CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Thesis

Text Summarization Methods in Czech

Marian Krotil
Open Informatics, Artificial Intelligence and Computer Science

May 2022
Supervisor: Ing. Jan Drchal, Ph.D.

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Krotil Marian** Personal ID number: **492001**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Text Summarization Methods in Czech

Bachelor’s thesis title in Czech:

Metody sumarizace textu v češtině

Guidelines:

The aim of this project is to research state-of-the-art NLP methods for text summarization:
1) Focus on summarizing Czech news articles.
2) Give review of state-of-the-art abstractive summarization methods and datasets.
3) Select most promising methods, train models and compare quality of generated summarizations.
4) Experiment with SumeCzech dataset as well as with data supplied by the supervisor.

Bibliography / sources:

[1] Puspitaningrum, Diyah. "A Survey of Recent Abstract Summarization Techniques." arXiv preprint arXiv:2105.00824 (2021).
[2] Lewis, Mike, et al. "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension." arXiv preprint arXiv:1910.13461 (2019).
[3] Straka, Milan, et al. "Sumeczech: Large czech news-based summarization dataset." Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018). 2018.
[4] Fabbri, Alexander R., et al. "Summeval: Re-evaluating summarization evaluation." Transactions of the Association for Computational Linguistics 9 (2021): 391-409.

Name and workplace of bachelor’s thesis supervisor:

Ing. Jan Drchal, Ph.D. Artificial Intelligence Center FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **27.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. Jan Drchal, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgement / Declaration

I would like to express my gratitude to my supervisor ing. Jan Drchal, Ph.D., for the time he gave me during the consultations as well as for his advice and guidance.

Last but not least, I would like to thank my parents, my girlfriend, and others who supported me during my studies.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date 20.5.2022

.....

Abstrakt / Abstract

Transformer architektura a modely, založené na této architektuře, předtrénované na rozsáhlých korpusech textu v posledních letech prokázaly značný úspěch v řešení různých úloh z oblasti Natural Language Processing. Tento koncept rovněž zahrnuje a usnadňuje úlohu sumarizace jak pro rozšířené jazyky jako Angličtina nebo Čínština, tak ale i pro menší jazyky jako je právě Čeština, kterou se tato práce zabývá. Úloha sumarizace spočívá v krátkém shrnutí delšího textu se zachováním všech důležitých informací. V této práci, je představen přístup použití předtrénovaného Transformer modelu mBART k vytváření českých abstraktních sumarizací. Tři české zpravodajské datasety, SumeCzech, privátní CNC, a jeden velký, vytvořený z předchozích dvou, jsou použity pro trénování modelů. Práce rovněž rozebírá vliv parametrů inferenčních metod na generování českých sumarizací. Provedené experimenty ukazují, že všechny naučené modely překonaly doposud nejlepší výsledky na všech úlohách datasetu SumeCzech, a rovněž dokazují, že model naučený na obou datasetech funguje nejlépe.

Klíčová slova: NLP; NLG; čeština; abstraktní sumarizace; SumeCzech; CNC; Transformer; mBART; HuggingFace Transformers.

Překlad titulu: Metody sumarizace textu v češtině

The Transformer architecture and models derived from this architecture, pre-trained on large corpora of text, have shown considerable success in solving a wide range of Natural Language Processing tasks in recent years. This approach also incorporates and facilitates the task of summarization for widespread languages like English or Chinese, but also for less covered languages such as Czech, which is the focus of this work. The summarization task consists in forming a short summary from a long text while preserving all relevant information. This thesis presents an approach of applying the pre-trained Transformer model, mBART, to produce Czech abstractive summaries. Three Czech news datasets, SumeCzech, private CNC, and a large one developed from the previous two, are utilized for training the models. The thesis also studies the impact of parameters of the inference methods on generating Czech summaries. The experiments conducted demonstrate that all learned models achieved state-of-the-art results on all tasks of the SumeCzech dataset and prove that the model learned on both datasets performed the best.

Keywords: NLP; NLG; Czech; abstractive summarization; SumeCzech; CNC; Transformer; mBART; HuggingFace Transformers.

Contents /

Tables / Figures

Chapter 1
Introduction

Recent research on the Attention mechanisms[1] has driven a substantial expansion of the Transformer architecture, which has become dominant across the field of Natural Language Processing (NLP) for both Natural Language Understanding (NLU) and Natural Language Generation (NLG). Similarly, pre-trained Transformers on large corpora of texts have demonstrated remarkable achievement in fine-tuning on downstream NLP tasks.

Natural language processing is a field of artificial intelligence that allows computers to understand human language and work with it. Overall, NLP encompasses many sub-tasks, but we are dealing with Natural Language Generation. Natural Language Generation refers to any environment in which we generate new text. This could be Language modeling in which the next word is predicted based on words given so far, or it could be conditioned on previous inputs and is therefore named Conditional language modeling. In Conditional language modeling, a new word is predicted after reading all previous words $y_{t-1}, \ldots, y_1$ and also some input x. This can be expressed as $P(y_t|y_{t-1}, \ldots, y_1, x)$, where P is the probability of the new word y_t. This task is used, for example, by machine translation but also by summarization. Summarization is a task in which we produce a shorter text y from the input text x, denoted as a summary, containing important information from x. We usually distinguish two types of summarization strategies, extractive and abstractive. Extractive summarization is a technique that selects or highlights parts (typically sentences) of the input document to form a summary. Whereas abstractive summarization, which we are concerned with within this thesis, is the generation of new text using neural language generation techniques. For this neural technique, in this work, we use the state-of-the-art Transformer architecture[1]. To quickly understand how this works, the Transformer architecture model is first unsupervisedly trained on a large text corpus to build textual information. This learned model is subsequently fine-tuned on the downstream task, such as summarization. The final model is then capable of generating summaries.

In this thesis, we present an approach of employing a pre-trained multilingual Transformer model, mBART[2], to summarize Czech news datasets. We trained a total of six models on three datasets for two summarization tasks, with which we achieved state-of-the-art performance on all tasks of the SumeCzech[3] dataset. Respectively, we used only two datasets, the publicly available SumeCzech dataset and a non-public CNC dataset provided by the supervisor; however, the third dataset was constructed by concatenating the previous two. Since we are dealing with a Czech summarization, there is almost no representation of research conducted or datasets for this language, in contrast to languages such as English or Chinese. Therefore, we would like to contribute to the research of Czech summarization by this work. Furthermore, we would be happy if these models would serve even to summarize texts in everyday life because there are more and more long but interesting articles on the internet. However, nobody wants to or is able to read such amounts of texts, and thus, our models would make it

easier. In addition, we hope that our work might facilitate journalists' work or at least inspire them by writing headlines and abstracts for them from the text they write.

The following lines briefly describe the structure of this work. This section also introduces the summarization tasks addressed in this thesis. In the next chapter, we introduce state-of-the-art models based on the Transformer architecture, followed by inference methods for generating summarizations and metrics that measure the resulting summarizations. The Datasets chapter details the information and partitioning of the Czech summarization datasets that we use to train our models and presents modifications made to the CNC dataset. The Implementation chapter describes the summarization pipeline and our implementation. The Experiments chapter describes the process of training our models, followed by testing various inference methods. It concludes with examples of summarizations as well as with results on our tasks and the SumeCzech tasks. At the end, the last chapter summarizes our work.

1.1 Summarization Tasks

Several types of summarization tasks can be considered depending on the structure of our news datasets, whose documents consist of a headline, abstract and full text. An example of the news document is shown in the figure 1.1. More about datasets is discussed in the chapter 3.

Headline

Moderní trendy ve vytápění

Abstract

Při plánování stavby domu nebývá volba způsobu vytápění prioritou. Jelikož ale topný systém zásadně ovlivňuj
e charakter interiéru, je vhodné zvolit topení nejlépe před zahájením projekčních prací.

Text

Důležité také je, zda bude dům také chlazen. Pokud ano, je možné zvolit takový systém, který zabezpečí distr
ibuci tepla i chladu dle potřeby. Zejména v moderních stavbách s nízkými nároky na množství dodané energie m
áme k dispozici širokou paletu možností. Nejčastějším způsobem ohřevu interiéru je v současné době podlahové
vytápění, které pomalu vytlačuje dlouhou dobu dominující radiátory. V tomto článku...

Figure 1.1. An example of a document from Czech news datasets (SumeCzech document).

Therefore, the authors of the paper SumeCzech[3] propose three possible tasks.

- Abstract to headline (A2H): generating a one- or two-sentence summary based on sentences from the abstract.
- Full text to headline (T2H): generating a one- or two-sentence summary based on sentences from the full text.
- Full text to abstract (T2A): generating a multi-sentence summary based on sentences from the full text.

The concept of the proposed tasks was considered with minor modifications. In agreement with the supervisor and the CNC data provider[1], we decided to combine all the remaining sections listed in the document, resulting in the generation of summaries from the texts while keeping all available information. The thesis, therefore, deals with the following summarization tasks:

[1] The Czech News Center (CNC) is one of the largest media houses in Central Europe and provides its readers with a wide range of magazines, news articles, journals, and much more.[4]

- Abstract + full text to headline (AT2H): generating a one- or two-sentence summary based on sentences from a combination of the abstract and full text.
- Headline + full text to abstract (HT2A): generating a multi-sentence summary based on sentences from a combination of the headline and full text.

With three datasets and two types of summarization tasks, six models need to be trained. Depending on the model applied, all tasks are considered as abstractive summarization. At the end of this work, we also evaluate the tasks proposed by the authors of SumeCzech but using the models learned on our two tasks.

The assumption that the abstract is a summary of the full text can be somewhat misleading, as the abstract in news articles is intended to attract the reader's attention, not to summarize the text purely, and therefore may contain information not occurring in the full text. This problem could be solved by human-generated reference summaries that would preserve the relevance of the text. However, this solution is time-consuming as well as beyond human capabilities, so this assumption is the only available way for generating summaries for large datasets in the Czech so far.

Chapter 2
State-of-the-art Methods

This chapter discusses the Transformers architecture we use and presents state-of-the-art abstractive summarization methods based on this architecture. This chapter concludes by presenting generation methods and metrics measuring the summarizations. For a better understanding, the representation of the models discussed below is provided here.

Encoder-based models can produce a high-dimensional representation from an input sequence, usually using bi-directionality, which is the concept of the model being trained to understand text information from both left-to-right and right-to-left contexts. This is accomplished by pre-training the model by corrupting the input text and reconstructing the original text. Text corrupting varies from model to model, but usually, token masking is applied. Our representatives are BERT and RoBERTa.

Decoder-based models (auto-regressive) can predict the next new tokens conditioned on previous already generated tokens. Hence, the model is pre-trained by employing language modeling, where the model is trained based on its generated predictions and the expected target outputs. Our representative is GPT.

Encoder-Decoder-based models (sequence-to-sequence) are a combination of encoder and decoder, where the learned encoder produces a representation of the input sequence, which is then passed to the decoder output. The decoder then regressively generates the output sequence. These models are usually fine-tuned for many downstream tasks such as summarization, translation, etc. Our representatives are, i.e., BART and others.

2.1 Attention is all you need

Recurrent neural networks[5] (RNN) have performed excellently in language modeling and machine translation in terms of the state-of-the-art results so far. However, unfortunately, they have faced a few shortcomings. Their sequential text processing could not be parallelized and often lost text information when processing long sequences. In 2017, these ailments motivated Vaswani et al. to introduce a new network architecture, the Transformer, in the paper *Attention Is All You Need*[1]. The Transformer architecture showed admirable results, beating the best at the time, and since then, this architecture has become very popular and the most used in the field of NLP.

The Attention mechanism was first invented to solve the bottleneck problem in RNNs[6–7], which is the point between the encoder and decoder where the information of the entire sequence is passed. The idea behind this technique is that it uses a direct connection to the encoder at each step of the decoder and focuses on a specific part of the source sequence.

However, Vaswani et al. wondered why not just use Attention without a recurrent unit, and thus they present a new Multi-Head Attention and definition of the Scaled Dot-Product Attention mechanism in a slightly different way. Attention can be described by performing a mapping of query and a set of key-value pairs to a space of the same dimension, where query, key, and value are vectors representing just word vectors

with positional encoding. Thus, a weighted sum of values is calculated, where each value has its weight computed from the query and the corresponding key. Both Scaled Dot-Product Attention and Multi-Head Attention are shown in the figure 2.1

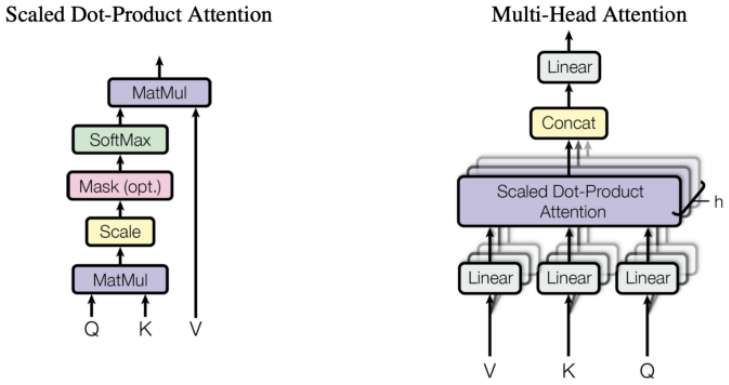

Figure 2.1. The Scaled Dot-Product Attention and Multi-Head Attention, copied from [1].

The authors use Scaled Dot-Product attention computed as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Where Q, K, and V are the matrices corresponding to queries, keys, and values, and $\sqrt{d_k}$ is the scaling factor of the size of the key vector dimension to preserve unit variance.

Next, they present Multi-Head Attention, which looks for different dependencies from different perspectives and solves a way for either distant or closer words to interact with each other. So here, Attention is computed multiple times in parallel, and the result is then concatenated and transformed into the desired dimension of the model. Multi-head attention is derived as follows:

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

They also found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections $W_i^Q \in R^{d_{model}*d_k}$, $W_i^K \in R^{d_{model}*d_k}$ and $W_i^V \in R^{d_{model}*d_v}$ to d_k, d_k and d_v dimensions, respectively. And they use $h = 8$.

Now we can define the Transformer architecture, which follows an encoder-decoder structure using Attention mechanisms and point-wise fully connected layers, as shown in the figure 2.2.

A transformer block is composed of two sub-layers, where the first is Multi-Head Self-Attention[1] and the second is a feed-forward network with ReLu, which allows us to parallelize the computation at each depth.

- The encoder part consists of these six transformer blocks with residual connections[8] around each, followed by a normalization layer[9], which changes input features to have zero mean and unit variance.

