University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Master's thesis

Summarization of News Articles

Plzeň 2022

Seják Michal

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Michal SEJÁK**
Osobní číslo:	**A20N0106P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Téma práce:	**Sumarizace novinových článků**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Prostudujte oblast vícedokumentové sumarizace se zaměřením na využití moderních modelů neuronových sítí a možnosti hodnocení kvality sumarizace.
2. Na základě předchozí studie zvolte techniku sumarizace.
3. Připravte pro realizaci této techniky trénovací data z článků ČTK a jejich shrnutí, které Vám budou poskytnuty.
4. S pomocí těchto dat implementujte sumarizační systém využívající neuronové sítě pro zpracování přirozeného jazyka.
5. Zvolte vhodnou metriku kvality výstupu a kriticky zhodnoťte dosažené výsledky.

Rozsah diplomové práce:	**doporuč. 50 s. původního textu**
Rozsah grafických prací:	**dle potřeby**
Forma zpracování diplomové práce:	**tištěná/elektronická**
Jazyk zpracování:	**Angličtina**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce:	**Ing. Jakub Sido** Nové technologie pro informační společnost
Datum zadání diplomové práce:	**10. září 2021**
Termín odevzdání diplomové práce:	**19. května 2022**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 11. října 2021

Declaration

Acknowledgment

I want to express my gratitude towards Ing. Jakub Sido for his invaluable guidance and supervision throughout the creation of this thesis and F. Feorh for proofreading it. Without the contribution of these two people, it would be of much lower quality than it is today. Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA CZ LM2018140) supported by the Ministry of Education, Youth and Sports of the Czech Republic.

Seják Michal

Abstract

Automatic text summarization is an important NLP task with many applications. Our particular area of focus is summarization of news articles. We introduce a new Czech summarization dataset created from CNA articles. Using this dataset, we trained multiple state-of-the-art approaches for extractive summarization using the BERT and Longformer model architectures and evaluate them using ROUGE-N, ROUGE-L and BertScore. We found that a pretrained Czech Longformer is the best approach regarding BertScore (0.802), when the number of summary sentences is known. If it is unknown, we found that the best approach is sentence-wise classification with context and positional metadata using a pretrained Czech BERT (BertScore 0.79).

Abstrakt

Automatická sumarizace textu je důležitý úkol z oboru zpracování přirozeného jazyka s mnoha aplikacemi. V této práci se zaměřujeme na sumarizaci novinových článků. V práci představujeme nový sumarizační dataset vytvořený z článků ČTK. Na tomto datasetu jsme natrénovali některé z nejmodernějších modelů pro extraktivní sumarizaci s využitím neuronových sítí BERT a Longformer a zhodnotili je podle metrik ROUGE-N, ROUGE-L a BertScore. Z experimentů vyplývá, že nejlepší model dle BertScore je založený na předtrénovaném Longformeru (0.802), ale lze jej využít jen pokud je dopředu znám či zadán počet vět ve shrnutí. Pokud tato informace k dispozici není, nejlepším přístupem se jeví klasifikace jednotlivých vět s kontextem a pozičními metadaty pomocí předtrénovaného modelu BERT (0.79).

Contents

1 Introduction

Summarization is the process of condensing information from a document into a shorter, more concise form. Summaries are useful for reducing the size of long documents, leaving their most essential information, and can be helpful for both authors and consumers.

Natural language processing (NLP) methods are generally applicable to all language domains, such as all-media communication, genetics [YM02], *byte codes* [Pop17], etc. However, in the context of this thesis, we will be focused on human language in textual form, and specifically, on that of the Czech News Agency (ČTK, CNA).

Automatic summarization of **text** specifically finds various applications in (but not limited to) social media [CA13, ACM$^+$18], search engine optimization [Mat18, RF00, SCJ$^+$20], research and medicine [FRK04, LRFP13], marketing [SSTW01, APPH16], question answering [DHD20, EBE19] and programming [HAM10], as opposed to summaries of music, genetic code, byte code or other natural and artificial languages.

There are multiple advantages of automatic over manual summarization: the former can be completed faster than the latter, especially for longer texts. It can be more accurate as well, since it relies on algorithms rather than human interpretation. Finally, automatic summaries can be easily customized to the needs of the user, whereas manual summaries require more time and effort to tailor.

In this thesis, we will explore the potential of state-of-the-art (SoTA) neural network models in automatic text summarization by evaluating their performance on a brand-new summarization dataset created from CNA articles. The best performing model will then be used to implement a system capable of summarizing multiple documents into a single summary.

2 Analysis

The first chapter of this thesis will focus on the study of existing summarization methods. Before we start, let us define the basic terms which we will use throughout this thesis.

2.1 Definitions

Documents are serialized entities (of any length) comprised of atoms originating from a single language domain. For example, a *text document* consists of individual *textual characters*, each one having a specific, unique position. One of many purposes of documents is conveying information between authors and consumers.

An *n-gram* is a substring (connected subsequence) of a document of length n (consisting of n atoms).

A *summary* is a document related to another document (an original) and exists in the same domain. The author of the summary tries to shorten its length while including the maximal amount of (general or specific) information from the original. Hence, the sole purpose of a summary is to carry information; the artistic, educational, or other value may be lost. From the syntactical point of view, summaries might have nothing in common with the originals, as they focus purely on semantics. Summarization, then, is the process of creating a summary from an original.

Since summaries rely only on the principles of compression and information preservation, it is easy to imagine summarizations of multiple semantically similar documents at once. The area of research developing and studying such techniques is called *multi-document summarization*.

2.2 Metrics

In order to assess the quality of automatic summaries without the need for human evaluators, a variety of metrics were designed. Discussing these metrics first, and only then describing the individual summarization methods, will help us understand the potential difference between training a summarization model and optimizing its metric value.

Every metric mentioned in this section is a *function* with two inputs:

1. *ground-truth reference summary* (usually created by humans, consequently denoted R) and
2. *evaluated summary* (consequently denoted E),

which returns a real number signifying how *good* the evaluated summary is.

2.2.1 ROUGE Family

The acronym ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation. ROUGE was invented in 2004 by Chin-Yew Lin in order to solve the problem of the necessary involvement of humans in validations of automatically generated summaries, which previously required 3000 man-hours of effort for the Document Understanding Conference [Lin04]. It is word-level oriented, i.e. it thinks of documents as sequences of words.

