

Applied a trigram approach, the model considers every pair of consecutive tokens as the context and then learns the likelihood of specific words appearing next. By aggregating counts for all such triplets in the training data, it can predict the most probable “next word” given a two-word context. The analysis of trigram frequencies reveals which three-token sequences are most prevalent—offering insights into the dataset’s dominant patterns, common expressions, and stylistic tendencies. For example, the bar chart highlighting the top 30 trigrams showcases frequently recurring phrases (such as specific day references or organization names) that characterize news headlines and descriptions. Furthermore, by examining the top candidate words following prompts like “interest rates” or “economic growth,” it becomes clear which terms the model deems most likely to occur next, often reflecting the typical language patterns around financial or economic topics in the corpus. Finally, generating text from this trigram model demonstrates its strengths—capturing local context well—but also illustrates some of the limitations inherent in n-gram approaches, such as producing repetitive or disjointed sequences when dealing with longer passages.

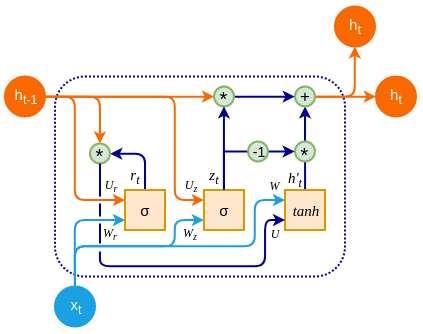
****

****

****

**4 Model structure and performance**

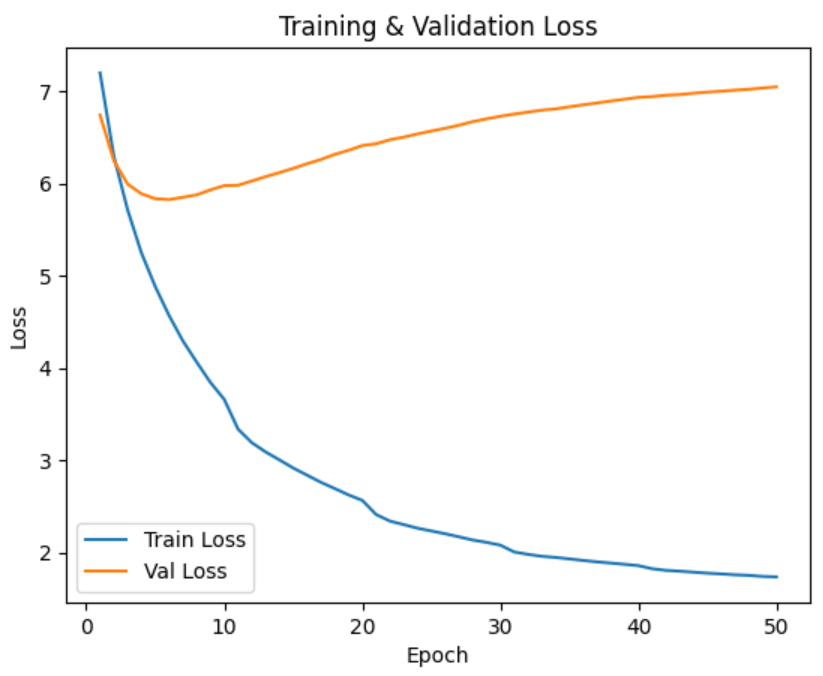
**4.1 Gru (details in GRU.ipynb)**



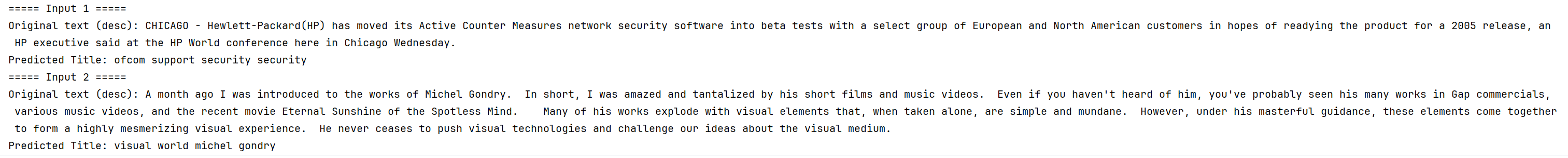
The GRU-based architecture is implemented as a sequence-to-sequence model designed to predict news titles based on their corresponding descriptions. The pipeline starts with an Encoder that first converts each token into a dense embedding vector using an embedding layer, which helps capture semantic relationships in a lower-dimensional space. These embeddings are then fed into a GRU, which processes the input sequence and encodes the contextual information into a hidden state. The GRU's role is crucial here as it models the sequential dependencies of the input tokens, effectively summarizing the description into a fixed-size context representation.