[1] Self-Attention is essentially attention, which in particular takes the input (a word vector in Q, K, V) and computes a weighted representation of the neighbors on top of it, thus figuring out how they interact with each other and whom to pay more or less attention to.

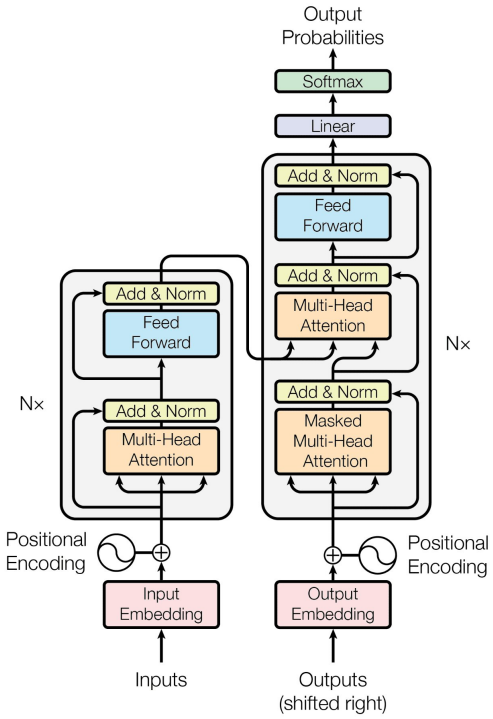

Figure 2.2. The Transformer architecture, copied from [1].

- The decoder is also made of these six transformer blocks with residual connections and normalization layers, but each block has a Multi-Head Attention between its two sub-layers to allow the decoder to access the encoder output, which is crucial in sequence to sequence tasks. The first layer is also altered Multi-Head Attention that focuses only on previously generated outputs to prevent the decoder from attending to future outputs, thus preserving decoder auto-regressive.

Before the encoder and decoder parts, learned input embeddings and positional encodings are performed, which converts each token into a vector representation of dimension d_{model} and adds positional information primarily based on cosine and sine functions of different frequencies (exact words at the different position have different overall representations). The importance of residual connections is that they carry positional information to higher layers, among other information. Moreover, a linear layer and softmax function is also added on top of the decoder to convert the decoder output to the probability distribution of the next token, where the standard cross-entropy loss could be used.

2.2 BERT

Bidirectional Encoder Representations from Transformers (BERT), one of the first developed pre-trained encoders based on the encoder Transformer architecture[1] with altered input positional embeddings, introduced in the paper *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*[10] by Jacob Devlin et al. is designed to learn bidirectional representations from an unlabeled text along with left and right context in all layers. The BERT model incorporates 12 layers of encoder Transformer and is fed with the tokenized data using WordPiece. Overall, the BERT model was pre-trained with two unsupervised learning methods. First, masked

language modeling (MLM), where the model predicts randomly selected masked input tokens (15% of the input), and second, next sentence prediction (NSP), where the model predicts if the next sentence enriches the previous or not. In particular, BERT learns a high-dimensional representation of the input sequences. The authors fine-tuned the pre-trained BERT model with just one additional output layer and achieved state-of-the-art results for a wide variety of downstream tasks.

The strength of the BERT model accounts for its widespread use in a plethora of NLP tasks, and, according to the paper[11], a substantial amount of models are derived from this BERT model. Further research on text summarization presented by Liu et al.[12] describes the application of the BERT model specifically for summarization using a document-level encoder and their new proposed learning techniques. Their results showed that the model with the pre-trained encoder relies less on individual features and learns a deeper representation of the document.

2.3 RoBERTa

RoBERTa released in *RoBERTa: A Robustly Optimized BERT Pretraining Approach*[13] by Lie et al. follows the BERT[10] architecture with different pre-training objectives and a modification of embeddings, since they utilized byte-pair encoding[14]. The authors found that hyperparameter choices have a significant impact on the final results, and as a consequence, the BERT model was substantially undertrained. In contrast to the BERT, RoBERTa is trained with dynamic masking consisting in a generation of masking patterns whenever the sequence is fed to the model and full-sentences without NSP loss, which samples sentences contiguously from one or more documents as an input. During training, the authors also used much bigger batches, learning rate, as well as more data based on their study of the BERT pre-training. In addition, the authors fine-tuned the RoBERTa on some downstream tasks, resulting in state-of-the-art performance.

2.4 GPT

GPT, the first auto-regressive model derived from a slightly different variant of Transformer decoder architecture[1], was introduced in the paper *Improving Language Understanding by Generative Pre-Training*[15] by Radford et al. The authors trained the decoder-only model using language modeling objective for pre-training on English-based dataset, followed by discriminative supervised fine-tuning on specific tasks. Despite the single decoder model, this approach achieved strong natural language understanding and improved the state-of-the-art results in many NLP tasks.

2.5 T5

T5 stands for Text-to-Text Transfer Transformer, and it is an encoder-decoder Transformer model[1] with altered positional encodings, which was introduced in the paper *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*[16] by Raffel et al. The authors provide a T5 framework, in which the model is capable of solving a wide range of NLP tasks, just by specifying a prefix in the input corresponding to the task, and then the model converts the input into a text-to-text format to understand it. The model is pre-trained using corrupted text, especially the model is trained to predict missing tokens (15% of the input), where the authors used a sizeable English

dataset. In addition, the authors fine-tuned the model with the use of the conversion of text-to-text format on downstream tasks, covering summarization, and more, in which they achieved state-of-the-art results.

2.6 PEGASUS

Authors of a paper *PEGASUS: Pre-training with Extracted Gap-sentences for Abstract Summarization*[17] introduce PEGASUS, a sequence-to-sequence model with gap-sentences generation as a pre-training objective tailored for abstractive text summarization, by which they achieved state-of-the-art performance on 12 downstream summarization tasks of English-based datasets. The model architecture is derived from the encoder-decoder Transformer[1] and is inspired by BART[18] and T5[16] models. Their pre-training consists in predicting randomly masked tokens (15% from each document) and in masking and removing a few important sentences from an input document, and subsequently generating these gap-sentences from the rest of the document as a pre-training objective. In contrast to other models, this approach proved to be suitable for abstractive summarization as it appears relatively similar to the downstream task.

2.7 ProphetNet

Another sequence-to-sequence pre-training model proposed in *ProphetNet: Predicting Future N-gram for Sequence-to-Sequence Pre-training*[19] by Weizhen Qi et al. is again based on the encoder-decoder Transformer architecture[1] with modified positional encoding in the decoder. The authors use a newly proposed technique of the self-supervised objective, future n-gram prediction, where at each step of the decoder, the decoder predicts n future tokens simultaneously. For this, they also use n-stream self-attention in the decoder, which allows them to predict n future continuous tokens respectively. During pre-training, they utilized the denoising task, where they masked 15% of the input tokens and trained ProphetNet to recover the next n future tokens within each masked token span. In addition, they fine-tuned the model with the disabled predicting stream, which acts as a standard Transformer decoder, on English summarization datasets and achieved state-of-the-art results.

According to the paper[20], where the authors investigated recent abstract summarization techniques and compared the T5, PEGASUS, and ProphetNet models for summarizing English and Indonesian Wikipedia texts. They found that although the first two models performed the best, the ProphetNet model could perform particularly well for languages other than English.

2.8 BART

BART, a denoising autoencoder for pre-training sequence-to-sequence models, was introduced *in BART: Denoising Sequence-to-Sequence Pretraining for Natural Language Generation, Translation, and Comprehension*[18] by Lewis et al. and achieved new state-of-the-art results for the text generation tasks, including abstractive text summarization, where the model outperformed all existing work. As we continue to use this model in our work, it is more fully elaborated here. The BART architecture uses a sequence-to-sequence Transformer architecture [1] with a modification of ReLU activation functions to GeLUs[21]. The authors used six layers for the base and twelve layers

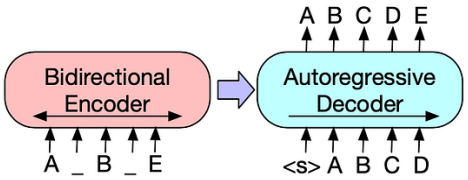

Figure 2.3. The BART model, copied from [18]. The encoder learns vector representation with bi-directional information from randomly masked input sequences, and the decoder auto-regressively predicts the output tokens from these vectors passed by the encoder.

for the large model in both encoder and decoder transformer blocks[1]. The model can be perceived as a generalization of the BERT[10] model for the bidirectional encoder and the GPT[15] model for the auto-regressive decoder, in that the encoder is attached to the decoder, shown in the figure 2.3.

Unlike BERT, which was trained using a simple token masking technique, BART utilizes complex types of masking algorithms in its pre-training, including two noising techniques, text infilling and sentence permutation, which the authors found to be the best according to their study of pre-training objectives. In the text infilling method, a few contiguous tokens are randomly masked (they chose 30% of the document), and the model learns to predict the missing tokens. In contrast, the sentence permutation technique consists in randomly mixing sentences where the model learns the logical consequences of the sentences. So the authors tokenized documents using the same byte-pair encoding as GPT-2[22] and trained BART by corrupting documents and then backpropagating the cross-entropy loss between the decoder's output and the original document.

In contrast to the BART model, the BERT model predicts masked tokens independently and is, therefore, more complicated to use in generation tasks than BART.

2.9 mBART

While BART has been pre-trained only for English, researchers at Facebook AI investigated the effect of pre-training on different languages and came up with a new model, mBART, which they present in the paper *Multilingual Denoising Pre-training for Neural Machine Translation*[2]. According to the abstract, this is the first method for pre-training a complete sequence-to-sequence model by denoising full texts in multiple languages, leading to a multilingual model. The mBART model is trained in the same way as the BART[18] model but on a multilingual corpus. This approach works effectively, which is also confirmed by the state-of-the-art results in the machine translation task.

The authors downloaded a large corpus with a total of 25 languages from Common Crawl, including English, Russian, and many others but also including Czech, which is an advantage for us because the bidirectional encoder and tokenizer can interpret Czech words. Then, they tokenized the data with a SentencePiece model[23] learned on the entire dataset, leading to a vocabulary size of 250 000 subword tokens. Also, each input document was separated by the end of a sentence (</S>) token and extended by the language token id. They configured the maximum length of the sequences to 512 tokens. For the decoder, the input is the original text with one position offset and with the addition of a language token id as the initial token to predict the sentence.

The mBART model follows the BART large model, consisting of 12 encoder layers and 12 decoder layers with a model dimension of 1024, resulting in 680M learnable

parameters. The authors trained several models, but we are focusing only on the mBART25, which we use in our work. The mBART25 model has been trained for 500K steps with a batch size of 8000 samples using the BART[18] objective, which took approximately 2.5 weeks, and this is the checkpoint with learned weights we use. The following figure 2.4 shows an example of the pre-training and fine-tuning process.

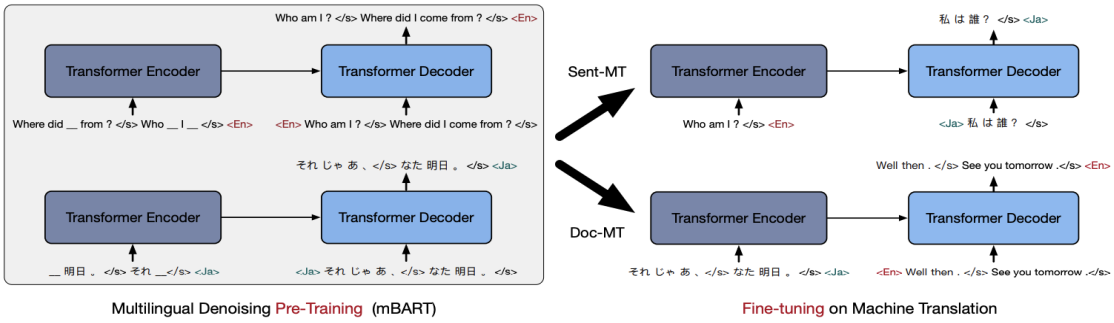

Figure 2.4. Scheme of Multilingual denoising pre-training(left) and fine-tuning on downstream machine translation task(right), where the sentence permutation and text infilling as the objectives. Language id is added at both encoder and decoder. Copied from[2].

In conclusion, the model proved to be very powerful, and, among other things, the authors demonstrated that an extensive multilingual vocabulary could improve generalization even for unseen languages. We use this model in our work based on a recommendation from the supervisor who has conducted preliminary experiments with many models in which this model appeared to be the best.

2.10 Inference

In the following sections, we introduce the theory behind the different methods of output generation in the decoder, which is usually used by the state-of-the-art models and which we employ in our work for generating summaries. Beam search 2.10.1 measures the most probable summary, whereas sampling methods measure the current most probable word at a given step.

2.10.1 Beam search

Beam search [24] is widely used in NLP to generate output sequences. It is a search algorithm that, on each time step, tracks the b most probable sequences, called hypotheses, and finally selects the sequence with the highest product of probabilities of the included words, where b denotes the beam size. For more efficient use, instead of the product of the probabilities, the sum of the logarithms of these probabilities normalized by the length of the hypothesis is performed. The normalization factor addresses the difference among sequences of varying lengths since each hypothesis can produce the end sentence token at various time steps. Despite giving reasonably good results for text fluency, this algorithm is not optimal overall, leading to not finding the best solution and often suffering from repetitive word sequences. In addition, when the beam size is fixed at 1, the beam search algorithm becomes greedy and selects the word with the highest probability at each time step, which requires less computation resulting in the faster output generation but at the cost of non-fluent repetitive text.

2.10.2 Sampling with Temperature

Sampling is a non-deterministic method that randomly selects a new word from a probability distribution of words conditioned on previously generated words. On its own, this method is not very efficient and can generate improbable words out of context. Therefore it is commonly used in conjunction with temperature. According to the paper *CTRL: A Conditional Transformer Language Model for Controllable Generation* [25], Nitish Shirish Keskar et al. briefly describe the shaping of probability distributions using temperature[26]. They show that an established size of temperature can flatten or sharpen the distribution as follows $p_i = \frac{exp(x_i/T)}{\sum_j exp(x_j/T)}$, where $T > 0$ denotes the temperature, $x_i \in R^d$ indicates scores for each word (token) i in the vocabulary, and p_i is the probability of predicting the i-th word. Setting $T \in [0, 1)$ magnitudes the high-probability words as well as lowers the low-probability words, whereas $T \in (1, \infty)$ flattens the distribution. In addition, this sampling method could be applied in combination with other mentioned methods.