ROUGE-N The simplest method for calculating the ROUGE score is ROUGE-N, which creates unordered sets of all N-grams R_g, E_g from R and E respectively before calculating the *recall* of E_g on R_g (hence recall-oriented). In other words, it calculates the amount of N-grams in R_g which were present among E_g divided by the amount of N-grams of R_g in total, as shown in Equation 2.1. N stands for N-gram co-occurrence statistics.

$$\text{ROUGE-N}_r(R, E) = \frac{|E_g \cap R_g|}{|R_g|} \tag{2.1}$$

For example, if we had:

- $R =$ cats have fur
- $E =$ cats have ears
- $N = 1$,

we obtain $R_g = \{cats, fur, have\}$ and $E_g = \{cats, ears, have\}$ with $R_g \cap E_g$ being $\{cats, have\}$, therefore arriving at the final recall value of 2/3. However, consider the case if E were to be *"cats have ears, whiskers, paws and fur"* instead. The updated E is much longer than the ground-truth summary, which suggests that E is of poor quality (summaries should be as short as possible). Nonetheless, the ROUGE-N value becomes 1 (the maximum). Generally speaking, recall approaches 1 as $|R_g|$ approaches zero and $|E_g|$ approaches infinity. This is a classic problem appearing when recall is used as a metric of performance.

In the original ROUGE paper [Lin04], Lin assumed R to be a *set of documents* instead, therefore expecting (after concatenation of source documents) that $|R_g| >> |E_g|$. Such expectation also motivates the choice to avoid using *precision* as the metric core. Precision is calculated similarly

$$\text{ROUGE-N}_p(R, E) = \frac{|E_g \cap R_g|}{|E_g|} \tag{2.2}$$

and – inversely – approaches 1 as $|E_g|$ approaches zero and $|R_g|$ approaches infinity.

Improvements Fortunately, at least two possible ways to combat this problem exist. In *machine translation*, the BLEU [PRWZ02] metric is commonly used for evaluating automatic translations [PRWZ02, AL08, CBOK06]. Translation is a sequence-to-sequence task, just like summarization (mapping documents to documents). It is therefore unsurprising to find that metrics applicable to translation can be applied to summarization as well, be an inspiration for summarization metrics, and vice versa. BLEU is based on precision and its standard implementations include something called a *brevity penalty* [PRWZ02], which penalizes generated Es which are *shorter* than the originals, R. An extension of ROUGE-N can include a similar penalty called *redundancy penalty*, which functions in the opposite manner, as we intend to penalize Es *longer* than the corresponding Rs.

Another solution would be to simply calculate a *mean* of Equations 2.1 and 2.2, which automatically handles both cases at once. Two such metrics exist: the *geometric mean* of precision and recall,

$$GM(p, r) = \sqrt{|p * r|} \tag{2.3}$$

called G-measure, and the *harmonic mean* of precision and recall,

$$HM(p, r) = \frac{1}{\frac{1}{p} + \frac{1}{r}} \tag{2.4}$$

called F-measure. Both are suitable for this purpose, because unlike the *arithmetic mean*, they represent a *soft minimum*: the resulting metric value will be reduced more severely if either precision or recall are smaller than the other, which is intended. This forces the evaluated generator to produce outputs which have both high precision and recall. The harmonic mean is stricter in this sense.

In scientific publications, it is common to specify the parameter of N directly in the name: ROUGE-1 stands for ROUGE-N with N=1, ROUGE-2 stands for ROUGE-N with N=2, etc.

ROUGE-L Another way to calculate ROUGE is to attend to the longest common subsequence in both R and E. Thinking of $D \in \{R, E\}$ as a sequence of atoms, we call s a subsequence of D if there exists a strictly increasing sequence of indices $[i_0, i_1, ..., i_n]$ such that $\forall k \in \mathbb{N}, k \leq n : s_k = D_{i_k}$. The longest common subsequence of R and E, then, would be a subsequence which is both in R and E and has maximum length [Lin04]. Again, ROUGE-L is equal to 1 if R is equal to E and 0 if they have no common subsequence.

The difference between a substring and a subsequence is that, for a substring, the sequence of indices i_k must be specifically an interval of natural numbers (no skipping allowed). The longest common subsequence approach therefore does not punish exclusion of words of sentences from the summary as severely.

The common subsequence length does not suffice as a standalone metric however, since it scales with the length of R and E. Hence, it requires normalization. When normalizing with respect to the reference summary R, the LCS length is divided by the length of R, and vice versa with E. Intuitively, the former approach is recall-oriented (equals 1 if the whole reference is covered) and the latter approach is precision-oriented (equals 1 if the whole evaluated summary is correct).

This metric suffers from the same problems regarding the use of purely recall-or-precision-based metrics as the previous one; Lin recommends using the F-measure improvement with this metric as well. In the Document Understanding Conference 2004, which motivated the invention of ROUGE, however, the evaluators have applied a bias $\beta = 8$ to the recall part of F-measure F_l, as seen in Equation 2.5, which linearly weighs the importance of recall by a factor of β over precision.

$$R_l = \frac{|LCS(R, E)|}{|R|}, P_l = \frac{|LCS(R, E)|}{|E|}, F_l = \frac{(1 + \beta^2) R_l P_l}{R_l + \beta^2 P_l} \tag{2.5}$$

There are two advantages of using ROUGE-L over ROUGE-N. Word-level n-gram-based metrics are sensitive to consecutive word matches, which can punish good abstract summaries that can, for example, insert custom words in what would be an extracted summary sentence. LCS, on the other hand, pays attention to sequence matches, which just reflect the word order on the sentence level, skipping non-matching words. Another advantage is that the LCS is non-parametric, so there is no need to empirically set the N-gram size, perform grid evaluations, or anything of the like [Lin04].

Notice that ROUGE-L and ROUGE-1 both calculate unigram precision and recall and differ only in the fact that ROUGE-1 does so regardless of word order. This inherent feature of ROUGE-N unjustifiably forgives incorrect sentence structure. Consider the following example with ROUGE-2:

- $R =$ the cat chased a mouse
- $E_1 =$ a mouse chases the cat
- $E_2 =$ the cat chases a mouse

Here, the ROUGE-2 score for (R, E_1) and (R, E_2) are the same, since all three summaries have exactly two common 2-grams: 'the cat' and 'a mouse'. Notice, however, that E_1 and E_2 have completely opposite meanings. This factor has been exaggerated in this example for the purpose of explanation, however, in practice, this can pose a significant problem. The ROUGE-L score, on the other hand, equals 0.4 for (R, E_1) and 0.8 for (R, E_2), signalizing that E_2 is of much higher quality.