Next, the Decoder takes over, where it also employs an embedding layer to convert the previously generated or ground truth tokens into dense vectors. The Decoder’s GRU uses the hidden state from the Encoder to generate the title token by token. At each step, the GRU output is passed through a fully connected layer to project the hidden state onto the vocabulary space, thereby predicting the probability distribution over the next token.

Dropout is applied within the GRU layers (when using more than one layer) to help prevent overfitting. Additionally, the pipeline incorporates teacher forcing—at each decoding step，teacher\_forcing\_ratio = max(0.5 - epoch \* 0.01, 0.1),which means during the early stages of training, the model benefits from a higher chance of using the ground truth as the next token, but as training progresses, it increasingly relies on its own predictions, thereby encouraging better generalization.



I utilized a straightforward decoding strategy known as greedy decoding to generate news titles. Starting with an encoded hidden representation derived from the input description, the decoder sequentially predicted one token at a time by always choosing the most probable token at each step. This process continued until either an <end> token was produced or the maximum specified length was reached.

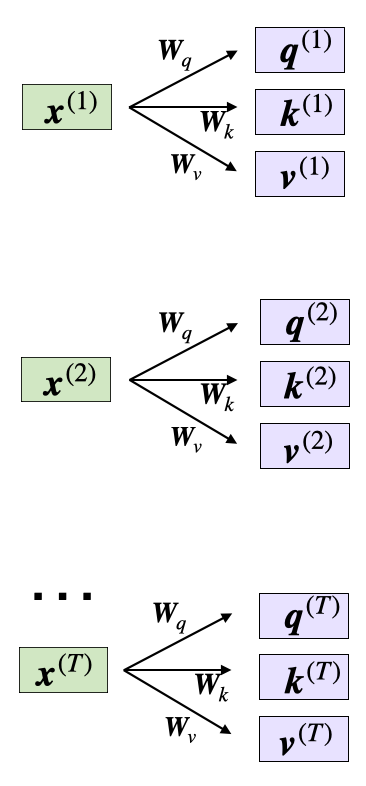
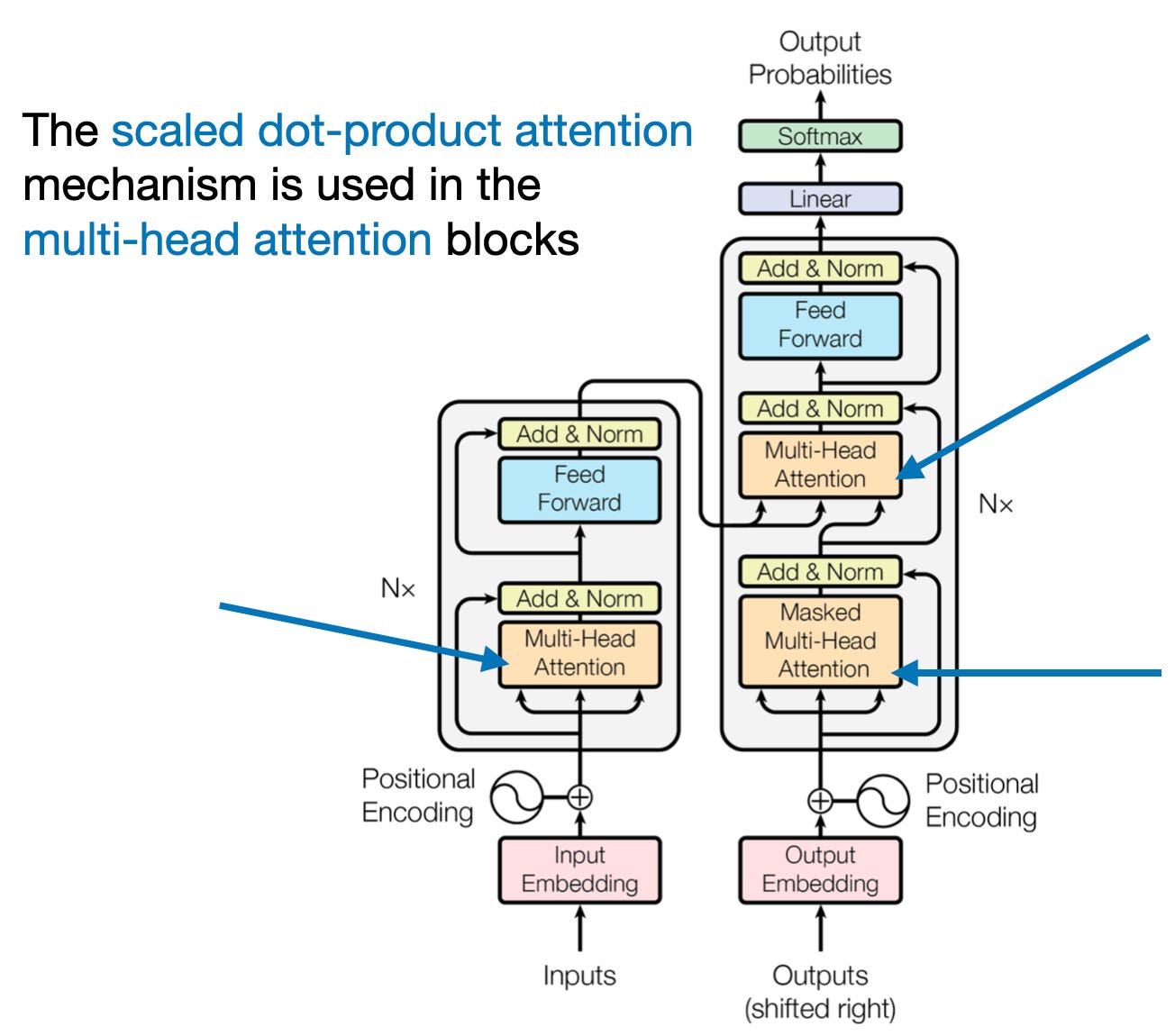