2.10.3 Top-k Sampling

In the course of story generation research, the top-k method was introduced in the paper *Hierarchical Neural Story Generation*[27] by Angela Fan et al. They applied a top-k sampling method to generate stories using the learned models, which proved to be more efficient than beam search in some contexts. The top-k sampling method consists in selecting the next generated word from the k most probable words in the vocabulary, where the number k is determined in advance. To put it in perspective, the model produces a probability for each vocabulary word based on previously generated words at each time step, then selects the k words with the highest probability, which are rescaled to produce a new probability distribution. To generate a new word, the algorithm randomly samples one word from the new distribution composed of the k most probable words. The main reason for using this strategy is that it reduces the possibility of generating unlikely words when focusing only on k samples.

Let us have an example, we have set $k = 3$ and have already generated "*I love*" the part of a sentence. Suppose we have a sorted list of vocabulary words with their probabilities based on the words already generated "$V_{P(word|I,love,)}$ ={ *'you'* : *0.4*, *'programming'* : *0.15*, *'the'* : *0.09*, *'a'* : *0.06*, ..., *'do'* : *0.001*}". We take the 3 most probable words from the vocabulary and rescale their probabilities to add up to 1. We do this by multiplying the normalizer= $\frac{1}{0.4+0.15+0.09} = \frac{1}{0.64}$ with each probability of the selected words. Then we get a new probability distribution of top-k words "V_{top-k} ={ *'you'* : *0.625*, *'programming'* : *0.234*, *'the'* : *0.141*}" from which we randomly sample a new word, let's say we have selected the word "*the*", and thus our sentence is "*I love the*". Since we are using the top-k sampling strategy, we are filtering out the least likely words, but we are still sampling randomly, and among the selected top-k words, there may be some words due to a flat distribution that, i.e., beam search would never select in some context, and this method can thus be a bit creative.

2.10.4 Nucleus Sampling

The top-k strategy has a major problem with fixed k by obtaining very unlikely words when having a peaked distribution or vice versa. This problem could be solved by using the Nucleus sampling method known as top-p sampling, which was introduced in the paper *The Curious Case of Neural Text Degeneration*[28] by Ari Holtzman et al. The authors showed that by using the top-p method, generative models could reduce the

repetition of outputs, and also, the vocabulary distribution generated by models that have an "unreliable tail" consisting of thousands of low-probability candidate words are truncated. This method is very similar to the top-k method, but instead of selecting the top-k words, the words with the highest probability whose cumulative probability is equal to or higher than the specified number p are selected. Then the next generated word is sampled from a new distribution composed of these top-p words.

Let's consider our example, we have set $p = 0.5$ and have already generated "*I love*" the part of a sentence. Suppose we have a sorted list of vocabulary words with their probabilities based on the words already generated "$V_{P(word|I,love,)}$ =*{ 'you' : 0.4, 'programming' : 0.15, 'the' : 0.09, 'a' : 0.06, ..., 'do' : 0.001}*". We are searching for the most probable words from the vocabulary whose probabilities add up to or exceed 0.5. We then rescale the probabilities of the top-p words into a new probability distribution as described above. Hence, the new probability distribution with top-p words is "V_{top-p} =*{ 'you' : 0.73, 'programming' : 0.27}*" from which we then randomly sample a new word.

2.10.5 Repetition penalty

The problem of generating repetitive text negatively affects all of these methods. Beam search, in particular, suffers from this problem the most. However, the use of sampling also faces this problem when a peaked distribution is present, either generated using temperature or for many different reasons. The authors of the paper *CTRL: A Conditional Transformer Language Model for Controllable Generation* [25] propose a novel sampling scheme preventing repetitions through a penalty by discounting the scores of previously generated words (tokens). According to the paper, the probability distribution p_i is defined as $p_i = \frac{exp(x_i/(T*I(i \in g))}{\sum_j exp(x_j/(T*I(j \in g))}$ $I(c) = \theta$ if c is true otherwise 1, where $T > 0$ denotes the temperature, $x_i \in R^d$ indicates scores for each word (token) i in the vocabulary, g is a list of generated tokens and $\theta > 1$ denotes a penalization factor.

2.11 Metric

ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation and was first introduced in the paper *ROUGE: A Package for Automatic Evaluation of Summaries*[29] by Chin-Yew Lin et al. This evaluation metric is widely used in summarization tasks, and all the aforementioned state-of-the-art models dealing with the summarization task were measured by this metric. The main idea of this metric is to measure evaluation by the overlapping n-grams between the text generated by a system or model and the reference text, denoted as golden text, which was written by a human. An n-gram is defined as a subsequence of n consecutive words from a given sequence. In essence, ROUGE involves two scores, recall and precision:

- Recall in the context of ROUGE for summarization basically measure how much of the golden summary is captured by the system generated summary and can be computed as follows:

$$Recall = \frac{numberOfOverlappingWords}{numberOfWordsInGoldenSummary} \tag{1}$$

- To understand how many extra words are in summary generated by the system, precision plays a role. Essentially, this measures the proportion of words in the

system summary that occur in the golden summary. The equation is derived as follows:

$$Precision = \frac{numberOfOverlappingWords}{numberOfWordsInSystemSummary} \tag{2}$$

In addition, the ROUGE scores are computed in three ways ROUGE$_{RAW}$-1, ROUGE$_{RAW}$-2, and ROUGE$_{RAW}$-L in this thesis because it was considered to use language-agnostic ROUGE, which utilizes no stemmer and no stop words, provided by the authors of SumeCzech[3] due to the comparable results. They denoted it as ROUGE$_{RAW}$. For example, let us have golden summary "*I love programming*" and system summary "*I love programming so much*" which carry the same meaning but are written differently.

- ROUGE$_{RAW}$-1 computes recall and precision from overlapping uni-grams between the golden summary and the system summary. Let's consider our example, the golden summary and the system summary in uni-grams are *[I, love, programming]* and *[I, love, programming, so, much]* respectively. Hence, the overlapping uni-grams are *[I, love, programming]* making $recall = \frac{3}{3}$ (1) and $precision = \frac{3}{5}$ (2).
- ROUGE$_{RAW}$-2 computes recall and precision from overlapping bi-grams between the golden summary and the system summary. According to our example we have bi-grams as *[I love, love programming]* for golden summary, similarly done for system generated summary. The overlapping bi-grams thus are *[I love, love programming]* which means $recall = \frac{2}{2}$ (1) and $precision = \frac{2}{4}$ (2).
- ROUGE$_{RAW}$-L computes recall and precision from the longest common subsequence words between the golden summary and the system summary. In our example, the ROUGE$_{RAW}$-L score equals the computed ROUGE$_{RAW}$-1 score because the longest subsequence captures all the words in the overlapping unigrams. For the scores to differ, the same word would have to occur in both summaries, but at apparently distant locations, to interrupt the longest sequence but be recorded by ROUGE$_{RAW}$-1.

All the results from the example above show a recall of 1, indicating that the system summary contains everything from the golden summary, whereas the lower precision indicates that the system-generated summary contains some different words, which is not a significant issue in this case but might be somewhere else. Therefore, these scores are often used to calculate the F1-score (f-score), which is the harmonic mean of precision and recall and is calculated as follows:

$$F = 2 * \frac{recall * precision}{recall + precision} \tag{3}$$

Chapter 3
Datasets

This chapter describes the Czech datasets composed mainly of news texts that we use in this work and play a key role in fine-tuning and testing the summarization models. This chapter also provides information about the documents included in the datasets and shows the statistical representation of their sections, as well as describes the partitioning of each dataset into the train, validation, and test sets. It also reports our modifications made to the new Czech private dataset to avoid textual inconsistencies when training the models.

3.1 The SumeCzech Dataset

SumeCzech dataset containing over one million Czech news articles was released in 2018 by the Institute of Formal and Applied Linguistics, *SumeCzech: Large Czech News-Based Summarization Dataset*[3]. The documents were collected from the five news websites shown in the table 3.1 below, which describes the representation of documents for each website. The dataset was downloaded from Common Crawl[1] via scripts provided by the authors of SumeCzech.

Website	Number	Percentage
ceskenoviny.cz	4 854	0.5%
denik.cz	157 581	15.7%
idnes.cz	463 192	46.2%
lidovky.cz	136 899	13.7%
novinky.cz	239 067	23.9%
Total	1 001 593	

Table 3.1. Number of documents from individual websites. Table is copied from [3].

3.1.1 Structure of the Dataset Documents

Each document in the dataset is divided into three sections. The first section, named headline, describes the entire article, usually in one sentence. The second section, an abstract, is supposed to be a shorter version of the third section called full text. And the full text bears the whole message and is the article itself. SumeCzech dataset also includes additional sections focusing on more precise specifications regarding the document; however, these are not part of the summarization task and are ignored.

The authors provide the dataset in JSON Lines format without escaping non-ASCII characters to keep it human-readable, with each line containing all three sections of one document.

[1] `http://commoncrawl.org`

	Q1	Median	Q3	Mean	Stddev
Headline	7	9	11	9.4	2.9
Abstract	33	42	51	42.2	14.8
Text	265	378	553	470.1	365.3

Table 3.2. Quantitative statistics of lengths of headlines, abstracts, and texts in words for the SumeCzech dataset. Q1 and Q3 denote the first and the third quartile, respectively. Table is copied from [3].

3.1.2 Dataset Statistics

Authors provide the table 3.2 showing the statistics of the number of words in each section of the headline, abstract, and full text of the Sumeczech dataset[3].

3.1.3 Dataset Split

Before splitting the dataset, the authors created an out-of-domain (OOD) test set by clustering the dataset with the K-Means algorithm based on the normalized L2 similarity of abstracts. Therefore, the out-of-domain test set contains different articles from the subsequently created train, validation, and test sets. The main idea behind creating the OOD test set was to collect data that did not appear in the training set on which a model is trained and can thus be considered as real-life data. In total, the OOD set contains 44 976 documents, the test set 44 567 documents, the validation set 44 454 documents, and the training set 867 596 documents.

3.2 The CNC Dataset

The Czech News Center dataset (further CNC dataset) is supplied by a supervisor who obtained it from the company Czech News Center, one of the largest media houses in the Czech Republic[4]. The CNC dataset is private; hence only a few examples are available in this thesis, and mainly consists of Czech articles from an online media such as blesk.cz, isport.cz, e15.cz, reflex.cz, and many others. The correct representation of articles belonging to certain newspapers is unknown due to the fact that the data has been obtained in the raw text without a description of its origin.

3.2.1 Structure of the Dataset Documents

Each document is divided into three sections in the same way as the SumeCzech dataset 3.1.1 and therefore contains a headline, abstract, and full text sections.

3.2.2 Data Preparation

When examining the dataset, we revealed inconsistencies in the texts that could lead to bias and adversely affect the training of the models. Data exploration was performed using functions from the Pandas library, and subsequent data cleaning was accomplished by the filter and map functions from the Hugging Face Datasets library. The preparation of the CNC dataset is partly based on the idea of the SumeCzech paper [3] in order to maintain the similarity of the datasets and avoid feasible biases. The table 3.3 and the following steps demonstrate how we prepared the data.

1. Documents containing duplicate text either in the abstract, headline, or full text were dropped. Unique documents were retained. In total, 15 173 documents with

duplicate headlines, 23 650 documents with duplicate abstracts, and 4 113 documents with duplicate full texts were removed.

2. Documents that included text with JSON format and advertising subtext were dropped based on a simple regex. In total, 184 documents were removed.
3. Documents with English and German full text were removed by using the detect function from the langdetect[2] library ported from Google's language detection library. The detect function also provides Czech language detection. However, when testing the detect function, the results for Czech texts were not very accurate, and sometimes the function outputs Polish or Slovak for entered Czech text. Therefore only German and English recognition is used here. In total, 470 non-Czech documents were removed.
4. Several documents have been cleaned of various frequently occurring special symbols such as HTML tags, hashtags, or unique keywords, both in the abstracts and full texts. These symbols have been replaced with empty strings based on regex.
5. A large number of occurrences of the keywords VIDEO and FOTO were observed during data exploration. These keywords were frequent in the documents, both in the headlines and full texts. In the headlines, only the keywords were replaced by an empty string, whereas in the full texts, these keywords appeared with their descriptions which were out of context and would bias the training of the models. Therefore the full texts were cleaned of the keywords with their descriptions based on a regex.
6. Several documents were cleaned of multiple occurrences of spaces by replacing them with a single one.
7. A couple of documents containing an empty section and considerable amounts of documents that did not satisfy the word count requirements were filtered out. The word count requirements were set for each section separately:
   - Headline at least 1 word
   - Abstract at least 10 words
   - Full text at least 50 words

   In total, 21 188 documents were dropped based on these word count requirements.

	Initial	Dups	JSONs	NonCz	Count	Final
Docs	810002	42936	184	470	21188	745224

Table 3.3. A number of documents removed during data preparation. Initial means the initial dataset size in the documents, Dups indicates removed documents with detected duplicates (step 1), JSONs means removed documents with JSON format occurrences (step 2), NonCz denotes removed non-Czech documents (step 3), and Count stands for removed documents based on word count requirements (step 7). The Final is the final size of the dataset in the documents after these steps have been performed.

3.2.3 Dataset Statistics

Before the modifications mentioned in the section 3.2.2 above, the dataset contained approximately 810 000 documents. An estimated 65 000 documents were filtered out during the data preparation, and the resulting dataset hence includes around three-quarters of a million documents.

The table 3.4 shows the statistics of the number of words in each section of the headline, abstract, and full text of the CNC dataset. It can be seen that the average

[2] `https://pypi.org/project/langdetect/`

length of the headline section is approximately 10 words and reaches one-fifth the size of the abstract, and the abstract reaches almost one-seventh the size of the full text. Compared with the statistical values from the SumeCzech dataset table 3.2, it can be inferred that the full texts of the CNC dataset's documents are relatively shorter with respect to the abstract.

	Q1	Median	Q3	Mean	Stddev
Headline	9	10	12	10.5	2.7
Abstract	34	45	58	46.9	17.5
Text	147	240	375	318.4	309.7

Table 3.4. Quantitative statistics of lengths of headlines, abstracts, and texts in words for the CNC dataset. Q1 and Q3 denote the first and the third quartile, respectively.

3.2.4 Dataset Split

After data preparation, we split the dataset into train, test, and validation sets using a function *train_test_split* from the *sklearn.model_selection* library[3]. The data was also randomly shuffled with chosen seed to reduce bias in the evaluation, and then 35 000 documents were selected for the test set, 35 000 documents for the validation set and the remaining 675 224 documents were kept for the training set.

3.3 The CNC-Sum Dataset

As the name suggests, this dataset was developed by concatenating two previous datasets, the SumeCzech dataset, and the CNC dataset, in order to provide a wider range of articles from different websites and, most importantly, to design a dataset twice the size, which should help in training the large models that we use. Furthermore, to attempt to reduce the bias by training the model on this large dataset and then testing it on the test sets of each dataset separately.