One significant disadvantage of ROUGE-L is the fact that only the longest common subsequence matters, which disregards other common subsequences [Lin04].

ROUGE-W Another disadvantage of ROUGE-L is its disregard for the spacing between atoms in the sequence. A human evaluator would claim that, when choosing between two summaries E_1 and E_2, both having the same LCS with R as the other, the one that has more consecutive atoms (i.e. is more like a common substring) is better [Lin04]. Therefore, Lin recommends the use of ROUGE-W (weighted LCS) instead. Using dynamic programming, ROUGE-W stores the lengths of consecutive matches during the computation of LCS, which are weighted and summed together. The sum of the weighted lengths, then, substitutes the LCS length.

The weighting is done by a separate function f which maps the number of consecutive matches to a real number. Meaningful variants of f are polynomials $f(k) = k^\alpha, \alpha > 1$, which assign more value to longer substrings.

Finally, the sum of weighted lengths is normalized as shown in Equation 2.6.

$$R_l = f^{-1}\left(\frac{WLCS(R, E)}{f(|R|)}\right) \tag{2.6}$$

P_l is adjusted likewise.

2.2.2 BertScore

Although the ROUGE family metrics are very popular as summarization evaluators, these metrics rely mostly on syntactical properties of the compared summaries and are slowly becoming obsolete. BertScore [ZKW$^+$19] is one of the most recent methods of summarization evaluation which utilizes the contextual embedding capabilities of large neural network models such as BERT (Section 2.5).

In order to calculate BertScore, both E and R are input to BERT, which returns a vector of size d_e for every single *token* of both E and R (see Section 2.5.1 for more details). These vectors are normalized to unit vectors and dot-multiplied between E and R, creating a matrix of size $(|E|, |R|)$. This matrix represents the similarity of individual tokens of E and R. To calculate BertScore precision and recall, one has to find the average maximum value of this matrix across the second and first dimension respectively. BertScore, then, is the F1 measure (harmonic mean) between these two values, calculated using Equation 2.4. It has been shown that this value correlates much more strongly with human judgment as opposed to ROUGE [ZKW$^+$19].

2.3 Single vs. Multi-Document Summary

Before we begin with examining possible approaches to multi-document summarization, let us enumerate the most significant differences between single-document summarization (SDS) and multi-document summarization (MDS) [GMCK00].

Redundancy Every input document will contain the relevant information to be highlighted by the summary, however, these documents will share a significant fraction of their content regardless of its relevance. For example, multiple news articles about a specific research conference might include historical information about its previous instances. Generally speaking, all input documents will contain irrelevant text, such as background description, introduction, etc., which will be *shared* by the individual documents. Therefore, a greater focus on redundancy elimination should be exercised.

Temporal Difference Topically similar documents may be created with an unspecified temporal gap between each other, which is typical for a stream of news reports. Specific parts of *newer* documents might override information carried by their *older* counterparts, for example, when reporters piece more data together and update their stories. This implies that a certain priority should be given to newer documents in MDS.

Compression Ratio In MDS, intuitively, the summary size should not depend on the amount of input documents. However, their number can vary and is usually larger than one (SDS), meaning that the compression ratio is much larger, roughly proportionally to the amount of input documents. According to Goldstein et al. in [GMCK00], in contrast to the TIPSTER Text Summarization Evaluation (SUMMAC) in 1998 [MHK$^+$99], where the summary compression ratios were 10%, Goldstein et al. summarized clusters of 200 documents each, which caused their compression ratios to drop as low as 0.1%, which puts more pressure on summary quality.

2.4 Automatic Summarization Techniques

In this section, we will discuss the possible ways of summarizing multiple documents, where the *input* is an ordered set of documents, and the output is a single document - a summary.

From the viewpoint of automatic summary generation methods, in practice, two standard approaches exist: *extractive* and *abstractive.* Methods belonging to the former group generate summaries by starting with the original documents, selecting which parts of the text to preserve. The "parts" are usually sentences, but counterexamples exist, such as when Jadhav, A. and Rajan, V. extracted summaries word-wise [JR18]. After selecting the important parts, the remnant is cropped and the result is deemed a summary. Hence, this approach is called "extractive", as it extracts the summary from the source text. The latter approach uses text-generation techniques to create an original summary from the source text.

2.4.1 Extractive Summarization

The most common approaches to extractive summarization are based on clustering sentences or representing the set of documents as a discrete graph. Both of these approaches will now be described.

Graph-based Graph-based methods function similarly as the PageRank algorithm [PBMW99] used in information retrieval. The common feature is, as was stated, trying to represent a document (or multiple documents) with a single discrete/network graph. Vertices of such graphs represent individual sentences of the original documents. The similarity of these sentences indicates whether there should be an edge between the corresponding vertices. The similarity can be calculated statistically using TF-IDF or by applying function approximator methods.

The subgraphs created by this process represent different topics covered by the source documents. More importantly, sentences being connected to many others signalizes their usefulness as a representative delegate. Such sentences should be more likely to be included in a summary [APA+17].

Allahyari et al consider TF-IDF weighting to be suboptimal for the use in this approach because of its reliance on statistics solely without taking the semantic similarity of sentences into account [APA+17]. This approach can be substituted by using a function approximator directly for calculating similarity, or by embedding the sentences in a semantic space.

Cluster-based Cluster-based methods function similarly to the graph based ones: sentences are split into clusters depending on their similarity. Specifying the distance between individual elements in clustering is key. There are two basic approaches this paper studies. One can either explicitly provide a full distance matrix, which is computed from sentence similarity, or one can embed the individual sentences into a vector space. The representations are then thought of as points in Euclidean space and their distance is deemed as Euclidean (Equation 2.7) or they can be projected onto the unit sphere centered at the origin and compared via their dot product (cosine distance, Equation 2.8).

$$dist_e(p,q) = \sqrt{\sum_{i=0}^{n-1} (p_i - q_i)^2} \tag{2.7}$$

$$dist_c(p,q) = p \cdot q \tag{2.8}$$

The main difference between graph based and cluster based methods is the fact that latter select a single representative sentence for each created cluster.

