Development and Evaluation of a Custom-Built Abstractive Summarisation System L90 - Practical 03

Tim Luka Horstmann
Computer Laboratory
University of Cambridge
tlh45@cam.ac.uk
https://github.com/TimLukaHorstmann/L90_tlh45_exercise3
Word Count: 3541/4000

Abstract

Summarisation is a core discipline of NLP with strong practical relevance. This report presents the development and evaluation of a custom-built abstractive summarisation system (ASS) using the CNN/Daily Mail data set. Based on a Transformer model, this work explores various summarisation strategies and highlights the challenges in generating coherent and meaningful summaries using ASS. Following a thorough hyperparameter optimisation, the performance of the ASS is qualitatively and quantitatively evaluated through ROUGE metrics. A comparison with the previous practical's extractive summarisation system and baseline models reveals substantial limitations and suggests directions for further research, such as training larger models with more data, further analysing and optimising the Transformer, and exploring other machine learning techniques. This work helps to shed light on the challenges of developing effective ASS.

1 Introduction

Nowadays, the internet and digitisation have led to the production of information at an unprecedented speed [10]. This has intensified the need for systems that can efficiently condense text into concise pieces without losing relevant information. Summarisation, a key task in Natural Language Processing (NLP) within Machine Learning (ML), seeks to address precisely this problem. Due to its practical relevance with applications in various domains, summarisation has become a significant area of research in NLP and brought forward numerous tools to address the outlined challenges [6].

There are two major categories in which summarisation tools can be classified: *extractive summarisation* and *abstractive summarisation* [10]. Table 1 provides a comparative analysis of these two standard summarisation techniques and describes their strengths and weaknesses.

While the second NLP practical examined a custom-built *Extractive Summarisation System* (hereafter ESS), this report delves into building and assessing a custom-built *Abstractive Summarisation System* (hereafter ASS) which aims at summarising news articles. Just like before, a subset of the CNN/Daily Mail data set[1], containing 10,000 articles with summaries for training and 1,000 pairs for both validation and testing, will be used for the development of the ASS. As this is the same data set used in the last practical, its characteristics will not be explicitly outlined again. The ASS's code is accessible on Github[2].

[1] https://huggingface.co/datasets/cnn_dailymail
[2] https://github.com/TimLukaHorstmann/L90_tlh45_exercise3

Category	Extractive Summarisation	Abstractive Summarisation
Approach	Select most relevant coherent (existing) parts from text (e.g., sentences) and combine them [10].	Generate new text (e.g., using new words and rephrasing original text) [10].
Analogy	Highlighter: underscore the most relevant parts of a text.	Human-like: generate a summary word by word.
Strengths	Preserves original phrasing. Often higher accuracy. Faster and simpler to implement. [1]	Can be more coherent and concise. More flexible. [1]
Weaknesses	Not human-like. Less flexible. [1]	Difficult to implement. Less accurate. Computationally intensive. [1]

Table 1: Comparison of extractive and abstractive summarisation.

2 The Custom-built Abstractive Summarisation System

Deep Learning (DL) methods — ML techniques based on large neural networks — have recently found extensive application in the context of ASS [10]. Although many DL-based ASS have been developed, all of these rely on an *Encoder-Decoder* architecture [10]. Given that these *sequence-to-sequence* (Seq2Seq) models, which convert an input sequence into an output sequence, performed well in tasks similar to the one at hand [1], this report also opted for such an approach.

Based on the architecture for an ASS proposed by El-Kassas et al. [1], a custom-built ASS was developed, utilising a Transformer model for natural language generation. The custom ASS pipeline, shown in Figure 1, was implemented in Python using the PyTorch[3] DL library.