When designing this dataset, the train, test, and validation sets of both datasets were concatenated separately using the concatenate function from the Hugging Face Datasets library to preserve the uniqueness and avoid undesirable phenomena, i.e., to prevent the model from training on documents that it would then have in the test set. Subsequently, a random shuffling of the data of each set was performed, and as a result, the documents of the individual dataset were mixed with each other. Overall, the dataset exceeds the number of 1.7 million Czech articles, which are divided as follows: 79 567 into the test set, 79 454 into the validation set, and 1 542 820 into the training set.

3.3.1 Dataset Statistics

The table 3.5 shows the statistics of the number of words in each section of the headline, abstract, and full text of the CNC-Sum dataset. As expected, most of the values remained within the ranges of both datasets. All the values of quartiles in the abstract have apparently decreased by a few units, probably caused by the union of a large number of abstracts with lower word counts. However, the average number of words in the abstract did not decrease significantly, most likely due to the presence of a few abstracts with notably large word counts.

[3] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`

	Q1	Median	Q3	Mean	Stddev
Headline	8	10	11	9.5	2.7
Abstract	32	40	50	42.2	15.5
Text	192	289	438	369.7	318.1

Table 3.5. Quantitative statistics of lengths of headlines, abstracts, and texts in words for the CNC-Sum dataset. Q1 and Q3 denote the first and the third quartile, respectively.

Chapter 4
Implementation

In this chapter, we introduce the background of the summarization pipeline and the way we built the individual parts of the pipeline. For the implementation, we used the Python programming language, the two main libraries from HuggingFace that are mentioned in this chapter, and then other notorious libraries such as Pytorch[1], NumPy[2], Pandas[3], etc.

4.1 HuggingFace Transformers

We took advantage of HuggingFace Transformers[30] library (HFT library) to facilitate the manipulation of models because this library provides state-of-the-art Transformer architectures[1] under a single API to simplify user access. Thus all the pre-trained models mentioned in the chapter 2 are downloadable in this library from Model Hub[4]. To unify, the models are defined by three main components - Tokenizer, Transformer, and Head. The Tokenizer is different for each model and crucial, especially since it converts the text into a numeric representation as expected by the model input. Transformers are implementations of given Transformer architectures (BART[18], BERT[10], etc.) incorporating their specifics. Heads are pre-implemented output layers for a specific task, such as Conditional Language Generation, Language modeling, etc., that are added on top of the base Transformer model.

4.2 HuggingFace Datasets Library

We also used the HuggingFace Datasets library[5] which provides easy access to public shared datasets designed primarily for NLP as well as local datasets stored in a local machine and also produces efficient data pre-processing with powerful built-in methods. With Apache Arrow[6] support and the use of mappings between RAM and the storage file system, HuggingFace Datasets is capable of working with large datasets that would overflow RAM and therefore are not fully loaded in memory. Datasets can be stored in various formats such as JSON, CSV, Arrow, or plain text.

In this thesis, the Datasets library is utilized primarily for the ease of use of datasets when training the models from the HFT library, as well as for the ability to work with large datasets, but also for efficient data processing using two powerful functions, Map and Filter.

[1] https://pytorch.org/
[2] https://numpy.org/
[3] https://pandas.pydata.org/
[4] https://huggingface.co/models?sort=downloads
[5] https://huggingface.co/docs/datasets/v2.1.0/en/index#datasets
[6] https://arrow.apache.org/

- The Map function iterates over each row of the entire dataset and applies a passed function to it separately, caching and storing the altered data. The whole process can be accelerated by batching the input that iterates over batches of rows with an established size instead of one row. Or moreover, the map function can be parallelized. This function is frequently used to tokenize data to feed the model, but the details are described in the next section.
- The Filter function iterates over batches or rows of the dataset as the map function does, but instead of modifying the data, it filters out rows that do not satisfy a condition specified in the passed function.

When comes to data examination, this library provides the ability to change the output format of the dataset entries. It can be set to Numpy, Pytorch tensors, as well as DataFrame from Pandas library, which brings a simple way of data exploration.

4.3 Summarization Pipeline

The summarization pipeline represents the entire process of a summarization task, from preparing data for training to generating summaries.

Thus, the first step is to prepare the data so that the model can be trained on it. Since the model cannot read the text as humans usually do, the raw text needs to be preprocessed into a numerical representation. This is done by a tokenizer, which needs to be consistent with the model used. The tokenizer could be loaded using a function *AutoTokenizer.from_pretrained* from the HFT library with a passed argument indicating the name of the model used. The core function of the tokenizer is to split the input into words or subwords referred to as tokens and then map the tokens to integers and add any special tokens the model needs. Special tokens can be sentence separators, language symbols, etc., depending on the input of the model. The tokenized input is denoted as input ids and is usually stored in tensors. To facilitate tokenization, the map function from the HuggingFace Datasets library comes into play. Since the map function was discussed in the 4.2 section, the tokenization is applied over the entire dataset. To effectively use the potential of the map function, batching is often used with the addition of truncation and padding strategy, which is handled by the tokenizer. Truncation is the act of shortening longer sequences to an established size, and padding ensures a rectangular shape of shorter sequences by adding extra padding tokens to a specified size. When padding is used, it is necessary to create an attention mask, which is a tensor of precisely the same shape as the input ids that specifies to which input id should or should not be attended by attention layers. This is primarily done due to the model's requirements because the model expects a batch of inputs in a rectangular shape.

After tokenization is applied, a model needs to be defined. With the name of the desired model checkpoint passed in, the *AutoModel.from_pretrained*[7] function from the HFT library allows to conveniently download the pre-trained model instead of training it from scratch. Moreover, depending on the downstream task, a different *AutoModel* class is used to automatically add a model head on top of it, usually consisting of several linear layers.

The next step is training, specially fine-tuning, which is the process of training a pre-trained model on some downstream tasks such as summarization, translation, etc. First comes the tricky part, all the hyperparameters for the training need to be tuned.

[7] `https://huggingface.co/docs/transformers/model_doc/auto#auto-classes`

This is done by setting them up in the *TrainingArguments* class from the HFT library. Afterward, a *Trainer* class from the HFT library is created by passing these arguments, the model, the tokenizer, and the tokenized train and validation set of a dataset. Finally, calling the *Trainer.train* method, which is an optimized training loop allowing additional features to be used, starts the training.

Once the training is completed, the learned model is ready for text summarization. The model class stores a generate function, which can predict the output from the input using the learned weights. This function allows adjusting various parameters of inference methods that can positively affect the output leading to significantly better summaries. Finally, the output of the generation function must be decoded by the tokenizer to make it human-readable.

4.4 Model

The use of an mBART[2] model for the summarization task was considered for several reasons. The main advantage is that it is a multilingual model with the Transformer architecture pre-trained in 25 languages, which produces a universal representation across these languages. In addition, one of the included languages is Czech, and as a result, the pre-trained auto-encoder has a tokenizer for the Czech language. Moreover, the model can be conveniently set up for the summarization task by defining the *MBartForConditionalGeneration*[8] class and its method *from_pretrained* from the HFT library and passing the model checkpoint name "*facebook/mbart-large-cc25*". By this feature, we construct the mBart model from the given checkpoint, which is described in the section 2.9, with a Conditional Generation head ready for fine-tuning.

4.5 Tokenizer

The tokenizer was downloaded using the *MBartTokenizerFast*[9] class from the HFT library and its *from_pretrained* method by providing the model's checkpoint name "*facebook/mbart-large-cc25*" as an argument. This tokenizer inherits from *PreTrainedTokenizerFast* class, which works similarly as described in the section above 4.3, but it is a part of fast tokenizers. These fast tokenizers are written in Rust programming language instead of Python and are therefore capable of tokenizing multi-gigabyte datasets in minutes. Furthermore, both additional method's arguments, the source language, and the target language were set to the Czech language token id to generate a Czech summary from a Czech article. The advantage of this tokenizer, besides its speed, is that it is already pre-trained on a large corpus and reaches a vocabulary size of 250 000. This tokenizer works based on SentencePiece[23] unsupervised text tokenizer and detokenzier, which considers texts as sequences of Unicode characters, replaces spaces with _, and does not depend on a language logic. The main feature of SentencePiece is reversibility, which means that the decoding of the tokens is done by concatenating them and replacing _ with spaces. SentencePiece implements two subword segmentation tokenizations, Byte-Pair Encoding (BPE) tokenization[14] and a unigram tokenization[31]. Since we are using the pre-trained tokenizer, we do not train this tokenizer from scratch, and we simply use the learned encodings.

[8] `https://huggingface.co/docs/transformers/main/model_doc/mbart#transformers.MBartForConditionalGeneration`

[9] `https://huggingface.co/docs/transformers/main/model_doc/mbart#transformers.MBartTokenizerFast`

- Byte-Pair Encoding tokenization - splits a word into characters and applies the merge rules learned during training.
- Unigram tokenization - finds the most probable split into tokens using scores learned during training.

Let's have an example to see how this mBART tokenizer works.

```
'Nejkulaťoulinkatější balon'
["_Nej", "kula", "ť", "ou", "link", "at", "ější", "_balon"]
[24079, 21260, 1981, 796, 10187, 257, 62137, 118188, 2, 250002]
'Nejkulaťoulinkatější balon</s>cs_CZ'
```

As we can see, the sentence is first tokenized into subwords using a subword segmentation tokenizer (BPE), where the space was replaced by the symbol _, and then converted to token ids. Each token id corresponds to the subword token except the last two, which indicate special tokens added by the tokenizer. The last row demonstrates the decoded tokens, where we can see that the special tokens are a sentence separator and a language id, respectively.

4.6 Data Preprocessing

Since we are dealing with modified tasks, we first merged the abstract section with the full text for the *headline + full text to abstract* task and the headline section with the full text for the *abstract + full text to headline* task as input text for each dataset. According to the paper[2], the model expects input text with split sentences using a sentence separator token. This was done by splitting the input text of each document into sentences, then inserting a sentence separator token between each sentence. Since we are dealing with a supervised sequence-to-sequence generation using the encoder-decoder architecture, we also need to tokenize target output text to train the decoder part. This is done by tokenizing the target sequences in a slightly different way (means a different order of special tokens) and then by shifting created labels with replaced padding tokens to the right as expected by the model. The tokenization is "*<tokens>* *<eos>* *<language code>*" for input language texts, and "*<language code>* *<tokens>* *<eos>*" for target language texts, where "*<eos>*" is an end of sentence token. We tokenized all datasets using the tokenizer4.5 with fixed sizes of truncation and padding regarding the specific summarization task. For the *headline + full text to abstract* task, truncation and padding were set to 512 tokens for the encoder input and 128 tokens for the decoder input. For the *abstract + full text to headline* task, truncation and padding were configured to 512 tokens for the encoder input and 64 tokens for the decoder input.

The following table 4.1 shows statistics of lengths of input ids and decoder input ids for each tokenized dataset after the truncation was performed. The values of the input ids show that in almost every dataset, we truncated more than half of the documents to 512 tokens, possibly losing some text information at the expense of learning speed. However, most documents contain essential information at the beginning of the text, so we believe that the truncation will not significantly impact the generation of summaries. In addition, the values of the decoder input ids were not significantly truncated, with only a tiny percentage of the documents, preserving the information for the target summaries.

SumeCzech		Q1	Median	Q3	Mean	Stddev
AT2H	input ids	507	512	512	484.9	58.6
	decoder ids	16	20	23	19.9	4.8
HT2A	input ids	454	512	512	471.4	72.4
	decoder ids	59	74	88	73.8	22.0
CNC		**Q1**	**Median**	**Q3**	**Mean**	**Stddev**
AT2H	input ids	366	512	512	437.9	103.8
	decoder ids	20	23	27	23.5	5.5
HT2A	input ids	309	486	512	410.1	123.1
	decoder ids	66	86	109	86.9	27.1
CNC-Sum		**Q1**	**Median**	**Q3**	**Mean**	**Stddev**
AT2H	input ids	444	512	512	464.3	84.8
	decoder ids	18	21	25	21.5	5.4
HT2A	input ids	389	512	512	444.5	102.5
	decoder ids	62	78	96	79.6	25.2

Table 4.1. Quantitative statistics of lengths of input ids and decoder input ids for each tokenized dataset, after truncation. Q1 and Q3 denote the first and the third quartile, respectively. AT2H and HT2A represents the summarization tasks *abstract + full text to headline* and *headline + full text to abstract*, respectively.

4.7 Training

We applied a Trainer class from the HFT library in the training phase, which contains a pre-programmed optimized training loop. For clarity and visualization of the learning process, we used the Weight and Biases[10] integration, which tracks the learning progress of the models.

We encountered a few problems during the training. We ran out of memory several times. After training all the models dealing with *headline + full text to abstract* task, we also found that when we separated the target texts for the decoder with a sentence token separator, the models tended to generate one max two long sentences connected by commas. So we then separated only the encoder input and left the decoder input only tokenized.

4.8 Inference

We implemented a Python script that loads the learned model and, according to the inference methods 2.10, generates summaries and computes ROUGE_{RAW} scores on them. For inference methods, we utilized the generate method, which is built into the model class from the HFT library, and different parameters of different inference methods can be tested in various combinations using *ParameterGrid*[11] from the *scikit-learn* library.

[10] `https://wandb.ai/site`

[11] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.ParameterGrid.html`

Chapter 5
Experiments

The main focus of this chapter is on training the summarization models, testing the parameters of the inference methods, and the final results on the test sets. In the first part, we present experimental settings, followed by the training process of our models. Then, we focus on the effects of various inference method parameters on the resulting summarizations we produced by our models learned on the SumeCzech and CNC validation sets to determine which summarization hyperparameters fit best for each model and dataset. The results of the ROUGE_{RAW} scores of generated summaries using the various inference hyperparameters are provided in the appendix B to maintain the clarity of this work. The best hyperparameters tested in the experiments on the validation sets were then used to generate summaries on the test sets of each dataset. In addition to validation inference testing, this chapter also presents the ROUGE_{RAW} scores on the test sets of all the datasets for the tasks we considered, as well as the results on tasks of the SumeCzech dataset proposed by the authors. At the end of this chapter, we provide some examples of the summaries generated by our models and a discussion regarding the experiments with the summary generation parameters and the results of the work. We use shortcuts for model names based on the name of the summarization task and the dataset the model was learned on, shown in the table 5.1.