2.4.2 Abstractive Summarization

Abstractive methods can be further divided into *fully abstractive methods*, *graph-based methods*, *template-based methods* and *neural-net methods*, which will briefly be explained in this subsection.

Fully abstractive Fully abstractive methods are methods which extract information in the form of candidate phrases from the source text, select a subset of those phrases fit for inclusion in the summary, and finally combine them into syntactically correct sentences. The extracted noun and verb phrases can be selected according to automatically generated queries, ignoring content which is unlikely to be included in a summary [BLL$^+$15, GL12, MCN14]. The work of Wang and Cardie [WC13], which summarizes *dialogue acts* of spoken meetings, also shows that topic and domain knowledge can be utilized to guide the information extraction [LN19].

Graph-based Graph-based methods are methods which start by building a discrete graph over the information atoms, capturing (potentially complex) abstract relations between them [Gre11]. Every node corresponds to a specific action, trigger or state change (generalized as *event*) in the input text, while edges signify the semantic correspondence or relation between two events. After such a graph is built, it can be used to select the summary content by picking specific nodes to process. Those nodes can either be selected according to a length constraint [BLL$^+$15] or in order to keep the resulting subgraph connected [LN19].

Template-based Template-based methods are methods which seek to fill out trained summary template slots using the source document text. It has been observed that domain-specific summaries created by humans have recurring structures. If a dataset of such summaries existed, it is possible to learn and encode these templates directly from the dataset. Then, given an input document, the best-fitting template for the input document domain is selected and filled. An example of this approach can be found in [OMCN14].

2.4.3 Neural-net-based Abstractive Summarization

Neural-net based methods utilize general function approximation tools to map between input documents or extractive summaries and abstract summaries. For the purpose of mapping between ordered sets, the *sequence to sequence* (seq2seq) architecture is commonly used [Dug21, LN19]. Since these methods are currently state of the art, we allocate additional space for them in this thesis and pay increased attention to them.

Seq2seq networks can be split into two parts, an *encoder* and a *decoder*. The encoder learns to compress a sequence of any length into a constant-sized vector of features called the *context vector*, capturing the meaning of the sequence. If it is captured well enough, in theory, a decoder should be able to reconstruct the sequence from the context vector, possibly in another domain, language, etc., which is precisely what the decoder is trained to do. Encoders and decoders communicate only via context vectors; apart from the fact that the output of one is the input of another, they are independent of one another and do not have to share the same architecture. Their only common structural feature is an ability to process sequences of variable length; an input of length n is compressed to length $O(1)$, which is subsequently converted to an output of length m. This is commonly achieved using *recurrent neural networks* [SVL14], but *convolutional neural networks* can be used for this purpose as well. CNNs are more suitable for handling character-based representations of sentences as opposed to the classic word-based sentence representations. This is because the sentence length grows when represented by character atoms in contrast to word atoms, which further increases the risk of *vanishing gradient* [Hoc98] in already vulnerable RNN architectures.

Since the creation of the context vector is a process which has a theoretically unlimited compression ratio, encoding long documents without losing relevant information can be difficult. This is a serious problem particularly in the MDS case, as the input size is scaled by yet another factor: the number of documents. A possible solution lies in the extractive summarization methods, which can be used to filter out unnecessary sentences in order to shorten the input size [CB18, HLL$^+$18, LSL18]. Let us assume that a constant-length summary can be created from a variable amount of documents. If that is the case, it is safe to assume that such extractive methods can lead to the desired result, as the number of clusters (and therefore the amount of extracted sentences) will approach a constant limit.

2.5 BERT

The previous sections discuss the usage of neural networks frequently for purposes like contextual embedding, sentence representation, etc.. However, we have not yet introduced any specific architecture. Since most of our experiments utilize BERT directly (or its variants), we allocate this section for the description of this model.

BERT [DCLT18] stands for Bidirectional Encoder Representations from Transformers. Back in 2018, when it was initially released, it caused an uproar in the machine learning community, since it successfully overcame the state of the art in a vast range of NLP tasks. The main improvement of BERT over the Transformer [VSP$^+$17] is the implementation of bidirectional training using Masked Language Modeling [DCLT18], which combines processing the input sequence left-to-right and right-to-left [Hor18].

Language modeling [Ros00] is a NLP discipline which attempts to create computational models able to predict the next word in a partially built sentence or sequence. When performed using a neural network model however, its layered structure allows us to remove the last layer used for prediction and instead keep only those trained for representing the sentence as a feature vector in order for the prediction layer to work properly, leaving us with a powerful sentence embedding model usable as-is or for any other NLP task. This technique is called *transfer learning* and our work, among many others [PY09], makes great use of it.

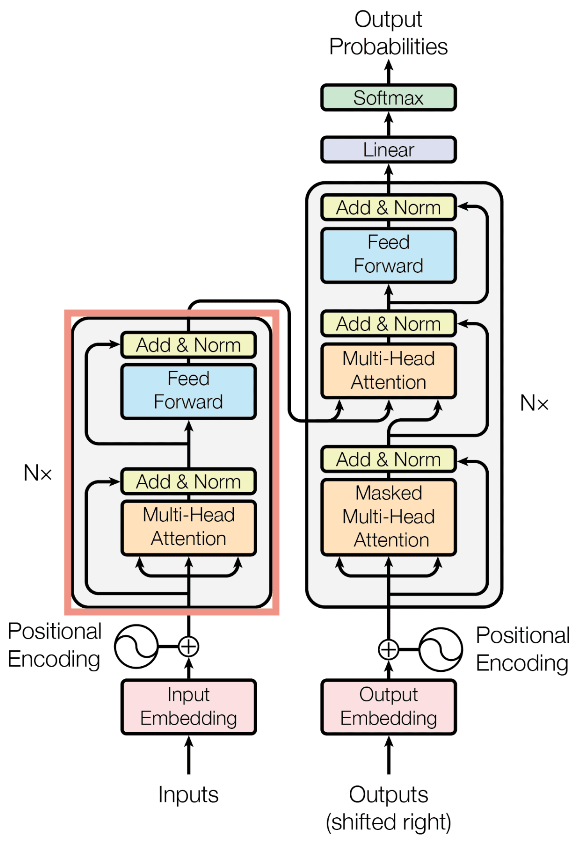


Figure 2.1: The architecture of a Transformer, highlighting the encoder part (the left-hand side), which evolved into BERT, a standalone model. The decoder (right-hand side) is used for generating text; it, too, has evolved into separate standalone models, none of which, however, were utilized in this thesis.