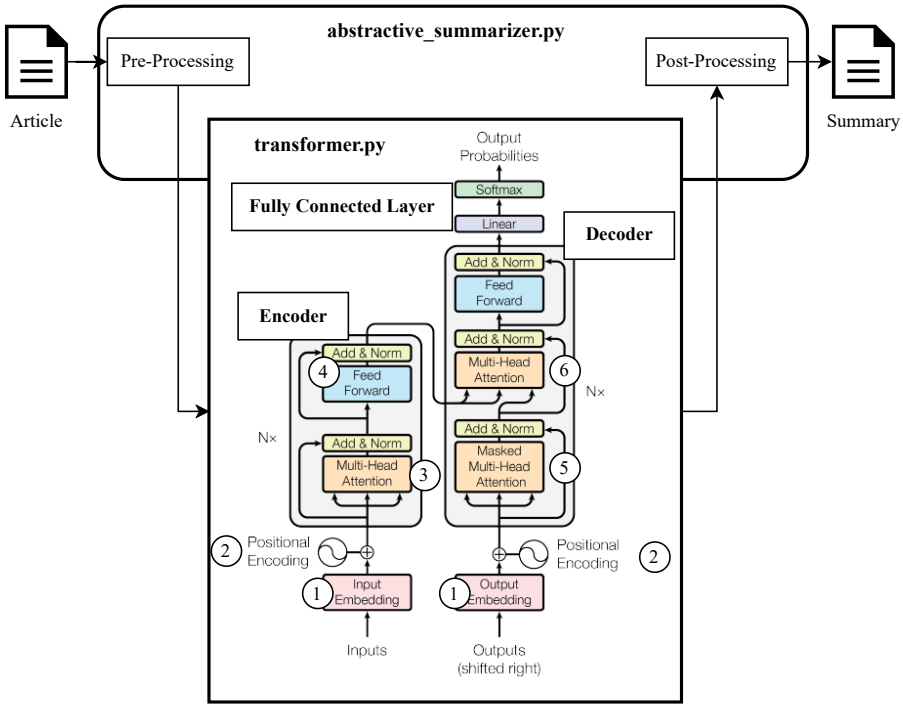

Figure 1: Simplified architecture of the custom-built ASS for article summarisation. Based on [1, 8]

In the following, the different components forming the ASS will be explained. First, a summary of the characteristics of the Transformer model will be given, followed by a description of the enclosing "abstractive summarizer" architecture that utilises the Transformer model.

[3] `https://pytorch.org/`

2.1 Transformer Model

The Encoder-Decoder-based Transformer architecture by Vaswani et al. [8] mitigates a crucial problem standard Encoder-Decoder architectures typically face [5]. Although these models often rely on *long short-term memory* (LSTM) networks — a form of *Recurrent Neural Network* (RNN) allowing Seq2Seq-models to work with longer sequences — they usually struggle with retaining information of words that appeared early on in long sequences [10]. The Transformer model efficiently implements *Attention* to retain the input values, eliminating the need for LSTMs or recurrence [5, 8].

2.1.1 Encoder

① Word Embedding and ② Positional Encoding
To prepare the input for the Encoder, the tokenized input is mapped to IDs based on a specified vocabulary and then transformed into vectors of size d_{model} (default: $d_{model} = 512$) that capture each token's semantic meaning and context through *Word Embedding* [8]. Furthermore, the Transformer requires a technique to keep track of the token order. For this, the Transformer relies on *positional encoding* through alternating Sine and Cosine functions [8]. The positional encodings are of the same size as the embeddings, d_{model}, so that they can easily be summed [8], as shown in Figure 2.

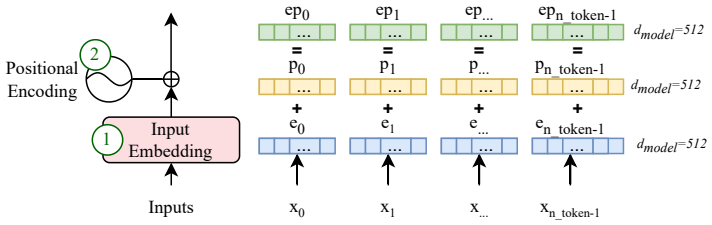

Figure 2: Summation of word embeddings and positional encoding. Based on [2]

③ Self-Attention (Sub-layer I)

Following encoding, the Transformer calculates *Scaled Dot-Product Attention* values, a form of *Self-Attention*. Self-Attention allows the model to establish relationships among the tokens and thus give context to each word [5]. The main idea is to calculate how similar each token is to all the tokens in the sequence (including itself) to determine how to encode each word [8]. Several layers of this unit, called *Attention Head*, can be formed to capture different relationships among words and enable *Multi-Head-Attention*. By default, eight heads ($nhead = 8$) are employed [8].

④ Feed-Forward Network (Sub-layer II), Residual Connections, and Add & Norm
A simple feed-forward network is added on top of the first sub-layer to generate the Encoder's output [8]. Finally, *Residual Connections* are added around both sub-layers to ensure that prior information is not lost inside either of the two layers [8]. This is achieved by adding input values x to the sub-layer outputs and normalising them: $LayerNorm(x + Sublayer(x))$ [8].

2.1.2 Decoder

As the Decoder has two inputs, this section is limited to explaining the two components distinguishing the slightly more complex Decoder from the Encoder. All sub-layers in the Decoder are enhanced with residual connections [8].

⑤ Masked Multi-Head Attention
Enabling the auto-regressive prediction behaviour of the Transformer model requires forcing the model to only attend to previous tokens to calculate its predictions because, during inference, the model cannot access future tokens as it predicts them sequentially. This is ensured by generating a square mask that sets the upper triangle of values to $-\infty$, which is applied to the input of the SoftMax function inside the first Multi-Head-Attention layer of the decoder [8].