Model name	Dataset	Task	Seen docs
AT2H-S	SumeCzech	abstract + full Text to headline	2576K
HT2A-S	SumeCzech	headline + full Text to headline	6928K
AT2H-C	CNC	abstract + full Text to headline	5984K
HT2A-C	CNC	headline + full Text to headline	3712K
AT2H-CS	CNC-Sum	abstract + full Text to headline	7936K
HT2A-CS	CNC-Sum	headline + full Text to headline	12896K

Table 5.1. Summary of the model names. Dataset indicates on which dataset the model was trained, and Task indicates for which task. Seen docs are the number of documents that the model has seen during training. For example, a model AT2H-S means Abstract + Full Text to Headline task performed on the SumeCzech dataset.

5.1 Experimental Settings

In these sections, we describe how we set up the hyperparameters for training. We then present what parameters of the inference methods 2.10 we used to generate summaries on the validation set, and finally, we explain how we measured the summaries against each other.

5.1.1 Training settings

We configured the training arguments as follows. The Adam optimizer with decoupled weight decay (AdamW), introduced in the paper *Decoupled weight decay regulariza-*

tion[32] by Ilya Loshchilov et al. was set with initial parameters β_1=0.9 , β_2=0.999 and ϵ=1e-8. The AdamW optimizer is a better variant of the Adam optimizer[33] modifying the weight decay[34] implementation by fixing it from the gradient update. Weight decay is a regularization technique that adds a penalty to the loss function by which it attempts to prevent the model from overfitting. We then assigned the batch size to 8 samples. The warm-up phase was established at 5000 steps, and the learning rate was adjusted to 3e-5 with a linear learning rate decay scheduling according to the fine-tuning of mBART25[2]. Then the model was configured for 16-bit floating-point precision (fp16) to speed up the calculation. Overall, we trained the models based on the cross-entropy loss calculated from the shifted labels and predicted tokens. The best models have been selected according to the lowest loss on the validation set computed after every 2000 steps. In setting the hyperparameters, we mainly followed the advice and settings of the supervisor, and since we are unable to perform the training multiple times and tune the hyperparameters mainly due to time constraints but also due to the carbon footprint, the selection of hyperparameters for training may not be as effective as it could be.

5.1.2 Inference settings

We tested the inference methods mentioned in the section 2.10 and were interested in how they would affect the generation of summaries measured by the ROUGE_{RAW} metric. We tried the Beam search with the beam sizes determined to be 1, 4, 6, 8, 12, and 16, both for the generation with and without sampling. We also applied the top-p sampling with p fixed at 0.28, 0.44, 0.55, 0.74, 0.84, 0.89, 0.92, 0.95, and also we performed the Top-k sampling with k values adjusted to 0, 20, 40, 50, 70, 80, 100. Each sampling was done with the temperature set to 0.7, 0.85, 0.89, 0.95, 1.1. In addition, we tried repetition penalty configured to 1.2 and 10.0. For each combination of the inference parameters, we generated summaries of 10240 documents from the validation sets, which corresponds to roughly one-third of the validation set of the CNC dataset and one-quarter of the validation set of the SumeCzech dataset.

We attempted to compare the generated summaries with each other using the ROUGE_{RAW} metric by assuming the generated summaries of one choice of inference parameters to be the system-generated summaries and calculating the ROUGE_{RAW} scores with the other generated summaries of other inference parameters that we considered to be the reference summaries. We followed this procedure for all of them. This idea comes from the paper *Texygen: A Benchmarking Platform for Text Generation Models*[35] introduced by Zhu et al., which presents a Self-BLEU measuring the diversity of documents in translation tasks. Since we use ROUGE_{RAW} scores, we have precision, recall, and f1 scores for each. Instead, we focus only on the recall score because the other scores can be easily calculated from recalls. Assuming that we have n documents with system-generated summaries and we compute recall scores for all documents relative to each other. We obtain an $n \times n$ matrix with ones on the diagonal, where row r is a document with summaries considered as system summaries, column c is a document with summaries considered as reference summaries, and $\text{cell}_{r,c}$ contains the three recall scores for ROUGE-1,-2,-L computed relative to r and c. When we transpose the matrix, we get precision scores with respect to r and c, and the f-scores thus become a symmetric matrix. In the Inference sections 2.10, we denote it as Self-Rouge.

5.2 Training

This section briefly describes the learning process of each model. Models with the suffix -S are trained on the SumeCzech dataset, with the suffix -C on the CNC dataset, and with the suffix -CS on the CNC-Sum dataset as described in the table 5.1. All models will be available on our website[1].

5.2.1 Hardware

We took advantage of the RCI, a center of scientific excellence in computer science and artificial intelligence, which also supports CTU students working on their theses. We used the RCI Cluster, a high-performance computing infrastructure providing CPU, GPU, and SMP nodes for use and data storage with a 100Gbit low-latency network. In this thesis, we primarily worked on GPU nodes, especially on NVIDIA Tesla A100 40GB with NVLink and on NVIDIA Tesla V100 32GB with NVLink, which both substantially accelerated the training of our models.

5.2.2 AT2H-S

We trained this AT2H model twice for 20 hours on 1x NVIDIA Tesla A100 40GB. The first part of the training with a total step count of 192,000 (192K) ran smoothly, and the loss continuously decreased, while in the second part of the training, the loss started to oscillate after reaching 322K steps. Thus, a total of 322K steps were run, which is equivalent to 3.0 epochs, and the total loss of the validation set reached 1,73 from the initial 2,62.

5.2.3 HT2A-S

We performed the training of this model once for 20 hours on 1x NVIDIA Tesla A100 40GB, twice for 20 hours on 1x NVIDIA Tesla V100 32GB, and then once for 20 hours on 4x NVIDIA Tesla A100 40GB. During training, we manually tuned the learning rate by decreasing it by a small value between each phase but in accordance with the linear learning rate scheduler. In total, 8 epochs were executed, corresponding to approximately 866K steps, and the final evaluation loss reached a figure of 2.03.

5.2.4 AT2H-C

We managed to train this AT2H model for 15 hours on 4x NVIDIA Tesla V100 32GB, for 10 hours on 4x NVIDIA Tesla A100 40GB, and an additional 20 hours on 1x NVIDIA Tesla A100 40GB. In addition, we tuned the learning rate during training. Overall, a total of 748K steps were performed, which corresponds to approximately 8.9 epochs. The final evaluation loss was reduced to a number of 1.67. In the final training phase, we observed that the evaluation loss oscillated significantly and tended to increase, which indicated overfitting.

5.2.5 HT2A-C

We trained this HT2A model three times for 20 hours on 1× NVIDIA Tesla A100 40GB. Between individual training phases, we adjusted the learning rate to a lower value to attempt to speed up the convergence while following the learning rate scheduler. In total, the model made 464K steps, which equals 5.5 epochs. The evaluation loss decreased approximately to the value of 1.97. Unfortunately, further attempts at training led to increasing and oscillating evaluation loss.

[1] `https://huggingface.co/ctu-aic`

5.2.6 AT2H-CS

This model was first trained for 40 hours on 1x NVIDIA Tesla A100 40GB, then 20 hours on 1x NVIDIA Tesla V100 32GB, and finally 20 hours on 4x NVIDIA Tesla A100 40GB. The model performed 992K steps equivalent to 5.1 epochs, and the evaluation loss decreased to approximately 1.61 from the initial 2.63.

5.2.7 HT2A-CS

We successfully trained this model first for 60 hours on 1x NVIDIA Tesla A100 40GB and then 40 hours on 4x NVIDIA Tesla A100 40GB. In addition, we tuned the learning rate during training. Overall, the evaluation loss was significantly reduced to roughly 1.96 from the initial 2.65, with the model performing 1 612K steps corresponding to approximately 8.4 epochs.

5.3 Inference

The following sections describe experiments testing various inference parameters to better generate summaries. We only performed these inference experiments on the validation sets of the SumeCzech and CNC datasets because we assume that models trained on the CNC-Sum dataset will have the same summarization properties as models learned on separate datasets. In addition, we have summarized the results of the ROUGE_{RAW} best inference parameters, which are listed in tables from B.1 to B.12 in the appendix, for clarity of the work, and also because these results are not as important as the results on the test sets.

5.3.1 Inference of AT2H models

This section tested inference for AT2H-S and AT2H-C models dealing with the AT2H task and compared generated summaries using Self-Rouge. We also provide the best results of AT2H inference parameters for each model in the tables from B.1 to B.6 in the appendix.

For generating summaries in the generation method, the minimum number of generated tokens is set to 10 by default. However, we also tested higher numbers obtained by subtracting the standard deviation from the average number of heading tokens of the validation sets. Thus, we also experimented with minimum token values of 15 and 18 for the SumeCzech and CNC datasets. By doing this, we aimed to test whether the generation of summaries is either somehow affected by the minimum length or whether the model can estimate the appropriate length based on the data provided. Since we are generating headlines that are intended to be concise, we set the repetition penalty to 10 to reduce the number of repeating tokens occurring in the generated summaries.

We first experimented with the beam search, which showed that a beam size of 4 with a minimum token count of 10 and no sampling produced the best results for both models. However, similar high-performance results were obtained for larger beam sizes demonstrated in the tables B.1 and B.2. For the SumeCzech model, the greedy search results (beam size 1) achieve almost the same score as the others. The CNC model, compared to the SumeCzech model, works better for larger beams, and this is also confirmed by its Self-Rouge, which showed significantly low diversity between beams of sizes 4, 12, and 16. Moreover, what we found interesting, the beam size of 6 was slightly different from the other beams in terms of diversity. Next, when examining the Self-Rouge, the results of both models revealed that the mutual recall scores were

reasonably high between each other, indicating that the generated summaries showed similarity and the different beam sizes produced relatively equal summaries. The greedy search showed considerable diversity from others, and its average recall was roughly 50 30 40 without sampling and 30 20 20 with sampling, but as for the other beam sizes, their recall was above 70 60 60 as well as the precision. Regarding beam size, closer beam sizes were more highly correlated than more distant ones. For example, beam size 16 with beam size 12 achieved recall 91 88 90 and precision 93 90 92, whereas with beam size 8 had recall 87 83 86 and precision 88 84 87, all without sampling. Furthermore, we observed a higher correlation among beam sizes that were produced without sampling, which is to be expected. From this, we can infer that sampling is more creative, but as the beam size increases, the diversity decreases.

We then tested top-k sampling, which performed well for both models, with the best result for $K = 20$, a temperature of 0.95, and a minimum token count of 10. Scores are demonstrated in the tables B.3 and B.4. All scores appeared fairly similar and decreased with increasing temperature and number of k. We were very surprised by the results of the last rows with a temperature of 1.1, which should flatten the distribution and lead to worse results. Examining this more closely, we found that this setting tended to generate longer sequences, which appeared to be good in recall score but not in precision. By examining the diversity of summaries under different top-k testing parameters, we found out that most lower k values have a higher recall with larger ones, but not vice versa. With a fixed k value, we can say that those with smaller temperatures show less diversity in contrast to the larger ones. We can also say that the difference in the minimum number of tokens does not play a role here. We just noticed that a smaller count with a large one exhibits higher recall than vice versa and hence has a larger recall than precision. Furthermore, with an established value of the temperature, we observed that the recalls are essentially similar regardless of the k values.

Finally, we experimented with top-p sampling, which yielded the best results for the SumeCzech model with p=1.0, a temperature of 0.89, and a minimum token count of 10, and for the CNC model with $p = 0.92$, a temperature of 0.89, and a minimum token count of 10, which are shown in the tables B.5 and B.6. We observed that top-p behaves the same as top-k in terms of Self-Rouge, except that lower values of p are more correlated with each other independently of the temperature value.

5.3.2 Inference of HT2A models

This section provides an overview of our experiments with the HT2A task, in which we tested inference for the HT2A-S and HT2A-C models, as well as a comparison of produced summaries. We collected the best inference results, which are demonstrated in the tables from B.7 to B.12 in the appendix.

We ran some experiments and found that the models often repeat tokens when generating summaries, so we tried adjusting the repetition penalty to 10 and 1.2 to see if this could affect the score. From previous experiments held on the AT2H problem, the default size of the minimum number of tokens worked out better because the model was probably able to estimate the required size of the summary from the given output; therefore, we left the minimum number of tokens set to the default value of 10.

Experimenting with beam search of various sizes, it turned out that a beam of size 4 without sampling was the best choice for both models, which is shown in the tables B.7 and B.8. Other beam sizes without sampling performed similarly well, while with sampling, the results only improved for larger beam size numbers. When considering

the repetition penalty, again, both models showed the best results for 4 beams. Nevertheless, the higher penalty did not allow essential words that were crucial to the topic to be repeated, resulting in a worse score. When investigated using Self-Rouge, we observed similar characteristics as described in the above section with a few differences. The greedy search showed significantly higher diversity than the others, for both with and without sampling. Also, the more extensive beam sizes had lower average recall among themselves. In addition, we observed that the precision of larger beam sizes with smaller ones is greater than their recall and vice versa.

As the next, we applied top-k sampling, which produced only one best result for both models shown in the tables B.9 and B.10 with set parameters to $k = 20$ and the temperature of 0.7, indicating that the models behaved mostly greedy. Overall, scores decreased as the temperature increased; also, for each temperature, better scores were obtained with the smaller k. Now the temperature of 1.1 showed a low score. When we examined diversity using Self-Rouge, we found that it behaves similarly to the behavior described in the section above, with a few differences. Here we observed that lower values of k have low diversity at a fixed temperature but start to become more diverse as the temperature increases. Furthermore, we observed that the overall recall is much smaller between each other compared to the AT2H models, most likely due to the greater length of the generated summaries.

Testing of top-p sampling performed the following best results shown in the tables B.11 and B.12. As can be deduced, the best results come for $p = 0.28$ with a low temperature of 0.75 that behaves mostly greedy. The value of p mainly influenced the scores, and thus temperature hardly mattered.

5.4 Test settings

From experiments conducted to investigate the best inference hyperparameters for generating summaries on the validation set, we derived which hyperparameters best fit each model. Now is the time to test our models on test sets to see how they perform, as demonstrated in the following sections 5.5 and 5.6. The evaluation focuses only on the SumeCzech test and out-of-domain test sets and the CNC test set, which contain 44 567, 44 976, and 35 000 documents, respectively. When evaluating SumeCzech tasks, we kept the same settings for generating summaries as for our tasks. We set the inference parameters for each model as follows.

For the AT2H models, we set temperature 0.89, beam size 4, top-k 80, and repetition penalty 10. Only the top-p values differed, which we adjusted to 0.92 or 1.0 for the CNC model and the SumeCzech model, respectively. We tried to configure these parameters according to their best results; however, we had to make a compromise between top-k and top-p values when we set the temperature according to the best results of top-p, and thus the value of k was adjusted to 80 for both regarding the selected temperature, which of course also does not yield bad results. For the AT2H-CS model, we set the hyperparameters to always generate according to the model learned on the dataset of the test set it was currently dealing with, which is not perfect. Nevertheless, we aimed to have the best performance for each test set separately.