An overview of the entire Transformer architecture is visualized in Figure

2.1. In the remainder of this section, we will focus on describing the input embedding and the encoder.

2.5.1 Tokenizer

input	raw text
output	sequence of tokens

The original input to BERT, most other NLP neural network architectures, or systems which include those networks, is raw text. To help neural networks parse raw text, we first let it undergo a process called *tokenization*, splitting the input sequence into *tokens*. Tokens can be individual words, characters, or parts of words (called subwords).

In order to create tokens from text, an input corpus is converted into a stream of tokens and every unique token identified in it stored in a *vocabulary*, which is an ordered set of tokens. This allows us to assign a unique integer (an ID) to each token in the future. The representation is discrete on purpose – the models should not be able to infer anything about the syntactical or semantic similarity of two tokens based on their vocabulary index.

Using a word-level tokenizer helps assign meaningful representations to each token, however, this approach struggles with out-of-vocabulary (OOV) tokens: words that are included in the input text, but happened to be excluded from the original input corpus on which we built the vocabulary. The classic approach to solving this problem is adding a special token, [OOV], to the vocabulary, which does allow the system to assign a token to a previously unseen word with the caveat that such words will be represented by that single token, completely losing any semantic information. Apart from that, word-level tokenization results in a very large vocabulary.

On the other hand, using a character-level tokenizer makes sure that any input written in a target alphabet (or in any from a set of alphabets) is tokenizable and the resulting vocabulary is minuscule compared to the previous approach, but introduces two big disadvantages. Firstly, for most alphabets and languages (the Czech language for example), it is impossible to assign meaning to and therefore learn the meaning of any specific character, rendering the entire concept useless. Secondly, tokenizing by characters leaves us with a significantly longer token stream, which, in the best case scenario, scales the time and memory complexity of our model's forward pass by a linear factor.

The subword tokenization method is a compromise between these two approaches. Subwords are short character sequences - not quite words yet, but can be used as word building blocks. The first subword tokenizer called WordPiece [WSC$^+$16] was introduced by Wu et al. in 2016 for the purposes of machine translation and is used by BERT as well. The idea is simple - initialize the vocabulary with just the set of all characters and then keep merging units in the vocabulary in order to maximize the likelihood of their appearance in the input corpus until a given size of the vocabulary is reached. The resulting set of tokens contains the most common words whole, word stems, common prefixes and suffixes, etc.

This approach solves all of the aforementioned problems. First, the alphabet will most likely be included in the vocabulary as a whole, so there is no need for using the [OOV] token. Second, the vocabulary size is chosen according to the designer's needs. Third, the tokens can now be assigned a certain degree of meaning, as they usually represent entire words, and in the rarer occasions, three or more characters. And last, the token stream lengths are significantly reduced in comparison to character-level tokenization, since most tokens will be entire words. This, of course, depends on the input text rarity.

2.5.2 Input Embedding

input	sequence of tokens
output	input embedding matrix

Tokens as simple integers have no meaning by themselves. The idea behind tokenization in neural-network-based NLP is letting the network learn the embedding of all vocabulary tokens itself. The input embedding layer is parameterized by a large matrix of shape $(|V|, d_e)$, where $|V|$ is the size of the vocabulary and d_e is a chosen embedding dimension. The original paper uses $768 = 256 \cdot 3$. One-hot encoding the sequence of tokens and stacking them yields a matrix of shape $(N, |V|)$, where N is the input sequence size. Multiplying those two matrices together results in an embedding matrix of shape (N, d_e), which represents the entire input sequence for processing by the neural network. Gradient is propagated into this layer as well, which allows the network to learn meaningful token representations.

2.5.3 Positional Encoding

input	input embedding matrix
output	input embeddings with positional encoding

As we will show in the remainder of this section, BERT does not process the input sequence as a sequence *per se*, but instead as an unordered set of tokens. To add positional information to the network, the authors added another, very similar layer to the beginning of the BERT stack. However, the inputs to the positional encoding layer are not the tokens themselves, but instead their positions (as a natural number). Positional encodings can be trained; in that case, the layer is parameterized by a matrix of shape $(max(L), d_e)$, where $max(L)$ is the maximum allowed length of an input sequence; the positional representation is calculated similarly as the input embeddings. However, Vaswani et al. found that instead computing the positional encoding as shown in Equation 2.9 has negligible impact on model performance and does not limit the model input length.

$$PE(pos, 2i) = sin\left(\frac{pos}{10000^{2i/d_e}}\right), PE(pos, 2i+1) = cos\left(\frac{pos}{10000^{2i/d_e}}\right) \tag{2.9}$$

The positional encoding generates an N by d_e matrix, which is added to the input embedding matrix and passed to the main stack of BERT.

2.5.4 Multi-Head Self-Attention

input	embedding matrix (N, d_e)
output	attended embedding matrix (N, d_e)

The attention mechanism is one of the key features of Transformers, and transitively, of BERT. Intuitively, it is a parameterized querying mechanism that returns *values* corresponding to *keys* which are similar to input *queries* (see Figure 2.2). More formally, it maps a query and a set of key-value pairs to a specific output, with the query, key, and value being all vectors. The output is computed as a weighted sum of all values, where the weights corresponding to each value are derived from a similarity metric between the query vector and the key vector corresponding to this particular value. This mechanism lets the network assign the contextual importance to individual tokens with respect to another token, helping it build deep knowledge about the language itself.

Thanks to the nature of the algorithm, the dimensions of the value vectors, key vectors and query vectors do not necessarily have to equal, but as BERT uses dot-product attention (for the purposes of speed), the query dimension and key dimension are the same, d_q. The value dimension is equal to d_v.