⑥ **Encoder-Decoder Attention**
The Decoder possesses a second Multi-Head Attention layer to connect the Encoder with the Decoder [8]. This layer enables the Transformer to establish a relationship between the input and output sequence, thereby retaining the original sequence's (here: article) semantics [8].

It is important to note that, similar to the concept of attention heads, the whole Encoder layer and the Decoder layer, can be stacked and computed in parallel to capture more complex relationships between tokens. Additionally, the *dropout* technique is commonly used to randomly set neurons to zero during training and counteract overfitting [8].

2.1.3 Fully Connected Layer - Generation of Output Probabilities

Like many other sequence transduction models, the Transformer employs a simple linear network mapping the final Decoder's output of dimension d_{model} to the size of the vocabulary by a learnable weight. This *fully connected layer* outputs logits, which are subsequently run through a SoftMax function to obtain the final predictions. These can be interpreted as probabilities for each token in the vocabulary and, therefore, be used to pick the most probable next token at each prediction step [8].

2.1.4 Implementation of the ASS Transformer

The custom-built ASS's Transformer employs PyTorch's *nn.Transformer* implementation and is realised in the *transformer.py* script. While the implementation uses PyTorch's default *nn.Embedding* class to create the word embeddings and *nn.Linear* as the Fully Connected Output Layer, the Positional Encodings are calculated in a custom class. Masks are created in the Transformer's *forward()* method to either enable disregarding of padding tokens or support Masked Multi-Head Attention. In this custom implementation, the final output is not directly converted into probabilities using a SoftMax function. Instead, logits are returned, which are only adjusted later in the ASS.

2.2 Abstractive Summarizer

The Abstractive Summarizer component of the ASS displays the entry point for the system and is called by *run_abstractive_summarizer.py*. As shown in Figure 1, this part of the ASS fundamentally consists of three different logical units implemented in the *abstractive_summarizer.py* script: preprocessing, interacting with the Transformer, and postprocessing.

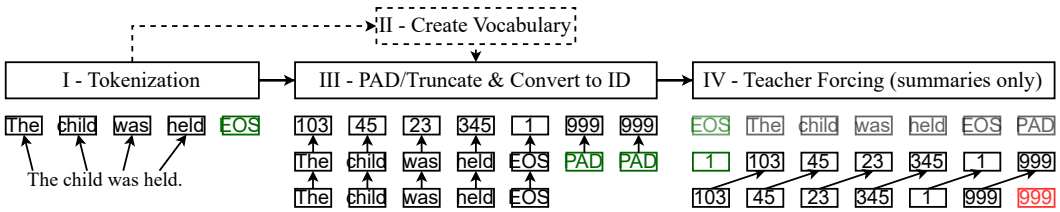

Figure 3: The custom preprocessing pipeline of the ASS. Image by Author

The custom-built ASS allows the user to choose between using the BERT Tokenizer[4] and its corresponding vocabulary or creating a custom vocabulary through the NLTK tokenizer[5]. Depending on this choice, preprocessing details differ. Figure 3 visualises the preprocessing pipeline of the ASS.

1. Tokenization: The text (article or summary) is split into tokens. An additional *End Of Sequence (EOS)* token is added at the end of each sequence.
2. Create Vocabulary (optional): If a custom vocabulary is selected, tokens are collected and numbered. Tokens can be added to the vocabulary based on a specified frequency threshold.
3. Pad/Truncate & Convert to ID: Each text is truncated or padded with extra tokens to their predefined maximum length. Due to the distinct nature of the input (articles) and output (summaries), the maximum token length was set at 2048 for articles and 128 for summaries. An analysis of the given training data, shown in Appendix A (Figure 6), validated that

[4] `https://huggingface.co/docs/Transformers/model_doc/bert`
[5] `https://www.nltk.org/api/nltk.tokenize.html`

these settings would cover the entire length of around 98% of given texts while optimising memory usage.

4. Teacher Forcing: The token IDs of the summaries are right-shifted. This is necessary to enable the later auto-regressive prediction behaviour of the Transformer by imitating it during training. In contrast to inference, however, during training, the original subsequent token is always used as the next input token — a technique known as teacher forcing.

The training of the Transformer model is executed in a conventional training loop using PyTorch's optimisation algorithms[6]. Specifically, *Adam* is employed as optimiser, and *ReduceLROnPlateau* as learning rate scheduler. Helper functions allow the validation score to be calculated as cross-entropy loss or ROUGE score. Additionally, a reduced training method for debugging (*train_to_overfit()*) was implemented. This method tests whether the Transformer can purposely overfit.