Despite implementing various techniques—such as a multi-layer GRU architecture, dropout, dynamic teacher forcing, and a decaying learning rate—the model still fails to generalize effectively on the test data. As seen from the training and validation loss curves, the training loss continues to decrease while the validation loss gradually increases, indicating significant overfitting. Furthermore, the generated titles often lack coherent semantic information, suggesting that the model struggles to capture deeper contextual relationships within the input text. This outcome underscores the inherent difficulty of the task and highlights the need for more advanced architectures, larger datasets, or additional regularization methods to improve the model’s ability to understand and summarize news content accurately.

**4.2 Transformer**

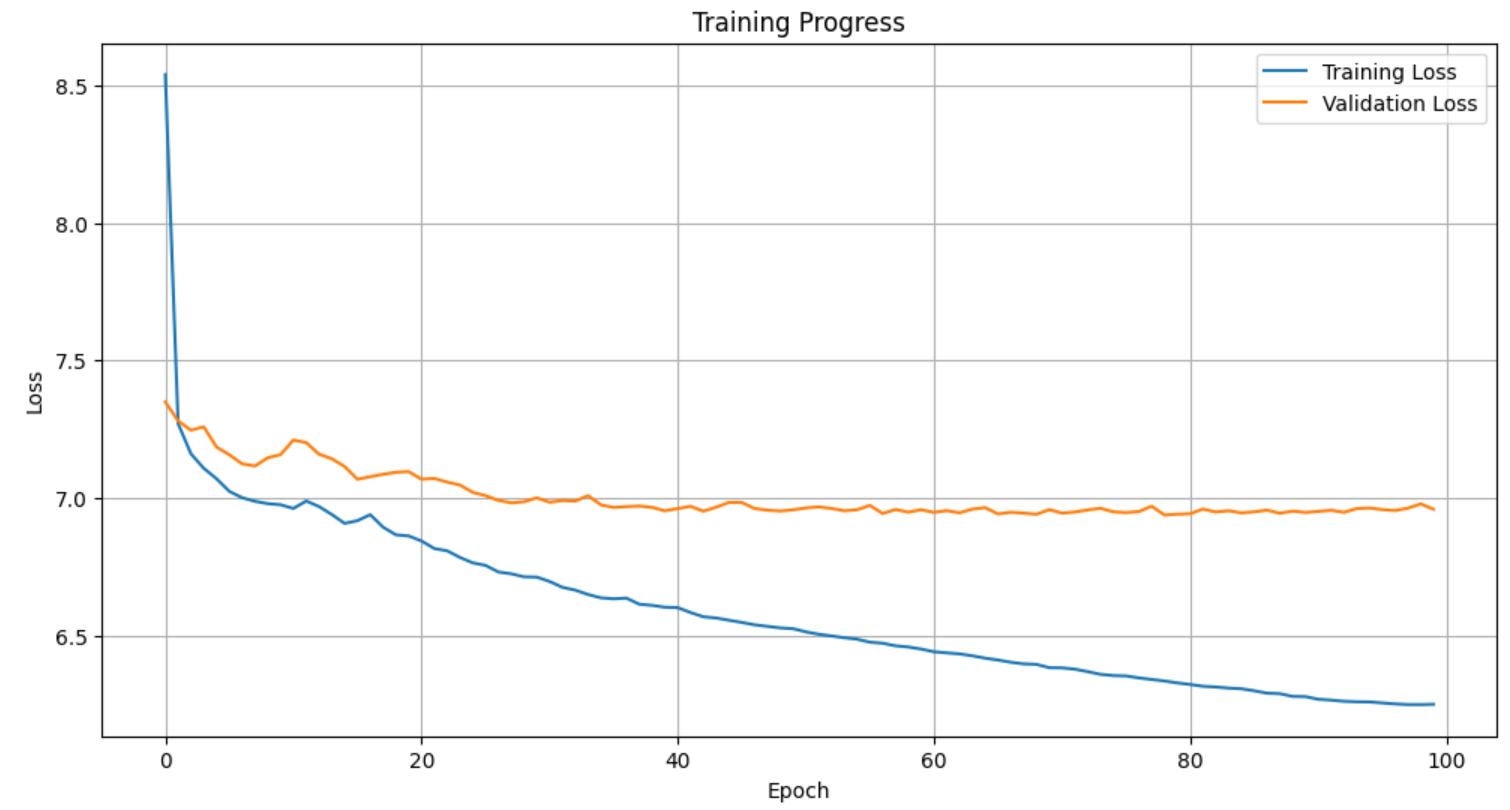
**(details in transformer\_fromScratch.ipynb)**

The Transformer architecture primarily relies on a self-attention mechanism rather than recurrent or convolutional operations. Its encoder-decoder structure consists of multiple layers, each containing self-attention and feed-forward neural networks, connected by residual connections and normalization layers. At its core, self-attention allows the model to dynamically weigh the importance of every token in the input sequence, enabling it to capture long-range dependencies and contextual relationships without being constrained by sequence length.



I implemented a Transformer architecture from scratch, closely aligning its configuration with the original Transformer paper. The custom Transformer consists of approximately 83 million parameters, with 6 encoder and decoder layers, each containing multi-head self-attention with 8 heads, 512 embedding size, and a feed-forward dimension of 2048. I adopted a dynamic learning rate scheduler, as proposed in the original paper, employing an inverse-square-root learning rate schedule with a warm-up phase to stabilize early training. Specifically, the warm-up phase here was set to 7000 steps(10 epochs), gradually increasing the learning rate to an optimal level before decaying it to ensure steady convergence.