Queries, keys, and values are computed from the input matrix using a projection matrix for each class, which have shapes (d_e, d_q), (d_e, d_q), and

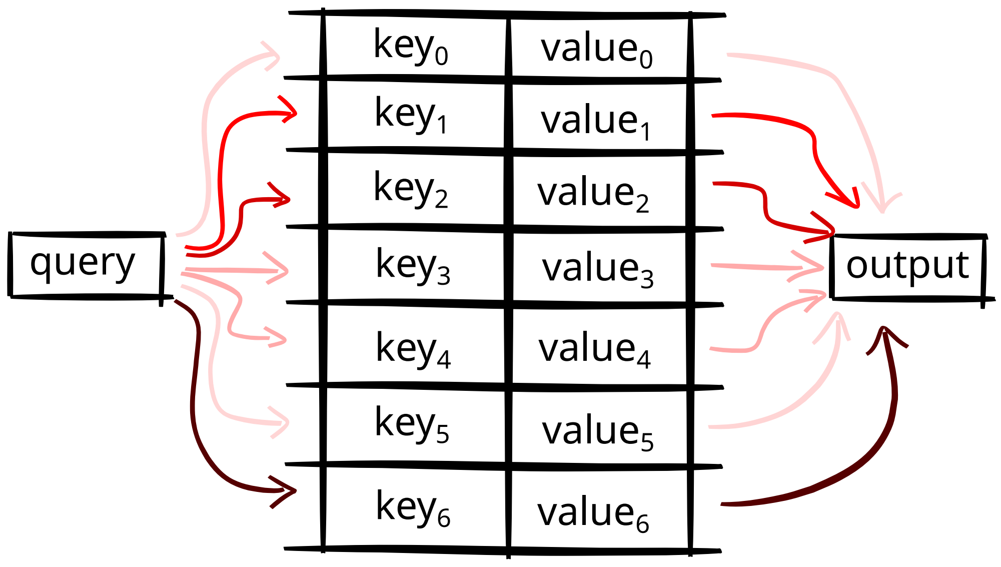


Figure 2.2: The query-key-value mechanism: *output* is created as a weighted sum of *values* according to the similarity of *query* to the individual *keys*.

(d_e, d_v) respectively. Every input vector is therefore transformed by matrix multiplication into a single (but separate) key, query, and value, compactified in matrices Q, K, and V of shapes (N, d_q), (N, d_q) and (N, d_v). Given the fact that the queries, keys and values come from the same source, the attention mechanism is called *self-attention*, since the inputs attend to themselves. Attention, then, is computed as seen in Equation 2.10,

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_q}}\right)V \tag{2.10}$$

where the softmax argument is scaled down by the factor of $\sqrt{d_q}$ in order to prevent the function from being calculated in regions where the gradient is too small [VSP+17].

However, instead of performing a single attention function, Vaswani et al. found it beneficial to project the inputs to smaller dimensions h times. h signifies the number of *heads* in the multi-head attention mechanism. Each head computes its own self-attention with the key, query and value dimensions h times smaller in order to preserve the total computational cost. The individual head results are concatenated along the last dimension, receiving the full weighted value matrix of shape (N, d_v). This matrix is then projected back to the original input dimension using another parameterized matrix multiplication, resulting in a matrix of shape (N, d_e), just like the original input.

2.5.5 Add and Norm Operations

input	embedding matrix (N, d_e)
output	normalized embedding matrix (N, d_e)

This layer consists of two normalization mechanisms which function as a stabilizer to the gradient flow. Neither transforms the shape of the input.

Operation Add stands for a residual connection sum [HZRS16], which is heavily used in deep neural networks to prevent gradient vanishing over long distances in the network. The problem is solved by introducing skip connections over sequences of layers (or individual layers).

Basically, if f is a layer in the network, it turns $y = f(x)$ into $y = f(x) + x$, which allows gradient to bypass the layer logic completely and makes modeling the identity mapping trivial, which, in theory, allows the model designer to add as many layers as desired without worry of vanishing gradient, since redundant layers can be trained to simulate identity.

Operation Norm stands for layer normalization, which simply keeps track of the first two moments of feature values token-wise and normalizes them (sets *mean* to 0 and *std* to 1) for processing by the next layer. Normalization helps regularization (generalization) and speeds up training [BKH16].

2.5.6 Feed Forward

input	embedding matrix (N, d_e)
output	linear projection of the embedding matrix (N, d_e)

The feed-forward layer consists of two classic, densely connected MLP layers. The authors use RELU as the activation function for the first pass and no activation for the second pass.

The self-attention and feed-forward together with normalization and residual connections form a BERT block that can be chained, as its input and output dimensions equal. This helps BERT fully utilize the representational power of deep learning.

2.5.7 Additional details

The tokenization process for BERT includes adding a classification token, [CLS], at the beginning of each token sequence. This token attends to other tokens as well and is used for classification purposes, for example during next sentence prediction. It also adds [SEP] tokens between sentences, for example, during question answering, where the question is separated from the answer this way. For this purpose, BERT also includes *token type ids*, which is a binary vector of the same length as the input length, which is used to distinguish between individual sequences further. This simply introduces a new, parallel input layer like the one which computes position encodings; all embeddings/encodings of a single token are always summed together before entering the BERT stack. Furthermore, if the input sequence size is longer than the sequence which is actually input, BERT uses a special *attention mask* vector, which is a binary mask signifying which tokens are used for padding, so that they are ignored during the computation of attention.

2.6 Longformer

Although the inferential power of the self-attention mechanism is huge, it also poses a problem: the computational and memory complexity of self-attention in BERT is quadratic in terms of sequence length. This motivated the design of Longformer [BPC20], which addresses this problem by adjusting the attention mechanism.

By default, every key vector was dot multiplied with every query vector. To reduce the complexity with respect to the sequence length to linear, the authors of Longformer replaced the full quadratic attention with a sliding window attention, letting tokens attend to a constant amount of only a handful of tokens in their close neighborhood, as seen in Figure 2.3.

2.7 Projection

As we have stated in Section 2.5, particularly in Section 2.5.6, the output of a BERT model is a vector v of some given size d_e for *every input token*, resulting in a matrix O consisting of vectors $v_0, v_1, ..., v_{n-1}$. For nearly all purposes in the context of this thesis, we need a way to convert this matrix to a constant-size vector representation for each sentence or even for the entire input sequence. An exception is the computation of BertScore, which uses vector representations of all tokens, see Section 2.2.2.