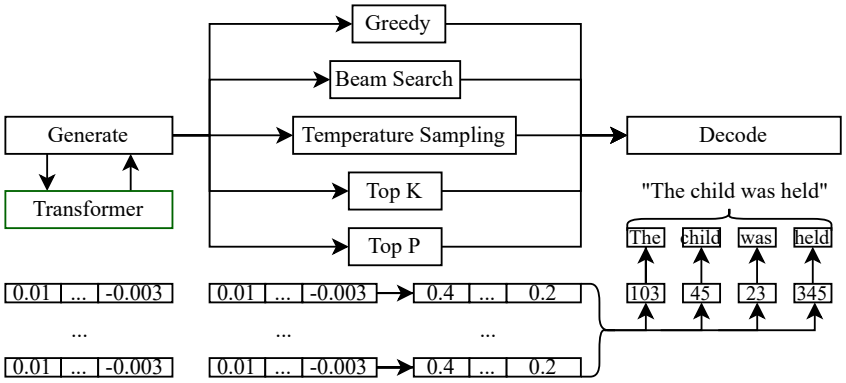

Figure 4: The custom postprocessing pipeline of the ASS. Image by Author

After training, to generate a summary (e.g., using *predict()*), the corresponding article must first be converted through reduced preprocessing by calling the *tokenize_text()* and *pad_and_convert_to_ids()* methods (see Figure 3). Subsequently, model predictions are generated using one of multiple possible strategies encapsulated in a designated *Generator* class. Thresholds can be defined to enforce desired minimum and maximum length constraints for the summary. After the Transformer has calculated predictions, the generated list of token IDs is postprocessed to transform the token IDs back into words and generate a contiguous summary. The complete prediction process is shown in Figure 4 and indicates the five different generation strategies implemented in the Generator class:

1. **Greedy**: at each step, select the token with the highest probability as the next token
2. **Beam Search**: keep track of *beam_width* hypothesis summaries. In the end, select the best summary out of these hypotheses.
3. **Temperature sampling**: at each step, sample from the returned probability distribution. A *temperature* value can be introduced to adjust the probability distribution and mitigate the generation of incoherent text, as described by Holtzman et al. [4]. For example, $temperature < 1$ can be set to increase the likelihood of high-probability tokens, thereby preventing extraordinarily unlikely tokens from appearing in the predicted summary.
4. **Top-K**: at each step, choose the K most likely next tokens and redistribute the probability mass among them before selecting the next token [3].
5. **Top-P**: at each step, find the smallest subset of tokens that have a combined probability greater than P and redistribute the probability mass among them before picking the next token [4]. In contrast to Top-K, this method accounts for the number of tokens to consider.

All five generation strategies, as well as the model's hyperparameters, were exhaustively tested in the context of this practical to identify the best-performing model. For this purpose, a custom hyperparameter search function with an associated objective (i.e., a simplified training loop) was implemented based on the hyperparameter optimisation (HPO) framework Optuna[7].

[6] `https://pytorch.org/docs/stable/optim.html`
[7] `https://optuna.org/`

3 Generating Summaries and Qualitative Analysis

After training, the ASS can be used to generate summaries as described in Section 2.2. As a precursor, *train_to_overfit()* was executed to train the model for 100 epochs on a small data set of five articles. The model successfully reproduced the original summaries, indicating effective implementation. Further manual model analysis was conducted via TensorBoard integrated in the ASS. Subsequently, the ASS (after HPO) was tasked with generating summaries for an unseen set of five news articles extracted from the NBC and BBC news websites. These articles and their sources can be found in *custom_example.json* and were already used in the second practical to evaluate the developed ESS. Table 2 provides an exemplary overview of the summaries the ASS generated for these articles.

Generation Strategy	**Selected custom examples, unseen by the model**
Greedy	**Article**: *A man with a rifle was arrested in a park near Senate office buildings across from Union Station in Washington on Tuesday, according to U.S. Capitol Police [...]* **Summary**: *new : the u. s. s. s. s. the u. the u. the u. the u. the u. [...]*
Beam Search	**Article**: *A war memorial is being guarded by police [...]* **Summary**: *the couple of his wife and a friend. the dog had been a new jersey, a new jersey, a new jersey and the attack .*
Temp. Sampling	**Article**: *A murder investigation has been launched after a 15-year-old boy was stabbed [...]* **Summary**: *study shows a motion to the 665 - year - old in theingham in the u. s. these it will be a big - a second.*
Top-K	**Article**: *A man with a rifle was arrested [...], according to U.S. Capitol Police , who said there is no reason to believe there is an ongoing threat . [...] Capitol Police Chief Thomas Manger identified the suspect [...]* **Summary**: *police chief of the victim of the attack, an attack , california, california, the police.*
Top-P	**Article**: *Researchers say they have trained artificial intelligence [...]* **Summary**: *supplies accused of reliefblyor have been jailed for 876nki learned , 000 scientists .[...]*

Table 2: Comparison of the different ASS generation strategies based on exemplary news articles.