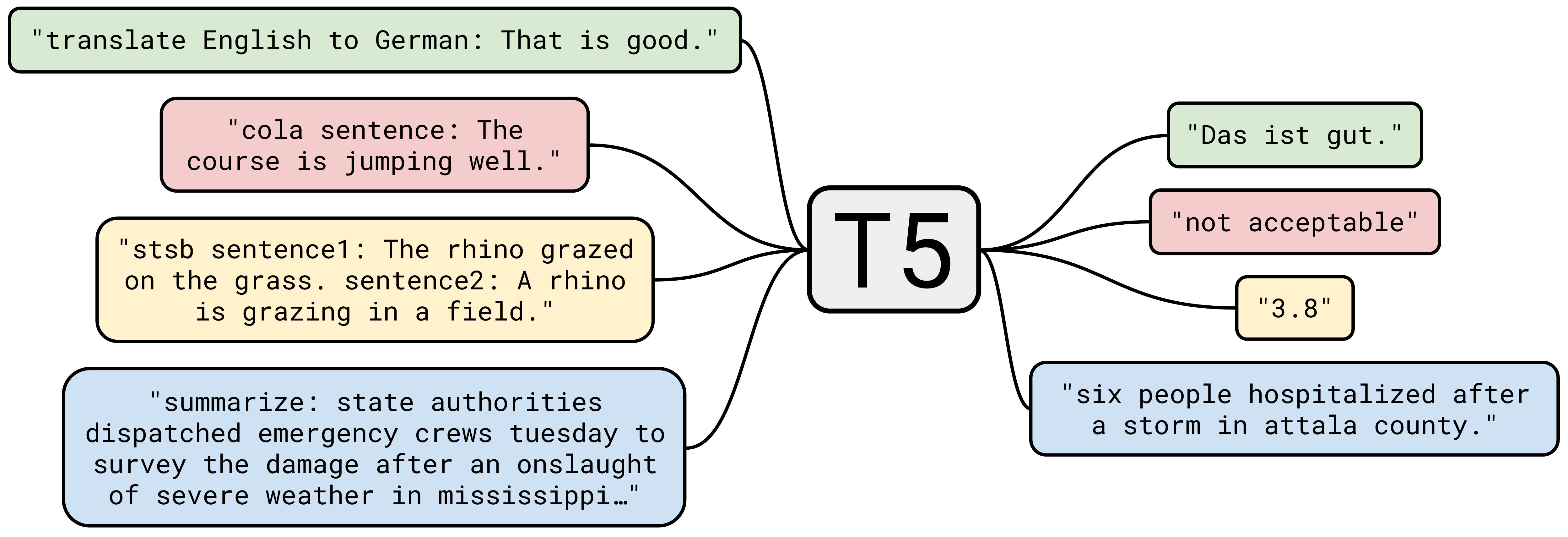
Due to computational and time constraints, the training was conducted over 100 epochs, each epoch consisting of approximately 700 iterations. Gradient clipping (max norm = 3.0) was employed to maintain training stability and prevent exploding gradients.



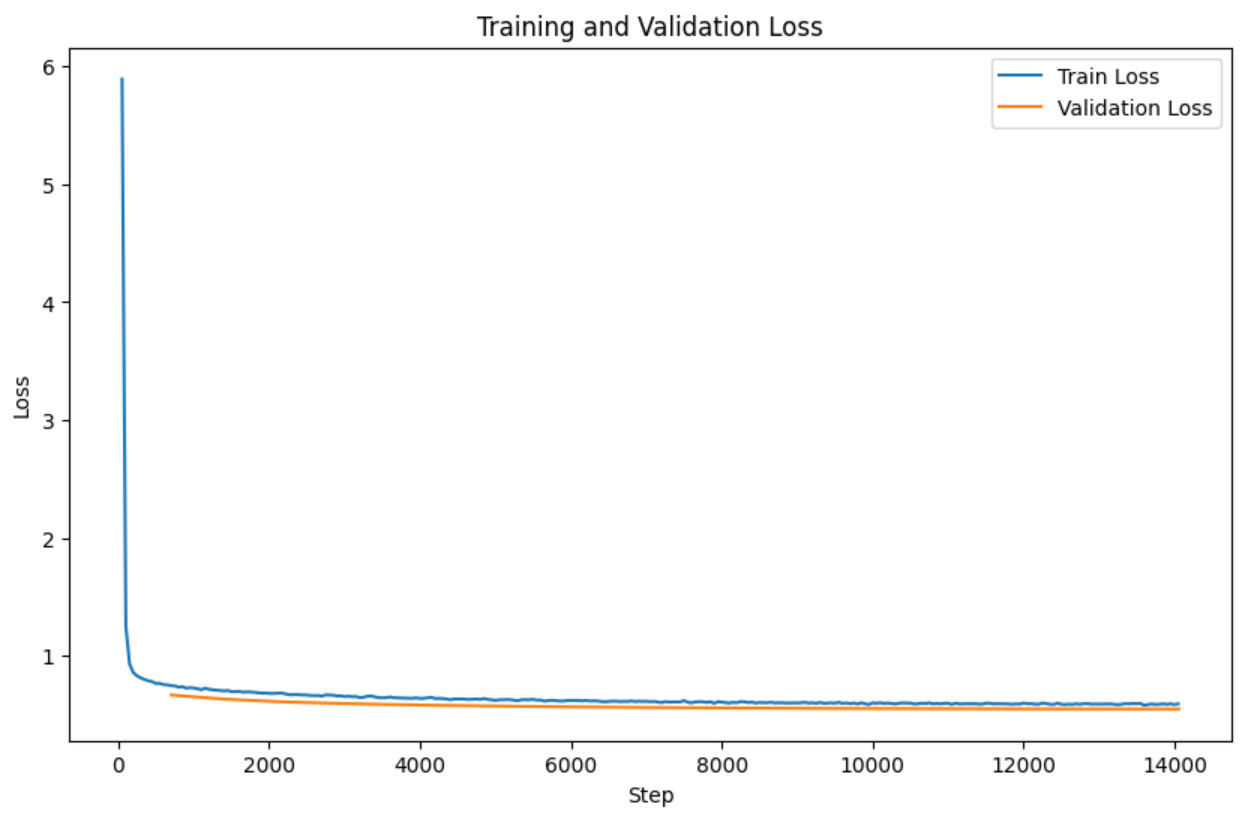
I employed Beam search decoding strategy in Transformer models to improve text-generation quality over basic greedy decoding. Instead of selecting the single most probable token at each decoding step, beam search explores multiple candidate sequences simultaneously, expanding the top-scoring sequences at each step according to a defined beam width (beam\_size=5).