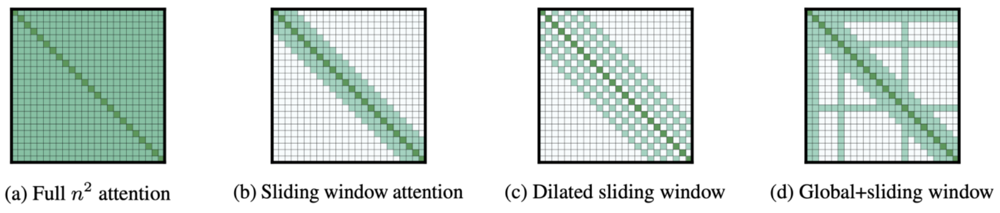

(a) Full n^2 attention
(b) Sliding window attention
(c) Dilated sliding window
(d) Global+sliding window

Figure 2.3: Longformer sliding window attention in comparison with BERT full n^2 attention. Each row represents one individual attention operation. The diagonal elements refer to tokens which are used as queries, i.e., which are going to be attending to other tokens. The remaining row elements are either dark (attended to) or light (ignored). Variant (a) visualizes the attention in BERT. In the Longformer, variants (b), (c) and (d) refer to ways in which the attention mechanism is adjusted. Variants (b) and (c) make the central token attend to its neighboring tokens consequently or with a given gap respectively. Variant (d) builds upon this idea by attending to important tokens, such as [CLS] and [SEP] globally, regardless of distance.

We have dubbed the function which handles this conversion in our thesis a *projection*. There are many ways to implement a projection and we have decided to analyze a few of the most basic approaches. Of course, simply reshaping the matrix, i.e. concatenating the individual vectors into one single vector does not work, as the size would not be constant (dependent on the input sequence size).

2.7.1 Special Tokens

This – according to our previous research on BERT – is the most common way of converting the output matrix O to a vector. The principle is simple: keep only the vector v corresponding to one of the special tokens, like [CLS] or [SEP], and discard the remainder. [CLS] has been created specifically for this purpose [DCLT18, Hor18] and is used for representing the entire sequence, [SEP] can be used in a similar manner to represent individual sentences.

2.7.2 Pooler

This approach can be extended by processing the [CLS] vector representation v_0 by a fully connected layer called the *pooler*. This layer has its own matrix of weights of shape (d_e, d_e), which performs a linear combination of v_0 with itself and transforms this output element-wise by a *tanh* function.

2.7.3 Aggregative Reduction

The final projection variant we have examined performs a reduction operation f over the input sequence dimension, like *averaging* or *maximum*. This results in a vector

$$\begin{bmatrix} f(v_{0,0}, & v_{0,1}, & ..., & v_{0,n-1}) \\ f(v_{1,0}, & v_{1,1}, & ..., & v_{1,n-1}) \\ ..., & ..., & ..., & ... \\ f(v_{d_e-1,0}, & v_{d_e-1,1}, & ..., & v_{d_e-1,n-1}) \end{bmatrix} = \begin{bmatrix} f_0, \\ f_1, \\ ... \\ f_{d_e-1} \end{bmatrix}$$

Such vector has the expected size d_e and can therefore be used for further processing.

2.8 Extractive BERT Summarization

Given the fact that BERT and its derivative models, such as the Longformer, are the SoTA for many NLP applications, it is not surprising to find BERT being applied to summarization. This section will briefly summarize the core idea behind these models, which will be thoroughly described and evaluated through experiments in the later sections.

2.8.1 BERT Sentence Classification

One of the simplest ways to perform extractive summarization using BERT has been suggested by Gu and Hu [GH19]. They simply used BERT as a classifier of individual sentences, deciding whether they belong to the summary or not. Their results show that, without the knowledge of the surrounding context, it is hard to figure out whether a sentence should be included in a summary.

2.8.2 BERT Sentence Classification with Context

An improved variant of this approach is to add the surrounding context of the classified sentence to the input. Since BERT has a limited amount of input tokens (sequence length), one can center out the classified sentence in this sequence and clip/pad the remainder. Adding context should theoretically help the classifier in identifying sentences which are significant for summary inclusion.

2.8.3 BertSum

A better approach has been suggested by Liu and Lapata [LL19]. The core idea is captured well by Figure 2.4, where the authors simply alter the

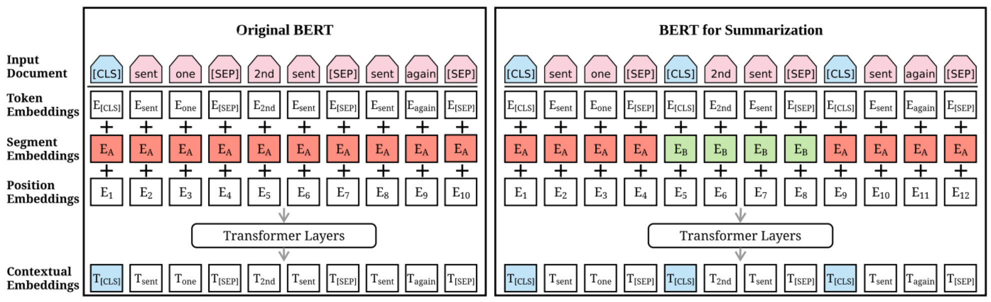


Figure 2.4: Difference between BERT and BertSum. Notice how the classification token [CLS] has been appended after each separator [SEP] token in the input. This allows the model to output classification embeddings for each sentence, which can be used for deciding whether a sentence belongs to the summary or not. Moreover, the segment embeddings used for NSP and QA are now used to distinguish between individual sentences.

tokenization procedure for the output to conform to its prerequisites, that is, being able to classify each sentence as being part of the extracted summary, or not, independently on one another.

For abstractive summarization, the authors used a 6-layered Transformer as a decoder [LL19].

2.8.4 BERT Clustering

Another approach to summarization using BERT has been proposed by Miller [Mil19], where a pretrained BERT is used as a sentence embedder. The embeddings are then passed along to a k-means algorithm, which is one of the simplest clustering methods. After the clustering algorithm converges, the sentences closest to the centroid are selected as summary sentences.