Examining the summaries the ASS generated for the above articles, it is apparent that the model struggles to provide meaningful, coherent summaries. In particular, the following types of error can be identified from this qualitative analysis:

- **Repetitions**: For all five examples, the summary is mostly nonsensical and repetitive — a typical phenomenon in neural Seq2Seq models [6]. Particularly the generation methods "greedy" and "beam search" seem to suffer from this problem, repeating "the u. (s.)" or "new jersey" multiple times, which is behaviour backed by contemporary research [9].
- **Hallucinations**: All examples illustrate cases of hallucination. The ASS fabricates words that are unrelated to the article. For example, the beam search-generated summary contains the word "dog", which does not appear in the source article.
- **Factual Inaccuracies**: Related to producing incoherent summaries, the model also reveals clear difficulties in accurately adopting facts. For example, in the third example, "15-year-old" is turned into "665-year-old".

Although the generated summaries shown in Table 2 are not meaningful, they indicate that the ASS was partially able to capture the semantics of the corresponding articles. These (potential) semantic relationships are highlighted in grey. For example, in the case of the fourth example, the ASS accurately identified the article as pertaining to a police operation and additionally referenced the police chief mentioned in the article. Furthermore, semantically related words, such as "victim" or "attack" were predicted.

4 Model Evaluation

4.1 Numerical Analysis

The numerical analysis of the ASS was carried out in two steps. First, the hyperparameters of the Transformer model were optimised through the Optuna HPO implementation presented in Section 2.2. Secondly, the ASS was quantitatively evaluated in different configurations based on unseen test data.

To accurately optimise a sufficient combination of hyperparameters in an efficient manner, the Optuna HPO implementation was run based on a Tree-structured Parzen Estimator (TPE) algorithm as sampler in combination with a MedianPruner to stop likely worse trials early. The following hyperparameters were tested (for all other parameters, default values were kept):

- **d_model**: The dimensionality of the model, with options of 128, 256, and 512.
- **nhead**: The number of (multi-)attention heads, with options of 4 and 8.
- **num_encoder_layers**: The number of Encoder layers, integer values ranging from 1 to 4.
- **num_decoder_layers**: The number of Decoder layers, integer values ranging from 1 to 4.
- **dropout**: The dropout rate, float values varying from 0.1 to 0.5.
- **lr**: The learning rate, float values in the range from 1e-5 to 1e-2 using logarithmic scaling.

Due to GPU computing power limitations, the HPO was limited to explore 100 distinct hyperparameter combinations. Figure 5 shows the normalised cross-entropy loss on the validation data (as given in *validation.json*) as a performance score per hyperparameter for each of the 100 individual trials, with lower scores reflecting better model performance.

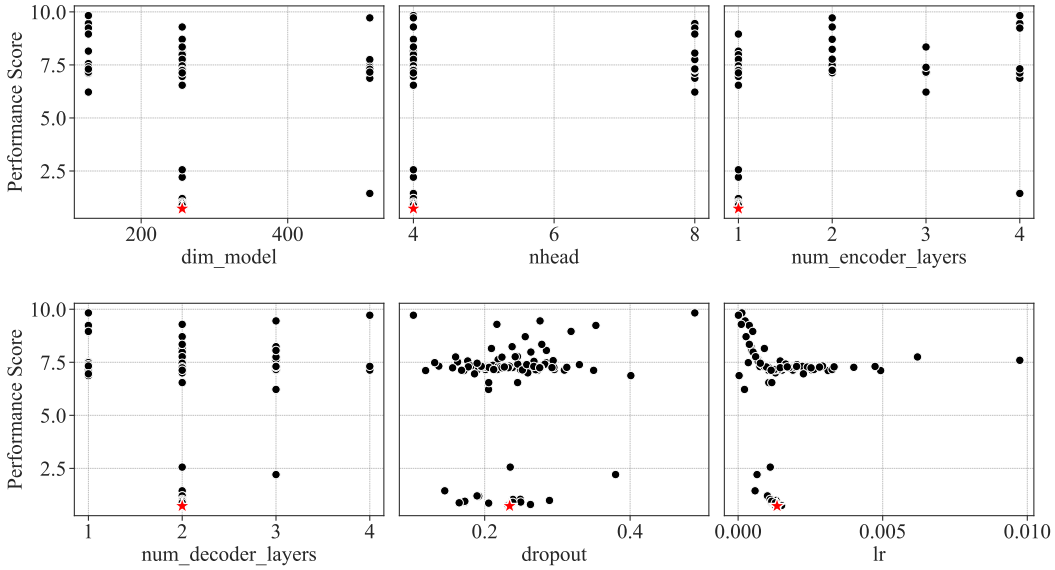

Figure 5: Results of the HPO using Optuna.