The HT2A models were all tuned to top-p 0.92, temperature 0.95, beam size 4, top-k 40, and repetition penalty 1.2. Neither the temperature, p-value, nor k-value matched the previously mentioned best-fit values from the experiments run, and the reason for this is that the program was unable to run, most likely due to an implementation error on the HuggingFace side, as they use a deprecated function from PyTorch that causes

an error "*RuntimeError: probability tensor contains either 'inf', 'nan' or element < 0*". So we tried to set the smallest possible values without errors.

We defined the minimum number of generated tokens as 10 and enabled sampling in all cases. In addition, an effect of the individual hyperparameters on a validation set can be seen in the tables from B.1 to B.12.

5.5 Results

This section presents the results of the ROUGE_{RAW} scores based on the evaluation of the summaries generated using our learned models. For each test set of both datasets, we produced summaries using all the models for a given task to compare their results with each other. We first analyze the models dealing with the *Abstract + Full Text to Headline* (AT2H) task, whose results are shown in the following table 5.2.

SumeCzech test set	ROUGE_{RAW}-1			ROUGE_{RAW}-2			ROUGE_{RAW}-L		
	P	R	F	P	R	F	P	R	F
AT2H-S	27.44	25.43	25.77	11.27	10.46	10.54	24.97	23.15	23.44
AT2H-C	22.73	25.78	23.45	8.41	9.65	8.67	20.28	23.05	20.93
AT2H-CS	**28.49**	**25.96**	**26.53**	**11.95**	**10.89**	**11.07**	**25.95**	**23.65**	**24.16**
CNC test set	ROUGE_{RAW}-1			ROUGE_{RAW}-2			ROUGE_{RAW}-L		
	P	R	F	P	R	F	P	R	F
AT2H-S	29.86	24.45	26.02	11.82	9.92	10.38	26.88	22.04	23.42
AT2H-C	33.13	**33.05**	32.46	14.32	14.46	14.08	29.52	**29.46**	28.92
AT2H-CS	**34.49**	32.69	**32.92**	**15.20**	**14.62**	**14.58**	**30.87**	29.31	**29.48**

Table 5.2. ROUGE_{RAW} results on test sets of both datasets for *Abstract + Full Text to Headline* task (AT2H). The suffix after AT2H indicates the dataset on which the model is learned. S, C, and CS denote SumeCzech, CNC, and CNC-Sum datasets, respectively. P, R, and F denote precision, recall, and F1 scores. Scores in bold are the highest.

As can be inferred from the results, the model learned on the CNC-Sum dataset, which was developed by concatenating the two datasets, exhibits the best ROUGE_{RAW} scores in both cases. In the case of the SumeCzech test set, this model outperforms even the one that was learned only on documents from the SumeCzech dataset. It can be argued that we only trained the ATH2-S model (trained on SumeCzech) for 2576K documents, and therefore it does not achieve acceptable results. However, this fact is disproved by the other case, where testing is conducted on the CNC dataset because there, the model AT2H-C has been trained for roughly 5794K documents, which is more CNC documents than AT2H-CS performed, but its scores were again outperformed by the model AT2H-CS. It can also be noticed that the result of the model AT2H-C on the SumeCzech dataset is significantly lower than the score obtained by the model AT2H-S, which was learned on it, this is also true in reverse, but the model AT2H-CS is the opposite of this. Hence, for this task, we can argue that training the model on both datasets' concatenation helped it utilize the important textual information from both datasets.

Next, we discuss the results of the models for the *Headline + Full Text to Abstract* task. The following table 5.3 shows the evaluation of the individual model on both test domains. From the table, we can notice that the HT2A-CS model trained on

the CNC-Sum dataset again achieves good quality scores for both datasets. For the SumeCzech documents, the results of the HT2A-CS and HT2A-S models are relatively similar, but the HT2A-CS model performs better again, probably due to the training on a large dataset and using features and dependencies learned especially from the CNC data. Whereas for the CNC dataset, the model that is trained on this dataset performs better, but only by a small percentage. This behavior is possibly caused by the fact that the input sequences were not excessively truncated during tokenization in contrast to the SumeCzech data, as shown in the table 4.1, and most of the textual information was retained. The CNC model achieves the best overall f-score on its data, while the SumeCzech model performs similarly there in terms of precision, even though it was trained on a different dataset.

SumeCzech test set	$ROUGE_{RAW}$-1			$ROUGE_{RAW}$-2			$ROUGE_{RAW}$-L		
	P	R	F	P	R	F	P	R	F
HT2A-S	26.89	19.26	**21.81**	8.02	5.82	6.56	20.09	14.45	16.34
HT2A-C	24.52	**20.18**	21.01	6.83	5.80	5.96	17.94	**14.77**	15.37
HT2A-CS	**28.73**	18.40	21.76	**8.97**	**5.83**	**6.86**	**21.73**	13.99	**16.51**
CNC test set	$ROUGE_{RAW}$-1			$ROUGE_{RAW}$-2			$ROUGE_{RAW}$-L		
	P	R	F	P	R	F	P	R	F
HT2A-S	30.73	18.16	22.10	9.51	5.57	6.81	22.78	13.52	16.43
HT2A-C	30.36	**22.84**	**25.35**	10.57	**8.12**	**8.95**	22.62	**17.08**	**18.93**
HT2A-CS	**31.51**	21.57	24.90	**10.96**	7.63	8.76	**23.62**	16.22	18.71

Table 5.3. $ROUGE_{RAW}$ results on test sets of both datasets for *Headline + Full Text to Abstract* task (HT2A). The suffix after HT2A indicates the dataset on which the model is learned. S, C, and CS denote SumeCzech, CNC, and CNC-Sum datasets, respectively. P, R, and F denote precision, recall, and F1 scores. Scores in bold are the highest.

In addition, both tables show that models learned on the CNC dataset yield relatively high recall when tested on SumeCzech documents, which means that the produced summaries are more relevant to the reference summaries than vice versa for SumeCzech models.

5.6 Results on SumeCzech tasks

This section demonstrates the results of our learned models on tasks designed by the authors of SumeCzech. Thus, we consider the tasks *Text to Abstract* (T2A), *Abstract to Headline* (A2H), and *Text to Headline* (T2H). Intuitively, the models learned on our HT2A task deal with the T2A task and the AT2H models with the A2H and T2H tasks. We evaluated our models on both test and out-of-domain test sets of SumeCzech, whose $ROUGE_{RAW}$ scores are shown in the following table 5.4, which also displays the best results of models designed by the authors of SumeCzech[3] and the author of the bachelor thesis, Štěpán Műller[36], which we obtained from their website[2]. Brief description of their models:

- first - by SumeCzech, takes one or three first sentences as a summary regarding the task.

[2] `https://ufal.mff.cuni.cz/sumeczech`

- textrank[37] - by SumeCzech, selects one or three sentences based on the representation of the text as a network of sentences based on their similarity.
- tensor2tensor - by SumeCzech, an abstractive fine-tuned neural machine translation model of Vaswani et al.[1].
- Seq2seq-FT - by Műller, a seq2seq (bi-directional GRU as an encoder and a regular GRU as a decoder)[38] model with global attention and FastText embeddings
- Seq2seq-FT-NER - by Műller, a seq2seq (bi-directional GRU as an encoder and a regular GRU as a decoder)[38] model with global attention and FastText embeddings based on named entity recognition.

We should mention that our models were engaged in slightly different tasks that may have benefited their understanding of textual information by learning headlines when generating an abstract or also abstracts when producing a headline. On the other hand, these features may have disadvantaged them when they relied only on that particular part of the text.

We first present 5.4 an evaluation of the T2A task, where our SumeCzech model (HT2A-S) achieved good results and outperformed the models of the SumeCzech authors. It shows significantly higher f-scores in both test sets and up to twice as high in the ROUGE$_{RAW}$-2. However, the results also suggest that the model had a problem with the recall value in ROUGE$_{RAW}$-1 and ROUGE$_{RAW}$-L, which the textrank model significantly beat. This can be explained by the fact that the textrank model is extractive and chooses sentences directly from the input text, in contrast to our abstractive model building new ones. In comparison with our other models, they perform similarly, achieving a higher recall for the CNC model and a relatively overall higher f-score for the CNC-Sum model in the out-of-domain test set, probably caused by training on the CNC documents.

Next, we present 5.4 the results of the A2H task with the best f-score obtained by our SumeCzech model, which again shows a weakness in the recall values in all rouge scores. What is most likely evident here is that our model was used to generate a headline from a combination of abstract and text, which is missing in this task. At the same time, the best recall score was achieved by the first model selecting the first sentence as a summary, indicating that the extractive approach has a considerable effect on recall for this task. On the other hand, our SumeCzech model achieves the best precision in all cases. Our other models follow similar scores and are also likely affected by the smaller input in the test set.

In the T2H task 5.4, the feature of learning on larger documents composed of abstract and text and a better understanding of textual information is likely to be beneficial, as our models outperform all others in all the ROUGE$_{RAW}$ scores by several times in some places. The CNC-Sum model is the best in this task on both test sets, followed by the SumeCzech model and then the CNC model. However, the overall ROUGE$_{RAW}$ scores are lower than in the A2H task inferring that the abstract captures the headline more than the text. The SumeCzech model also has greater precision than recall indicating more concise summaries.

The CNC model results are the worst relative to the performance of our other models, but still outperforming the other models despite the CNC model has never seen the SumeCzech dataset. The CNC-Sum model is comparable, if not better, to the performance of the SumeCzech model, proving once again that the size and diversity of the dataset it was learned on is indeed crucial.

T2A test set	Rouge_{RAW}-1			Rouge_{RAW}-2			Rouge_{RAW}-L		
	P	R	F	P	R	F	P	R	F
first	13.1	17.9	14.4	1.9	2.8	2.1	8.8	12.0	9.6
textrank	11.1	**20.8**	13.8	1.6	3.1	2.0	7.1	**13.4**	8.9
tensor2tensor	13.2	10.5	11.3	1.2	0.9	1.0	10.2	8.1	8.7
HT2A-S	**22.9**	16.0	**18.2**	**5.7**	**4.0**	**4.6**	**16.9**	11.9	**13.5**
HT2A-C	20.7	17.1	17.7	4.8	4.1	4.2	15.0	12.4	12.8
HT2A-CS	24.0	15.0	17.9	6.2	4.0	4.7	18.0	11.3	13.4
T2A ood set	Rouge_{RAW}-1			Rouge_{RAW}-2			Rouge_{RAW}-L		
	P	R	F	P	R	F	P	R	F
first	11.1	17.1	12.7	1.6	2.7	1.9	7.6	11.7	8.7
textrank	09.8	**19.9**	12.5	1.5	3.3	2.0	6.6	**13.3**	8.4
tensor2tensor	12.5	9.4	10.3	0.8	0.6	0.6	9.8	7.5	8.1
HT2A-S	**23.0**	15.6	**17.9**	**6.1**	**4.2**	**4.8**	**17.1**	11.6	**13.3**
HT2A-C	20.4	18.3	18.2	5.3	4.7	4.7	14.8	13.2	13.2
HT2A-CS	24.5	15.6	18.3	6.9	4.4	5.2	18.3	11.7	13.7
A2H test set	Rouge_{RAW}-1			Rouge_{RAW}-2			Rouge_{RAW}-L		
	P	R	F	P	R	F	P	R	F
first	13.9	**23.6**	16.5	4.1	**7.4**	5.0	12.2	**20.7**	14.5
tensor2tensor	20.2	15.9	17.2	6.7	5.1	5.6	18.6	14.7	15.8
AT2H-S	**25.9**	18.6	**20.9**	**10.3**	7.3	**8.2**	**24.1**	17.3	**19.4**
AT2H-C	22.1	17.9	19.0	8.0	6.5	6.8	20.3	16.5	17.4
AT2H-CS	24.8	18.0	20.1	9.7	6.9	7.7	23.1	16.8	18.7
A2H ood set	Rouge_{RAW}-1			Rouge_{RAW}-2			Rouge_{RAW}-L		
	P	R	F	P	R	F	P	R	F
first	13.3	**26.5**	16.7	4.7	**10.0**	6.0	11.6	**23.3**	14.7
tensor2tensor	19.4	15.1	16.3	7.1	5.2	5.7	18.1	14.1	15.2
AT2H-S	**29.4**	21.3	**23.8**	**13.8**	9.7	**10.9**	**27.4**	19.8	**22.2**
AT2H-C	24.6	22.1	22.4	10.8	9.6	9.6	22.7	20.4	20.6
AT2H-CS	28.7	21.9	23.8	13.4	10.0	10.9	26.7	20.4	22.2
T2H test set	Rouge_{RAW}-1			Rouge_{RAW}-2			Rouge_{RAW}-L		
	P	R	F	P	R	F	P	R	F
textrank	6.0	16.5	8.3	0.8	2.3	1.1	5.0	13.8	6.9
Seq2seq-FT	15.4	13.7	14.1	2.4	2.1	2.1	13.9	12.4	12.8
Seq2seq-FT-NER	15.3	13.6	14.0	2.4	2.0	2.1	13.9	12.4	12.7
AT2H-S	**19.4**	**17.1**	**17.7**	**6.1**	**5.4**	**5.5**	**17.7**	**15.6**	**16.1**
AT2H-C	16.0	17.3	16.0	4.2	4.6	4.2	14.3	15.4	14.3
AT2H-CS	20.4	17.6	18.3	6.6	5.6	5.9	18.6	16.0	16.7
T2H ood set	Rouge_{RAW}-1			Rouge_{RAW}-2			Rouge_{RAW}-L		
	P	R	F	P	R	F	P	R	F
textrank	5.8	16.9	8.1	1.1	3.4	1.5	5.0	14.5	6.9
Seq2seq-FT	12.6	11.4	11.6	1.9	1.6	1.7	11.7	10.7	10.8
Seq2seq-FT-NER	13.0	11.6	11.9	1.9	1.7	1.7	12.0	10.8	11.0
AT2H-S	**20.8**	**18.8**	**19.2**	**8.0**	**7.2**	**7.3**	**19.2**	**17.4**	**17.7**
AT2H-C	17.2	18.2	17.0	6.0	6.4	5.9	15.7	16.6	15.5
AT2H-CS	22.0	18.9	19.7	8.8	7.4	7.7	20.5	17.5	18.3

Table 5.4. ROUGE_{RAW} scores for SumeCzech tasks. Along with our models, the ones with the best scores are shown here, which were designed by the authors of the SumeCzech paper and Štěpán Műller. The suffix after HT2A or AT2H indicates the dataset on which the model is learned. S, C, and CS denote SumeCzech, CNC, and CNC-Sum datasets, respectively. P, R, and F denote precision, recall, and F1 scores, respectively. T2A, A2H, and T2H stand for *Text to Abstract*, *Abstract to Text*, and *Text to Headline*, respectively. The bold scores are the highest, but considering models learned only on the SumeCzeh dataset.