**4.3 T5 (details in T5.ipynb)**

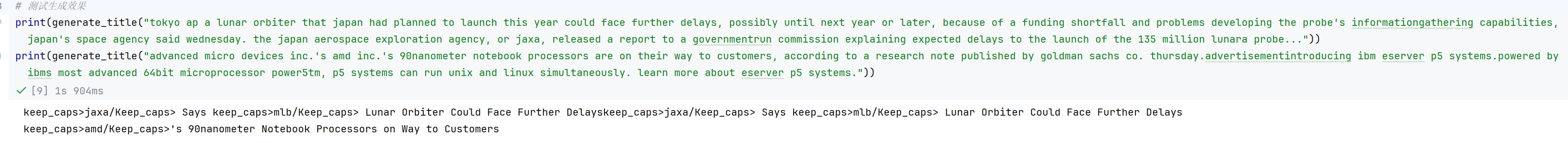
The T5 (Text-to-Text Transfer Transformer) architecture, specifically the version t5-small utilized here, is a transformer-based sequence-to-sequence model that treats all NLP tasks as a unified text-to-text problem. Its architecture primarily consists of an encoder-decoder transformer framework, featuring self-attention layers to efficiently capture long-range dependencies and contextual relationships in text. The encoder transforms the input text into a rich contextual representation, while the decoder generates the output sequence token-by-token using attention mechanisms over both encoder outputs and its own previous tokens. By using a standardized format (e.g., prefixing inputs with task-specific prompts such as "summarize:"), T5 can effectively generalize across various NLP tasks, making it particularly powerful for summarization and generation tasks.



In the training pipeline for the T5-small model, the input and target texts were tokenized separately using fixed-length padding—inputs at a maximum length of 256 tokens and targets at 64 tokens. This uniform length ensures efficient parallel computation and GPU utilization. The training utilized Hugging Face’s Trainer class with key hyperparameters set as follows: batch size of 128, learning rate of 5e-5, training over 20 epochs, and mixed-precision (fp16) training for computational efficiency.



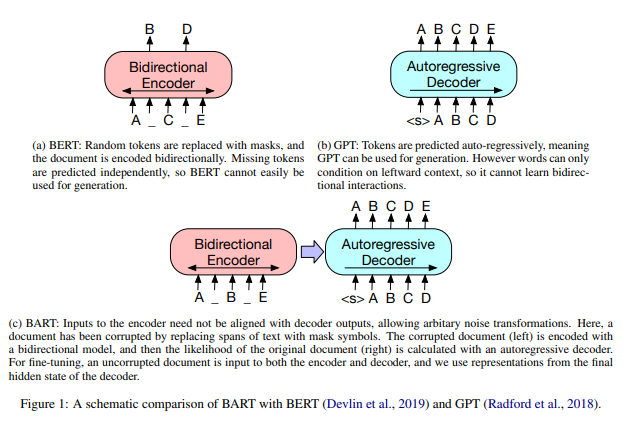
The training results, indicated by the loss curves, show rapid convergence with both training and validation loss decreasing consistently and stabilizing quickly, demonstrating good learning behavior without obvious overfitting.