The following combination of hyperparameters yielded the best results (highlighted as a red star in Figure 5): $d\_model = 256$, $nhead = 4$, $num\_encoder\_layers = 1$, $num\_decoder\_layers = 2$,

$dropout \approx 0.23426$, $lr \approx 0.00135$. The HPO suggests that for this practical, a Transformer model with reduced complexity outperforms the default version [8].

The performance of the ASS post-HPO was subsequently evaluated based on the unseen test data in *test.json*, calculating ROUGE scores for the test data and recording Precision (P), Recall (R), and F1 Score (F1) per ROUGE category. Summaries were generated using both the Transformer model with default parameters ("Base") and the optimised model. Additionally, a model with identical hyperparameters was trained on a data set ten times larger to assess the impact of data volume on performance. The corresponding training and validation loss trends are detailed in Appendix A (Figure 7). All generation strategies were explored. The results of this analysis are documented in Table 3.

		ROUGE-1 (in %)			**ROUGE-2 (in %)**			**ROUGE-L (in %)**		
Model	**Gen. Strategy**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Increased training data *100k*	Greedy	15.79	10.56	12.66	1.68	1.10	**1.33**	14.77	9.85	11.82
	Beam	16.84	8.82	11.58	1.39	0.76	0.98	15.85	8.04	10.67
	Sampling	17.29	12.40	14.45	1.43	1.05	1.21	15.27	10.93	12.74
	Top-K	17.43	13.73	**15.36**	1.43	1.12	1.25	15.19	11.93	**13.37**
	Top-P	13.76	11.54	12.55	0.62	0.53	0.57	11.67	9.77	10.64
Transformer after HPO *10k*	Greedy	12.16	5.88	7.93	0.92	0.40	0.56	11.74	5.66	7.63
	Beam	25.52	6.64	10.54	2.44	0.67	**1.06**	23.73	6.18	9.81
	Sampling	15.37	9.25	11.55	1.01	0.62	0.76	13.90	8.34	10.43
	Top-K	15.88	9.57	**11.95**	0.92	0.56	0.70	14.32	8.59	**10.73**
	Top-P	11.55	10.35	10.92	0.34	0.33	0.33	9.99	8.84	9.38
Base Transformer *10k*	Greedy	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Beam	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Sampling	19.27	7.18	**10.46**	0.26	0.11	0.16	17.50	6.42	**9.40**
	Top-K	26.63	6.02	9.82	0.49	0.12	**0.19**	24.80	5.59	9.12
	Top-P	10.62	8.36	9.36	0.06	0.04	0.05	9.34	7.17	8.11

Table 3: ROUGE performance metrics for different models and generation strategies.
The best F1 score per ROUGE category is highlighted in bold. Note: As the ROUGE-4 score of almost all models for all generation methods is close to 0, these values are not depicted.

The numerical analysis reveals numerous insights. Since the post-HPO model achieves better F1 scores across all generation strategies than the base Transformer model, the quantitative analysis confirms the hypothesis that a simple Transformer model generally generates better summaries for the given task when comparing ROUGE scores. Furthermore, using more training data seems to positively affect performance on unseen data. Although "Greedy" and "Beam" achieved higher ROUGE-2 F1 scores, sampling methods and especially "Top-K" are overall more effective in generating diverse and relevant summaries, rendering the *Transformer after HPO 10k, Top-K* model the best configuration in the context of this practical. The fact that the base model was not able to generate considerable ROUGE scores for the "Greedy" and "Beam" generation strategies also highlights the importance of HPO for the development of the ASS in the context of this practical. Still, an unoptimised model trained on more data outperforms the smaller optimised model, suggesting that the volume of available training data represents a critical factor for the performance of an ASS.

Overall, it is striking that while the models seem to perform well on capturing individual words (as indicated by ROUGE-1 scores showing the overlap of unigrams between original and generated summary), there is a significant drop in performance when comparing bigrams (ROUGE-2). An overlap of 4-grams (ROUGE-4) cannot even be discerned. This suggests that the model is unable to adequately capture longer phrase and sentence structures. However, when discussing the ASS's performance based on ROUGE, it should be considered that the HPO minimised the defined loss objective, which does not necessarily optimise the ROUGE score performance of the model [6]. Hence, other model configurations might yield better ROUGE scores and should be explored further.