5.7 Summaries

We provide a few examples of summaries from CNC and SumeCzech test sets generated by our models to see how they perform. The reference summarizations marked as "GOLD" are shown at the beginning, followed by summaries generated using our models, which we have translated into English for non-Czech speakers. More summaries are provided in the attached files C.

We present two shorter examples of summaries from the CNC test set for the *Headline + Full Text to Abstract* task in the following table 5.5. In the first example, all models captured the reference summarization reasonably well. Whereas the second example demonstrates a case where the reference text is trying to engage the reader and thus cannot be considered as a full-text summary and is more of a news text generation. However, in this case, our models summarized the text comparatively well with respect to what was written in the input text, but elsewhere they also generated engaging interrogative sentences with a question mark.

GOLD	Pátek se stal šťastným dnem pro Slováka z Košického kraje. V loterii Eurojackpot vyhrál neuvěřitelných 58 807 427,70 eur (1,6 miliardy Kč)! Stal se tak prvním Slovákem, který Eurojackpot vyhrál. Friday was a lucky day for a Slovak from the Košice district. He won a whopping €58 807 427.70 (CZK 1.6 billion) in the Eurojackpot lottery! He became the first Slovak, who won the Eurojackpot.
HT2A-C	Slovák vyhrál v pátečním losování Eurojackpotu 58 807 427,40 eur (1,6 miliardy Kč). Stal se tak prvním Slovákem, který vyhrál v loterii. A Slovak won €58 807 427.40 (CZK 1.6 billion) in Friday's EuroJackpot draw. He became the first Slovak, who won in the lottery.
HT2A-S	Slovák, který vsadil na Eurojackpot, vyhrál při losování 58 807 427,40 eur (1,6 miliardy Kč)! A Slovak who bet on the Eurojackpot won €58 807 427.40 (CZK 1.6 billion)!
HT2A-CS	V pátek se stal prvním Slovákem, který vyhrál v Eurojackpotu více než 1,6 miliardy korun! On Friday, he became the first Slovak, who won more than CZK 1.6 billion in EuroJackpot!
GOLD	Jeden z nejlepších hráčů Counter-Strike: Global Offensive Oleksandr „s1mple" Kostyliev, měl zabanovaný účet v konkurenční hře Valorant. Čím si to zasloužil? One of the best Counter-Strike: Global Offensive players, Oleksandr "s1mple" Kostyliev, had his account banned in the competitive game Valorant. What did he do to deserve it?
HT2A-C	S1mple je ukrajinský hráč, který se proslavil svými kontroverzními výroky o Valorantu. S1mple is a Ukrainian player who became famous for his controversial statements about Valorant.
HT2A-S	S1mple byl jedním z nejpopulárnějších hráčů Counter-Strike: Global Offensive (CS:GO). Na streamu Valorant válcoval protihráčům a předvedl několik skvělých herních momentů. S1mple was one of the most popular players of Counter-Strike: Global Offensive (CS:GO). On the Valorant stream, he rolled his opponents and showed off some great gameplay moments.
HT2A-CS	S1mple je jedním z nejpopulárnějších hráčů Counter-Strike: Global Offensive. Na streamu s1mple tvrdil, že je pro něj Valorant až moc snadný. S1mple is one of the most popular players of Counter-Strike: Global Offensive. On the stream, s1mple claimed that Valorant is too easy for him.

Table 5.5. Two examples of CNC test summarizations. GOLD indicates the reference summary, rows with summaries under the GOLD are generated using the appropriate model. The suffix after HT2A denotes the dataset on which the model is learned. S, C, and CS denote SumeCzech, CNC, and CNC-Sum datasets, respectively.

The following table 5.6 contains our summaries from the SumeCzech test set considering the *Abstract + Full Text to Headline* task. The first example shows that the models' summaries captured the reference text reasonably well, except for the event's location, even though it occurred in the input text. The next example shows that the CNC model produced a headline that was as eye-catching as the reference one. The other models also produced essentially creative texts. And in the last example, it can be seen that all the models started with the name of the country Sri Lanka, which probably had a considerably high probability, and then proceeded generatively, except the SumeCzech model, which copied the rest of the sentence from the text.

GOLD	Tým studentů ČVUT představil svou novou formuli úspěšnou na Hockenheimu A team of CTU students introduced their new formula successful at Hockenheim
AT2H-C	Tým CTU CarTech představil svou formuli FS.02 CTU CarTech team introduced its formula FS.02
AT2H-S	ČVUT představil formuli, která byla nejlepší na světě CTU introduced a formula, which was the best in the world
AT2H-CS	Studenti ČVUT představili svou novou formuli CTU students introduced their new formula
GOLD	Kde je umění za půl miliardy dolarů z loupeže století? Ani na smrtelné posteli to nechce mafián říct Where's the half-billion-dollar art from the heist of the century? Even on his deathbed, the mobster doesn't want to say it
AT2H-C	Kdo je muž podezřelý z největší krádeže umění v Americe? Who is the man suspected of the biggest art theft in America?
AT2H-S	Ukradené umění je záhadou. Policie tápe, kdo ho ukradl Stolen art is a mystery. Police wonder who stole it
AT2H-CS	Krádež umění za miliardy. Muž, který prodal zbraň vrahovi, nepromluví Theft of art of billions. The man who sold the gun to the killer won't talk
GOLD	Srí Lanka – exotika za rohem Sri Lanka - exotic round the corner
AT2H-C	Srí Lanka: Toulky exotickými skvosty indického ostrova Sri Lanka: Tours of the exotic gems of the Indian island
AT2H-S	Srí Lanka: Požehnaná zářící země Sri Lanka: Blessed shining land
AT2H-CS	Srí Lanka: ostrov, který vypadá jako z Indie Sri Lanka: island that looks like from India

Table 5.6. Three examples of SumeCzech test summarizations. GOLD indicates the reference summary, rows with summaries under the GOLD are generated using the appropriate model. The suffix after AT2H denotes the dataset on which the model is learned. S, C, and CS denote SumeCzech, CNC, and CNC-Sum datasets, respectively.

During the examination of the produced summaries, we observed factual errors more often than grammatical ones. The models mainly had a problem when generating sports scores, ages, percentages, or any other numbers, which they often mixed up. For instance, when generating sports scores, the model was not able to assign the correct score to the right team, so the summaries often showed that the team that actually lost won. We also detected that the models often mixed up the names of people in articles, creating nonsensical names of non-existent people, or even, for example, in reports from police investigations, they sometimes swapped the participants with the leading actor, resulting in giving false information. In practical use, these errors could become fatal and seriously misinform readers. Unfortunately, the ROUGE_{RAW} metric does not address these defects, and thus a new metric should be designed that

would take factuality into account. For future work that would focus on the factuality, the paper[39], which addresses the shortcomings of the ROUGE metric and suggests possible solutions for measuring consistency, could provide a good starting point.

5.8 Discussion

By experimenting with different inference methods and their parameters, we were able to derive which summary generation hyperparameters work best on our datasets. We found that the model is capable of estimating the minimum length of the summary by itself. Furthermore, excellent results were obtained from beam size 4 and estimated up to beam size 8. While we did not test all beam sizes in this range, it appeared that larger beam sizes tend to generate similar if not worse summarizations, varying only in a few words. A beam search of size one (greedy search) may not be a bad choice to quickly see how a model produces, but its ROUGE_{RAW}-2 scores are fundamentally smaller. Generally, summaries produced using the beam search algorithm seem to be more human-written and not as convoluted. Concerning top-k and top-p sampling, the summaries generated using these methods are more creative but often syntactically and grammatically incorrect, unlike the beam search algorithm. For top-k, we found that the summarizations generated by k values differing by tens are considerably similar when we use a low temperature. Higher values of k could affect the resulting summarizations, but we did not test this. When we utilized top-p sampling, lower values of p produced similar summaries in terms of diversity but independent of the temperature value. It would be worth testing top-p and top-k sampling together with a higher number of beams because beam search looks for the most likely summary, whereas top samplings select the most likely word at a given step.

Focusing on our tasks, the models performing on the test set of a dataset on which they were trained produced high-quality results in terms of ROUGE_{RAW} scores. However, they had lower but similar scores on the opposite dataset. The SumeCzech model mainly produced higher precision than the recall, and for the CNC model vice versa. Whereas the CNC-Sum model learned on both datasets performed best on test sets of both tasks showing that it was able to learn the textual information of both datasets.

For SumeCzech tasks, our models outperformed most of the best results to date, leading to state-of-the-art results on the SumeCzech dataset. For the *Text to Abstract* and *Abstract to Headline* tasks, we achieved the best results for all ROUGE scores except recall, where we were beaten by the extractive first and textrank models. However, for the *Text to Headline* task, we achieved state-of-the-art results with all models for all ROUGE_{RAW} scores. It also appears that the utilized pre-trained mBART model and its architecture indeed contributed to understanding the dependencies of textual information because our model, which had never seen the SumeCzech data and was only trained on documents from the CNC dataset, also achieved state-of-the-art results. Furthermore, it showed that the model learned on both datasets performed considerably well on ROUGE_{RAW} scores and achieved better scores than the model learned only on the SumeCzech dataset, and this also proves that having a large dataset is crucial for such a large model.

Chapter 6
Conclusion

In this work, we have presented recently published models derived from the Transformer architecture. Subsequently, we described the three Czech news datasets we used during the experiments. We have cleaned the private CNC dataset from textual inconsistencies that could have affected the training of the models. Furthermore, we have presented a large dataset CNC-Sum, which was developed by concatenating the CNC dataset and the SumeCzech dataset. This was aimed to test whether model learning is affected by the dataset size, which has been shown to be crucial. In addition, we trained a total of six pre-trained multilingual encoder-decoder Transformer-based models, mBART, on our datasets for two tasks. With the trained models, we studied the impact of different inference method parameters on the generated summaries, which were then compared with each other using the diversity measured by the ROUGE_{RAW} metric. We selected the best inference hyperparameters and applied our models to the test data, achieving state-of-the-art results on all tasks of the SumeCzech dataset. Experiments showed that the model architecture helped to improve text understanding and the model learned on both datasets significantly dominated the others. For our tasks, this model again proved best, followed by the model learned on the dataset of the current test set. We found that, with few exceptions, the summaries produced were legible and resembled human writing but contained occasional factual errors.

References

[1] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, ukasz Kaiser, and Illia Polosukhin. *Attention is All you Need*. Available from `https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

[2] Liu, Yinhan, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. *Multilingual denoising pre-training for neural machine translation. Transactions of the Association for Computational Linguistics*. Available from `https://arxiv.org/pdf/2001.08210.pdf`.

[3] Straka, Milan, Nikita Mediankin, Tom Kocmi, Zdeněk Žabokrtský, Vojtěch Hudeček, and Jan Hajič. *SumeCzech: Large Czech News-Based Summarization Dataset. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Available from `https://www.aclweb.org/anthology/L18-1551.pdf`.

[4] a.s., CZECH NEWS CENTER. *Czech News Center, O nás*. Available from `https://www.cncenter.cz/o-nas`.

[5] Hopfield, John. *Neural Networks and Physical Systems with Emergent Collective Computational Abilities*. Available from DOI 10.1073/pnas.79.8.2554.

[6] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. Available from DOI 10.48550/ARXIV.1409.0473. Available from `https://arxiv.org/abs/1409.0473`.

[7] Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. Available from DOI 10.48550/ARXIV.1508.04025. Available from `https://arxiv.org/abs/1508.04025`.

[8] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. Available from DOI 10.1109/CVPR.2016.90. Available from `https://ieeexplore.ieee.org/document/7780459`.

[9] Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. Available from DOI 10.48550/ARXIV.1607.06450. Available from `https://arxiv.org/abs/1607.06450`.

[10] Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. Available from DOI 10.48550/ARXIV.1810.04805. Available from `https://arxiv.org/abs/1810.04805`.

[11] Li, Junyi, Tianyi Tang, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. *A Survey of Pretrained Language Models Based Text Generation*. Available from DOI

10.48550/ARXIV.2201.05273. Available from https://arxiv.org/abs/2201.05273.

[12] Liu, Yang, and Mirella Lapata. *Text Summarization with Pretrained Encoders*. Available from DOI 10.48550/ARXIV.1908.08345. Available from https://arxiv.org/abs/1908.08345.

[13] Liu, Yinhan, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. Available from DOI 10.48550/ARXIV.1907.11692. Available from https://arxiv.org/abs/1907.11692.

[14] Sennrich, Rico, Barry Haddow, and Alexandra Birch. *Neural Machine Translation of Rare Words with Subword Units*. Available from DOI 10.18653/v1/P16-1162. Available from https://aclanthology.org/P16-1162.

[15] Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. *Improving language understanding by generative pre-training (2018)*.

[16] Raffel, Colin, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. Available from DOI 10.48550/ARXIV.1910.10683. Available from https://arxiv.org/abs/1910.10683.

[17] Zhang, Jingqing, Yao Zhao, Mohammad Saleh, and Peter J. Liu. *PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization*. Available from DOI 10.48550/ARXIV.1912.08777. Available from https://arxiv.org/abs/1912.08777.

[18] Lewis, Mike, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. Available from DOI 10.48550/ARXIV.1910.13461. Available from https://arxiv.org/abs/1910.13461.

[19] Qi, Weizhen, Yu Yan, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. *ProphetNet: Predicting Future N-gram for Sequence-to-Sequence Pre-training*. Available from DOI 10.48550/ARXIV.2001.04063. Available from https://arxiv.org/abs/2001.04063.

[20] Puspitaningrum, Diyah. *A Survey of Recent Abstract Summarization Techniques*. Available from DOI 10.48550/ARXIV.2105.00824. Available from https://arxiv.org/abs/2105.00824.

[21] Hendrycks, Dan, and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. Available from DOI 10.48550/ARXIV.1606.08415. Available from https://arxiv.org/abs/1606.08415.

[22] Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and others. Language models are unsupervised multitask learners. *OpenAI blog*. 2019, Vol. 1, No. 8, pp. 9.

[23] Kudo, Taku, and John Richardson. *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*. Available

from DOI 10.48550/ARXIV.1808.06226. Available from https://arxiv.org/abs/1808.06226.

[24] Freitag, Markus, and Yaser Al-Onaizan. *Beam Search Strategies for Neural Machine Translation*. Available from DOI 10.18653/v1/w17-3207. Available from https://doi.org/10.18653%2Fv1%2Fw17-3207.