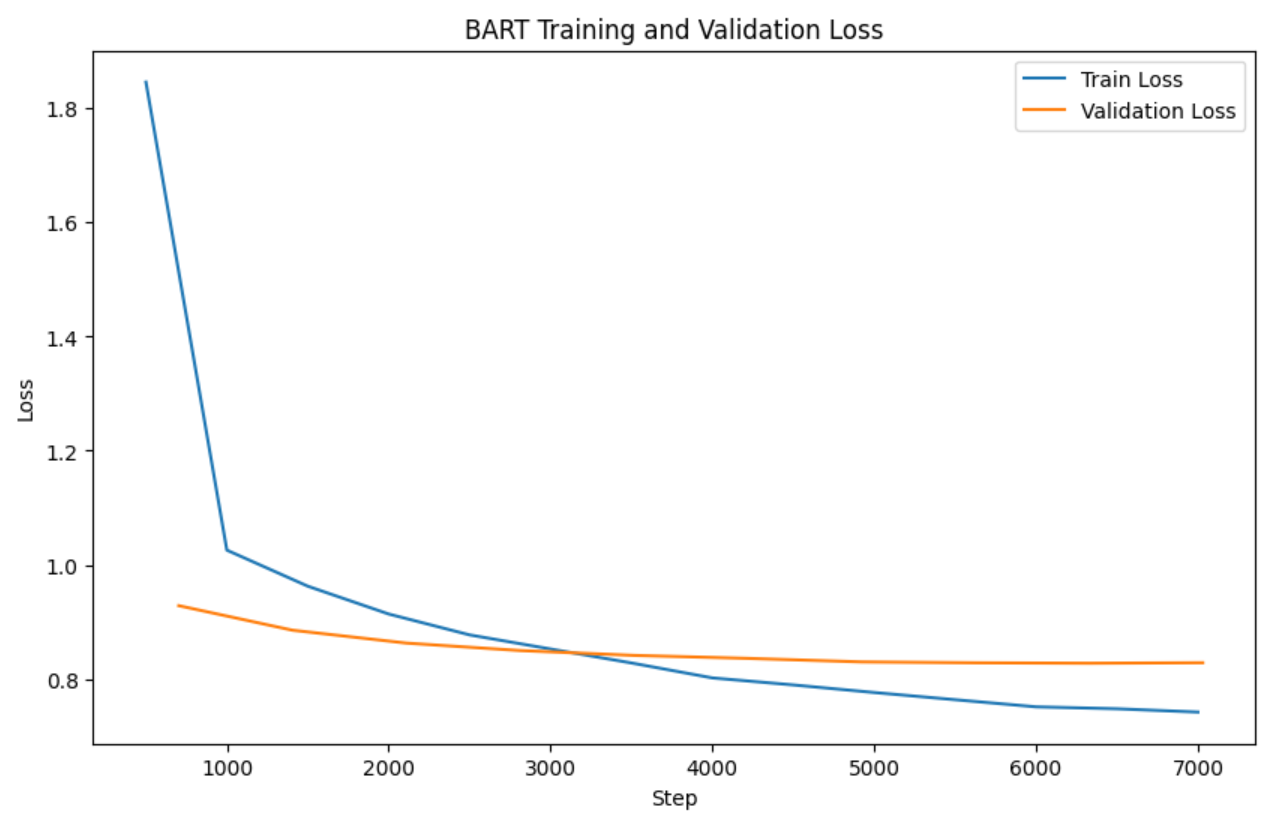
From the example outputs, the generated titles clearly demonstrate strong semantic extraction and relevance to the input descriptions, highlighting the model's superior contextual understanding compared to the previous GRU model. However, some invalid tokens, such as <keep\_caps> tags, occasionally appear. These artifacts emerge due to the specific preprocessing step intended to preserve all-capitalized terms, which introduced special placeholders in the data that T5 was not originally trained to handle.

**4.3 BART (details in BART.ipynb)**

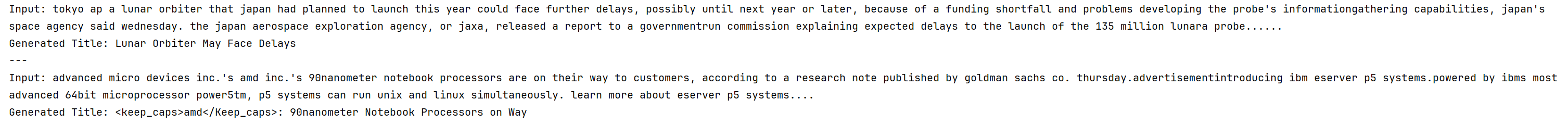
BART leverages an encoder-decoder architecture, combining the strengths of BERT and GPT. The encoder is bidirectional, similar to BERT, effectively capturing rich contextual information by understanding tokens based on both their preceding and succeeding context. The decoder, however, is autoregressive like GPT, generating text sequentially, making it ideal for text generation tasks. While BERT excels at encoding and understanding language, it doesn't natively support text generation because it lacks an autoregressive decoder. Conversely, GPT, while effective at generating coherent sequences, uses only unidirectional context, limiting its ability to fully understand context from both sides. BART’s hybrid architecture addresses these shortcomings by utilizing BERT’s strong bidirectional context modeling in the encoder to deeply understand input text, combined with GPT’s autoregressive decoder, allowing coherent and contextually relevant sequence generation. Thus, for the news title generation task, which requires precise context understanding and coherent text generation, BART provides a more effective and balanced solution than using BERT or GPT individually.