4.2 Abstractive Approach vs. Extractive Approach

To better illustrate the differing approaches of ASS and ESS, it is useful to compare the summaries generated by each for an exemplary article. Table 4 revisits an article from Table 2 and contrasts the summaries generated by the best ASS (Transformer after HPO 10k, "Top-K") and ESS, respectively.

Type	Text
Article	A man with a rifle was arrested in a park near Senate office buildings across from Union Station in Washington on Tuesday, according to U.S. Capitol Police, who said there is no reason to believe there is an ongoing threat. [...] Capitol Police Chief Thomas Manger identified the suspect as Ahmir Lavon Merrell, 21, of Atlanta.[...]
ESS Summary	S. Officers were seen sprinting to the Capitol, followed by reporters, some of whom were not allowed to leave. More than half a dozen law enforcement vehicles responded. [. . .]
ASS Summary	police chief of the victim of the attack, an attack, California, California, the police.

Table 4: Qualitative comparison of the summarisation systems developed in the NLP practicals.

The qualitative comparison vividly demonstrates that ESS summaries preserve the original phrasing and often achieve higher accuracy than ASS, due to their reconstructive character. Except for the first "sentence" generated by the ESS – which only consists of "S." – this also underlines why the ESS summaries are considerably more coherent and meaningful for the given task.

A ROUGE score-based comparative analysis of the ESS and ASS along with Lead-3 and Random-3 as baseline, as previously detailed in the second practical, reveals that the ASS did not outperform any of these models. The results recorded in Table 5 display that the ASS's performance is worse across all ROUGE categories. Since the ASS was not able to achieve considerable ROUGE-4 scores, a comparison was refrained from for this category. Furthermore, it is notable that the ASS recall scores are worse than the precision scores (although both are low). This indicates that the ASS is even less capable of capturing relevant content from the article than generating accurate summaries. For the ESS and other models, however, these scores are more balanced.

	ROUGE-1 (in %)			**ROUGE-2 (in %)**			**ROUGE-L (in %)**		
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Lead-3	32.0	45.0	38.0	13.0	19.0	15.0	27.0	38.0	31.0
ESS: Logistic Regression (Lead-3)	29.0	34.0	31.0	9.0	11.0	10.0	24.0	28.0	26.0
Random-3 (Baseline)	26.0	29.0	28.0	7.0	8.0	7.0	22.0	25.0	23.0
ASS: Transformer after HPO 10k, Top-K	15.88	9.57	11.95	0.92	0.56	0.70	14.32	8.59	10.73

Table 5: Quantitative comparison of the ASS, ESS, Random-3 (baseline) and Lead-3 model.

Overall, the comparative quantitative analysis indicates the limitations of ASS, which are findings consistent with existing literature. Other studies found similar challenges, especially if access to highly-capable computational resources and computation time were constraints [7], and it was noted that many challenges of developing ASS remain unsolved [10].

5 Conclusion and Future Work

In this practical, a Transformer-based abstractive summarisation system (ASS) was developed for the CNN/Daily Mail data set, incorporating preprocessing and postprocessing steps. Five different strategies to generate summaries were evaluated. Regardless of the strategy chosen, the ASS was capable of partially capturing relevant semantics but overall struggled to create meaningful summaries on unseen test data. Following a thorough hyperparameter optimisation, *Top-K* emerged as the most suitable generation strategy. Compared to the extractive summarisation system (ESS) developed in the last practical as well as the Lead-3 model and Random-3 model as baseline, the ASS underperformed. Accurate abstractive summarisation remains a significant challenge in NLP.

This practical suggests several directions for future research, such as training Transformer models on more than the 3% of CNN/Daily Mail data used, which could improve performance but would require advanced hardware. Furthermore, it may be helpful to analyse the Transformer's attention weights to understand which tokens are relevant for predicting certain combinations of text. Building on this, further model refinements could be introduced, such as n-gram penalties to avoid repetitive outputs, a more extensive HPO including additional hyperparameters and more combinations, or potential architectural changes (e.g., hybrid summarisation techniques combining extractive and abstractive approaches). Exploring other ML strategies for natural language generation is also recommended.

References

[1] W. S. El-Kassas, C. R. Salama, A. A. Rafea, and H. K. Mohamed. Automatic text summarization: A comprehensive survey. *Expert Systems with Applications*, 165:113679, Mar. 2021. ISSN 0957-4174. doi: 10.1016/j.eswa.2020.113679. URL https://www.sciencedirect.com/science/article/pii/S0957417420305030.

[2] K. Elhariri. The Transformer Model, Mar. 2022. URL https://towardsdatascience.com/attention-is-all-you-need-e498378552f9.

[3] A. Fan, M. Lewis, and Y. Dauphin. Hierarchical Neural Story Generation, May 2018. URL http://arxiv.org/abs/1805.04833. arXiv:1805.04833 [cs].