[25] Keskar, Nitish Shirish, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. *CTRL: A Conditional Transformer Language Model for Controllable Generation*. Available from DOI 10.48550/ARXIV.1909.05858. Available from https://arxiv.org/abs/1909.05858.

[26] Ackley, David H., Geoffrey E. Hinton, and Terrence J. Sejnowski. *A learning algorithm for boltzmann machines*. Available from DOI https://doi.org/10.1016/S0364-0213(85)80012-4. Available from https://www.sciencedirect.com/science/article/pii/S0364021385800124.

[27] Fan, Angela, Mike Lewis, and Yann Dauphin. *Hierarchical neural story generation*. Available from https://arxiv.org/pdf/1805.04833.pdf.

[28] Holtzman, Ari, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. *The Curious Case of Neural Text Degeneration*. Available from DOI 10.48550/ARXIV.1904.09751. Available from https://arxiv.org/abs/1904.09751.

[29] Lin, Chin-Yew. *ROUGE: A Package for Automatic Evaluation of Summaries. In Text summarization branches out*. Available from https://aclanthology.org/W04-1013.pdf.

[30] Wolf, Thomas, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. *Transformers: State-of-the-Art Natural Language Processing*. Available from DOI 10.18653/v1/2020.emnlp-demos.6. Available from https://aclanthology.org/2020.emnlp-demos.6.

[31] Kudo, Taku. *Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates*. Available from DOI 10.48550/ARXIV.1804.10959. Available from https://arxiv.org/abs/1804.10959.

[32] Loshchilov, Ilya, and Frank Hutter. *Decoupled weight decay regularization*. Available from https://arxiv.org/abs/1711.05101.pdf.

[33] Kingma, Diederik P, and Jimmy Ba. *Adam: A method for stochastic optimization*. Available from https://arxiv.org/abs/1412.6980.pdf.

[34] Loshchilov, Ilya, and Frank Hutter. *Fixing Weight Decay Regularization in Adam*. Available from https://openreview.net/forum?id=rk6qdGgCZ.

[35] Zhu, Yaoming, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. *Texygen: A Benchmarking Platform for Text Generation Models*. Available from DOI 10.48550/ARXIV.1802.01886. Available from https://arxiv.org/abs/1802.01886.

[36] Műller, Štěpán. *Text Summarization Using Named Entity Recognition*. Available from https://dspace.cvut.cz/handle/10467/87671.

[37] Mihalcea, Rada, and Paul Tarau. *TextRank: Bringing Order into Text*. Available from https://aclanthology.org/W04-3252.

[38] Cho, Kyunghyun, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. Available from DOI 10.48550/ARXIV.1406.1078. Available from https://arxiv.org/abs/1406.1078.

[39] Fabbri, Alexander R., Wojciech Kryściński, Bryan McCann, Caiming Xiong, Richard Socher, and Dragomir Radev. *SummEval: Re-evaluating Summarization Evaluation*. Available from DOI 10.48550/ARXIV.2007.12626. Available from https://arxiv.org/abs/2007.12626.

Appendix A
Acronyms

NLP	Natural Language Processing
NLG	Natural Language Generation
RNN	Recurrent neural networks
BERT	Bidirectional Encoder Representations from Transformers
GPT	Generative Pre-trained Transformer
BART	Bidirectional Auto-Regressive Transformers
mBART	Multilingual Bidirectional Auto-Regressive Transformers
GeLU	Gaussian Error Linear Unit
ReLU	Leaky Rectified Linear Unit
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
CNC	Czech News Center
OOD	Out-of-domain test set
CNC-Sum	A dataset concatenated from CNC and SumeCzech datasets
HFT	HuggingFace Transformers
BPE	Byte-Pair Encoding
AT2H	Abstract + Full Text to Headline task
HT2A	Headline + Full Text to Abstract task
A2H	Abstract to Headline task
T2H	Text to Headline task
T2A	Text to Abstract task
AT2H-S	A model learned on the SumeCzech dataset for the AT2H task
HT2A-S	A model learned on the SumeCzech dataset for the HT2A task
AT2H-C	A model learned on the CNC dataset for the AT2H task
HT2A-C	A model learned on the CNC dataset for the HT2A task
AT2H-CS	A model learned on the CNC-Sum dataset for the AT2H task
HT2A-CS	A model learned on the CNC-Sum dataset for the HT2A task

Appendix B
Inference results

In this section, several tables of the best inference parameter testing validation results are presented. Explanatory notes: L denotes a minimum number of tokens, B indicates the beam size, S stands for the sampling (F=False, T=True), K is the value of k in the top-k sampling, and P is the value of p in the top-p sampling. P, R, and F denote precision, recall and F1 scores, respectively. Scores in bold are the highest.

B.1 Inference results of AT2H models

Inference param	$Rouge_{RAW}$-1			$Rouge_{RAW}$-2			$Rouge_{RAW}$-L		
	P	R	F	P	R	F	P	R	F
S=F,L=10,B=1	24.98	27.11	25.40	9.18	10.00	9.30	22.12	24.01	22.48
S=F,L=10,B=4	**26.26**	28.13	**26.53**	**10.71**	11.47	**10.77**	**23.61**	25.31	**23.85**
S=F,L=15,B=6	24.31	28.97	25.84	9.49	**11.57**	10.14	21.64	25.93	23.05
S=F,L=10,B=12	25.85	28.00	26.26	10.63	11.54	**10.77**	23.26	25.23	23.64
S=T,L=10,B=4,	26.05	27.94	26.32	10.42	11.19	10.49	23.30	24.98	23.52
S=T,L=15,B=6	24.05	**29.19**	25.76	9.25	11.55	9.98	21.33	**26.03**	22.89

Table B.1. Rouge scores with the different beam sizes for the SumeCzech AT2H-S model.

Inference param	$Rouge_{RAW}$-1			$Rouge_{RAW}$-2			$Rouge_{RAW}$-L		
	P	R	F	P	R	F	P	R	F
S=F,L=10,B=4	**32.27**	35.47	**33.15**	14.25	15.60	14.59	**28.60**	31.42	**29.37**
S=F,L=10,B=6	32.11	35.52	33.09	**14.25**	15.70	**14.64**	28.48	31.48	29.33
S=F,L=10,B=16	31.81	35.38	32.89	14.17	**15.70**	14.60	28.21	31.38	29.16
S=T,L=10,B=4	31.97	32.33	32.89	13.85	15.27	14.20	28.26	31.22	29.05
S=T,L=18,B=12	30.94	**35.95**	32.63	13.25	15.63	14.03	27.22	**31.71**	28.73

Table B.2. Rouge scores with the different beam sizes for the CNC AT2H-C model.

Inference param	$Rouge_{RAW}$-1			$Rouge_{RAW}$-2			$Rouge_{RAW}$-L		
	P	R	F	P	R	F	P	R	F
L10,T0.89,K100	**27.75**	25.51	25.96	11.18	10.28	10.41	25.11	23.10	23.49
L10,T0.89,K80	27.69	25.56	25.95	**11.21**	10.36	10.46	**25.14**	23.21	23.55
L10,T0.95,K20	27.43	26.43	**26.29**	11.09	10.74	**10.60**	24.81	23.94	**23.78**
L15,T1.1,K20	23.91	29.33	25.73	9.09	**11.39**	9.83	21.08	25.96	22.70
L15,T1.1,K50	23.90	**29.36**	25.74	9.01	11.32	9.76	21.12	**26.06**	22.78

Table B.3. Rouge scores with the different values of k for the SumeCzech AT2H-S model.

Inference param	Rouge_{RAW}-1			Rouge_{RAW}-2			Rouge_{RAW}-L		
	P	R	F	P	R	F	P	R	F
L10,T0.89,K100	33.60	33.26	32.77	**14.74**	14.68	14.40	29.90	29.60	29.15
L10,T0.89,K80	**33.61**	33.17	32.72	14.69	14.59	14.32	**29.98**	29.61	29.19
L10,T0.95,K20	33.45	34.03	**33.08**	14.61	14.96	**14.47**	29.73	30.27	**29.40**
L18,T1.1,K40	29.75	**36.50**	32.16	12.46	**15.49**	13.51	25.94	31.90	28.05
L18,T1.1,K50	29.63	36.46	32.06	12.45	15.43	13.49	25.88	**31.92**	28.02

Table B.4. Rouge scores with the different values of k for the CNC AT2H-C model.

Inference param	Rouge_{RAW}-1			Rouge_{RAW}-2			Rouge_{RAW}-L		
	P	R	F	P	R	F	P	R	F
L10,T0.89,P1.0	**27.65**	25.47	25.88	**11.18**	10.29	10.41	**25.09**	23.13	23.49
L10,T0.95,P1.0	27.25	26.30	26.13	11.03	10.70	**10.56**	24.70	23.85	**23.68**
L10,T1.0,P1.0	26.06	27.99	**26.34**	10.43	11.19	10.48	23.31	25.03	23.54
L10,T1.1,P0.74	24.70	**29.13**	26.11	9.51	**11.31**	10.04	21.85	**25.81**	23.10

Table B.5. Rouge scores with the different values of p for the SumeCzech AT2H-S model.

Inference param	Rouge_{RAW}-1			Rouge_{RAW}-2			Rouge_{RAW}-L		
	P	R	F	P	R	F	P	R	F
L10,T0.89,P0.92	**33.57**	33.50	32.87	**14.64**	14.72	14.36	**29.90**	29.84	29.27
L10,T0.95,P0.92	33.32	34.01	33.00	14.54	14.95	**14.42**	29.57	30.23	**29.30**
L10,T1.0,P1.0	32.07	35.46	**33.01**	13.87	15.35	14.26	28.27	31.28	29.10
L18,T1.1,P0.74	29.69	36.44	32.09	12.39	**15.38**	13.42	25.88	31.84	28.00
L18,T1.1,P0.84	29.72	**36.46**	32.12	12.38	15.36	13.41	25.92	**31.87**	28.03

Table B.6. Rouge scores with the different values of p for the CNC AT2H-C model.

B.2 Inference results of HT2A models

Inference param	Rouge_{RAW}-1			Rouge_{RAW}-2			Rouge_{RAW}-L		
	P	R	F	P	R	F	P	R	F
S=F,L=10,B=4	**24.22**	22.14	**22.25**	**7.37**	6.59	**6.70**	**17.75**	16.18	**16.28**
S=T,L=10,B=4	24.12	22.10	22.21	7.12	6.42	6.51	17.54	16.03	16.12
S=T,L=10,B=12	22.73	22.83	21.75	6.94	**6.71**	6.52	16.63	16.55	15.84
S=T,L=10,B=16	22.19	**22.96**	21.56	6.77	6.68	6.42	16.19	**16.61**	15.66

Table B.7. Rouge scores with the different beam sizes for the SumeCzech HT2A-S model.

Inference param	Rouge_{RAW}-1			Rouge_{RAW}-2			Rouge_{RAW}-L		
	P	R	F	P	R	F	P	R	F
S=F,L=10,B=1,	23.94	24.07	23.07	6.50	6.74	6.38	16.33	16.43	15.75
S=F,L=10,B=4,	**28.10**	24.97	**25.65**	**9.88**	8.72	**9.01**	**20.76**	18.43	**18.94**
S=T,L=10,B=4,	27.92	24.96	25.58	9.69	8.63	8.88	20.58	18.38	18.85
S=T,L=10,B=8,	27.65	**25.11**	25.45	9.83	**8.74**	8.98	20.51	**18.54**	18.84

Table B.8. Rouge scores with the different beam sizes for the CNC HT2A-C model.

Inference param	Rouge_{RAW}-1			Rouge_{RAW}-2			Rouge_{RAW}-L		
	P	R	F	P	R	F	P	R	F
L10,T0.75,K20	**18.26**	**20.92**	**18.77**	**3.56**	**4.06**	**3.65**	**12.13**	**13.77**	**12.40**
L10,T0.95,K40	15.95	19.79	16.98	2.71	3.38	2.88	10.42	12.81	11.03

Table B.9. Rouge scores with the different values of k for the SumeCzech HT2A-S model.

Inference param	Rouge_{RAW}-1			Rouge_{RAW}-2			Rouge_{RAW}-L		
	P	R	F	P	R	F	P	R	F
L10,T0.75,K20	**20.33**	**22.92**	**20.61**	**4.95**	**5.66**	**5.07**	**13.66**	**15.29**	**13.81**
L10,T0.95,K40	17.97	22.05	18.93	4.06	4.95	4.27	11.92	14.44	12.48

Table B.10. Rouge scores with the different values of k for the CNC HT2A-C model.

Inference param	Rouge_{RAW}-1			Rouge_{RAW}-2			Rouge_{RAW}-L		
	P	R	F	P	R	F	P	R	F
L10,T0.7,P0.28	**21.47**	22.10	**20.98**	**4.94**	**5.11**	**4.83**	**14.55**	14.86	**14.15**
L10,T0.85,P0.28	21.14	**22.19**	20.85	4.79	5.06	4.73	14.31	**14.90**	14.04
L10,T0.95,P0.92	15.19	19.49	16.43	2.64	3.38	2.85	9.99	12.70	10.75

Table B.11. Rouge scores with the different values of p for the SumeCzech HT2A-S model.

Inference param	Rouge_{RAW}-1			Rouge_{RAW}-2			Rouge_{RAW}-L		
	P	R	F	P	R	F	P	R	F
L10,T0.7,P0.28	**23.76**	24.02	**22.95**	**6.46**	6.72	**6.35**	**16.22**	16.37	**15.66**
L10,T0.85,P0.28	23.65	**24.11**	22.94	6.45	**6.74**	6.34	16.12	**16.37**	15.61
L10,T0.95,P0.92	18.62	22.29	19.38	4.25	5.15	4.46	12.41	14.70	12.85

Table B.12. Rouge scores with the different values of p for the CNC HT2A-C model.

Appendix C
Attached files

This section describes the structure of the attached files. Models with learned weights are not included due to its size, but can be downloaded from the web site `https://huggingface.co/ctu-aic`.

`readme.pdf`	file with contents description.
`data`	directory with several examples of SumeCzech summaries from the test set.
`src/clean_cnc_dataset.py`	script for cleaning CNC dataset.
`src/concatenate_datasets.py`	script for concatenating datasets and creating test, train, and validation splits.
`src/tokenize_dataset.py`	script for tokenizing dataset.
`src/train.py`	script for training models.
`src/test_inference.py`	script for testing parameters of inference method on learned models.
`src/Summarizer.ipynb`	Jupyter Notebook file for generating summaries from arbitrary Czech texts using learned models.