The BART model, trained over 10 epochs, exhibits rapid and stable convergence due to a combination of carefully tuned hyperparameters and its robust architectural advantages. A modest learning rate of 3e-5 ensures steady convergence without overshooting minima, and mixed-precision training (fp16) significantly accelerates computation, enabling efficient training with larger batch sizes (128 in this case). Additionally, enabling predict\_with\_generate=True allows the model to leverage beam search or similar decoding strategies during evaluation, resulting in more coherent and contextually accurate generated titles.



The fast convergence and high-quality predictions primarily stem from BART's powerful bidirectional encoding capabilities paired with its autoregressive decoder, enabling it to deeply capture and generate semantically coherent text. The model effectively summarizes input descriptions by accurately identifying key semantic elements, as evident in the provided examples where generated titles closely reflect the main content of the descriptions.



**5 Examine Ethical Implications**

When deploying text-generation models such as Transformer-based architectures, especially when fine-tuned on specific news datasets, several ethical considerations must be examined carefully. Firstly, bias propagation becomes a significant issue: models fine-tuned on datasets like AG's News inevitably reflect biases present in the original data, potentially perpetuating stereotypes, misinformation, or biased perspectives in the generated news titles. Ensuring data diversity and balance becomes crucial to mitigate these effects. Secondly, factual accuracy and misinformation are critical concerns, as Transformer-based models can generate plausible but inaccurate or misleading titles. Such outputs could inadvertently influence public perception or lead readers astray. Thus, rigorous evaluation and fact-checking mechanisms should be integrated into the deployment pipeline. Lastly, content authenticity and intellectual property issues also arise. Titles generated by advanced models like BART or T5 might closely replicate existing copyrighted content, unintentionally infringing intellectual property rights. Thus, a thorough ethical evaluation and proper citation mechanisms should accompany deployment to responsibly manage generated content in real-world applications.

**6 Conclusion**

Through extensive experimentation with different language-generation architectures for news-title generation tasks, we found that models significantly differed in their ability to capture semantic coherence and generate contextually accurate headlines. Although GRU-based and vanilla Transformer models struggled with overfitting and limited semantic understanding, advanced architectures like T5 and especially BART delivered notable improvements in contextual comprehension and text fluency. BART consistently outperformed the others, effectively combining the robust context-capturing capability of BERT’s bidirectional encoder and the fluent generative power of GPT’s autoregressive decoder.

Despite notable advances demonstrated by Transformer-based architectures like T5 and BART, several areas remain open for improvement. Computational resources pose a significant limitation; deploying large models in resource-constrained environments demands optimization techniques such as parameter pruning, quantization, and knowledge distillation. Furthermore, enhancing inference speed is critical for real-world applications, necessitating efficient decoding strategies like beam search optimization or parallel decoding methods. Finally, incorporating methods like constrained learning—for example, controlling generation length, enforcing factual accuracy, or explicitly constraining outputs to avoid misinformation—can significantly enhance model reliability and ethical compliance.

**7 Reference**

https://magazine.sebastianraschka.com/p/understanding-and-coding-self-attention

https://paperswithcode.com/method/t5

<https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html>

<https://spacy.io/models>

<https://medium.com/@anishnama20/understanding-gated-recurrent-unit-gru-in-deep-learning-2e54923f3e2>

<https://huggingface.co/docs/transformers/model_doc/t5>

https://huggingface.co/docs/transformers/en/model\_doc/bart

<https://medium.com/dair-ai/papers-explained-09-bart-7f56138175bd>

<https://stackoverflow.com/questions/77444485/using-positional-encoding-in-pytorch>

https://arxiv.org/abs/1706.03762