[4] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The Curious Case of Neural Text Degeneration, Feb. 2020. URL http://arxiv.org/abs/1904.09751. arXiv:1904.09751 [cs].

[5] Y. Kim, C. Denton, L. Hoang, and A. M. Rush. Structured Attention Networks, Feb. 2017. URL http://arxiv.org/abs/1702.00887. arXiv:1702.00887 [cs].

[6] R. Paulus, C. Xiong, and R. Socher. A Deep Reinforced Model for Abstractive Summarization, Nov. 2017. URL http://arxiv.org/abs/1705.04304. arXiv:1705.04304 [cs].

[7] V. Vasavada and A. Bucquet. Just News It: Abstractive Text Summarization with a Pointer-Generator Transformer. 2019. URL https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/custom/15720251.pdf.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need, Aug. 2023. URL http://arxiv.org/abs/1706.03762. arXiv:1706.03762 [cs].

[9] A. K. Vijayakumar, M. Cogswell, R. R. Selvaraju, Q. Sun, S. Lee, D. Crandall, and D. Batra. Diverse Beam Search: Decoding Diverse Solutions from Neural Sequence Models, Oct. 2018. URL http://arxiv.org/abs/1610.02424. arXiv:1610.02424 [cs].

[10] M. Zhang, G. Zhou, W. Yu, N. Huang, and W. Liu. A Comprehensive Survey of Abstractive Text Summarization Based on Deep Learning. *Computational Intelligence and Neuroscience*, 2022:7132226, Aug. 2022. ISSN 1687-5265. doi: 10.1155/2022/7132226. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9359827/.

Appendices

A Additional Figures

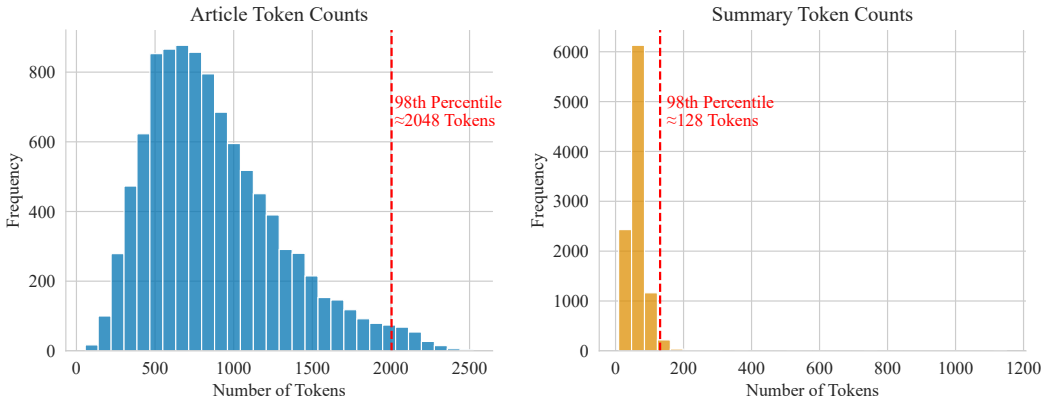

Figure 6: Distribution of the total number of tokens per article/summary in *train.json*. Image by Author

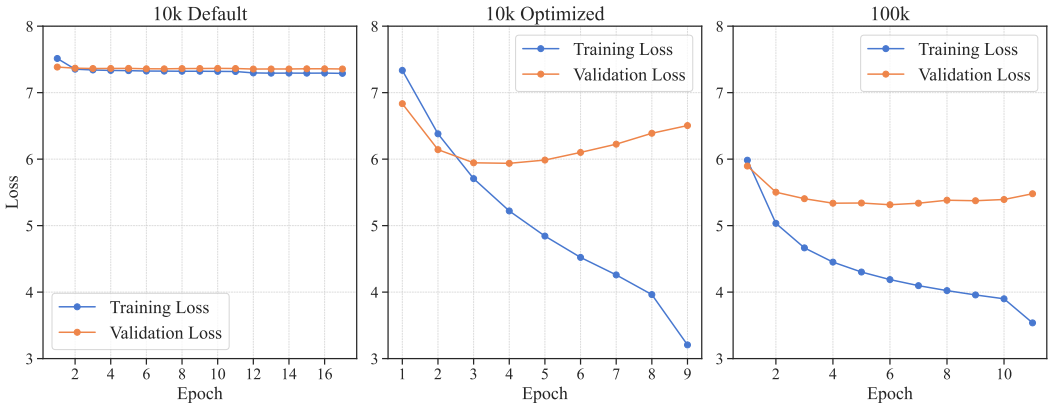

Figure 7: Training and validation cross-entropy loss for different Transformer configurations trained on 10,000 or 100,000 samples. Image by Author