2.8.5 BERT RNN

These sentence embeddings can be used as an input to a recurrent neural network (RNN), such as LSTM or GRU, inspired by the Extractor architecture in [WLZ+19]. In order to encode context in both directions, the bidirectional extension of each is preferred over the unidirectional alternative. Bidirectional RNNs function by splitting the RNN layer into two branches/heads, each processing the input separately and in opposite directions. Finally, the outputs from both heads are concatenated.

RNNs can be chained the same way BERT blocks can, since their inputs and outputs have the same format. Finally, the RNN outputs are processed by a single-neuron layer (perceptron), whose outputs signalize whether the sentence corresponding to the embedding should be included in the summary.

2.9 Analysis Conclusion

Given the scope of this thesis and the expected quality of extractive summarization methods, we had opted for using one of the above methods for implementing the summarization system, as each of them utilizes BERT, the SoTA neural network model for natural language processing. Based on our analysis of these methods, we have decided that our units of separation will be individual sentences. Since we were unable to determine which one of these methods works best for our purposes, we have decided to conduct experiments, evaluating each one of the methods described above. The underlying models were trained on our unpublished summarization dataset – which will be presented in the following chapter – and evaluated using ROUGE-N, ROUGE-L and BertScore.

3 Realization

This chapter will focus on presentation of datasets used for experimentation, detailed description of our experiment models, implementation details and obtained results.

3.1 Datasets

Before we discuss model implementation details, let us first present the datasets we have used for implementing and evaluating these models.

Both datasets mentioned here have been based on **incidents** provided to us by the Czech News Agency (CNA). An incident – in this context – is any media-relevant *event* reported by CNA. Each *event* is described by multiple **articles** created by individual authors, which are sorted by their time of creation. An abstractive **summary** of these documents is included in the incident as well.

type	**train size**	**eval size**	**format**
summarization	1,431	159	sentences, labels, perm
STS	116,956	1200	S1, S2, sts score

Table 3.1: Overview of datasets used for training the models used in this thesis. The *semantic textual similarity* (STS) dataset was created by Sido et. al. [SSP$^+$21] from news article annotations, and the summarization dataset, which was created for the purpose of this thesis, was built by us from the same annotations. The *format* descriptions for *summarization* and *STS* are explained in Sections 3.1.2 and 3.1.1 respectively.

3.1.1 Annotation Procedure

During the *first round* of annotation [SSP$^+$21], given one specific incident, annotators were shown sentences from its summary and each article. Their task was – for each summary sentence – to pick *three* sentences from the corresponding articles: one as semantically similar to the summary sentence (S) as possible (type A), one as semantically dissimilar from the summary sentence as possible (type C), and one vaguely related to the summary sentence (type B). Furthermore, their task was to provide a *semantic textual similarity* (STS) score for all three pairs SA, SB and SC. These STS scores were used for building the published Czech News Dataset for Semantic Textual Similarity dataset [SSP+21].

In this thesis, we have utilized this dataset given the need to estimate the similarity of two sentences for the purpose of clustering. Individual elements are 3-tuples $(S_1, S_2, \textit{stsscore})$, where S_1, S_2 are two Czech sentences and *stsscore* is a real number in the range $[0-6]$ representing their semantic similarity, where 0 stands for "unrelated" and 6 stands for "equivalent". The dataset contains 116,956 such tuples for training and 1200 tuples for evaluation.

3.1.2 Building the Summarization Dataset

However, in the context of this thesis, we clearly require an extractive summarization dataset as well. The aforementioned data annotation procedure has been designed with this purpose in mind. Therefore, during the preparation phase of training and evaluating the summarization models introduced in the Analysis chapter, we have been able to build such a dataset from the annotated data.

Starting with the abstractive summaries, we simply claim that the corresponding extractive summary is built from type A article sentences. However, the mapping between abstractive summary sentences and article sentences is not 1:1, but M:1, because any article sentence could have been selected as type A for a given summary sentence. If any such collisions occurred, they were ignored, allowing possibly shorter extractive summaries than their abstractive counterparts. This decision was motivated by the fact that such collisions simply signify the great quality of a sentence being included in a summary; we felt no need for forcing the annotators to select distinct article sentences for multiple summary sentences.

We have compiled this dataset into a JSONL file. JSONL is a text file format which contains a separate stringified JSON object on each line. The advantage of using JSONL instead of a JSON array is the possibility of loading individual lines of the dataset without having to load the entire file into memory or utilizing advanced parsing.

Each line contains one dataset element. The element structure is as follows:

- "text": array[array[sentence]],
- "label": array[array[0/1]],
- "perm": array[array[integer]]

One array of sentences makes up an article. Given the fact that this is a multi-document summarization dataset, we include every article, which therefore makes an array of arrays of sentences. Since the dataset is extractive as well, each sentence in "text" is assigned a binary label signifying whether the sentence was type A or not. Both of these array structures have equal shape; neither of them are padded, they are ragged instead.

Perm stands for permutation. An argument supporting the idea that extracted summaries should be reordered can be made, so the information about correct permutation is included in our dataset as well. Every integer in the "perm" substructure corresponds to a specific sentence; it is 0 iff the corresponding label is 0, otherwise it is the positional index of an abstractive summary sentence corresponding to the extracted type A summary sentence, starting with 1.

During training and evaluation, the full JSONL file, which had 1590 lines (elements), was split into train and test files using the commonly used 90%/10% split, producing files `train.jsonl` with 1431 lines and `test.jsonl` with 159 lines. The elements were randomly shuffled before the split occurred so as to not introduce any bias to our training and evaluation data: for example, the original full JSONL file contained elements sorted by abstractive summary length (caused by the script which generated it). Without shuffling, `test.jsonl` would contain summaries of previously unseen length to a model trained on `train.jsonl`.

3.2 BERT-based models

Before we start describing the individual model architectures, let us first address the common features shared by each of these models.

3.2.1 Pretraining and Tokenization

The BERT and Longformer models used as a basis for each model evaluated in this thesis were pretrained by Sido et. al. [SPP+21]. The tokenizer used for tokenizing the input text is a WordPiece subword tokenizer with vocabulary size = 30522. The models were pretrained on roughly 340,000 Czech sentences (approx. 50x more than the SoTA multilingual BERT models) and proved to outperform the competition in many NLP disciplines concerning the Czech language [SSP+21].

With accordance to the origin article, we will call these pretrained models (BERT and Longformer with specific parameter settings) CZERT and CZERT-long respectively in the remainder of this thesis.

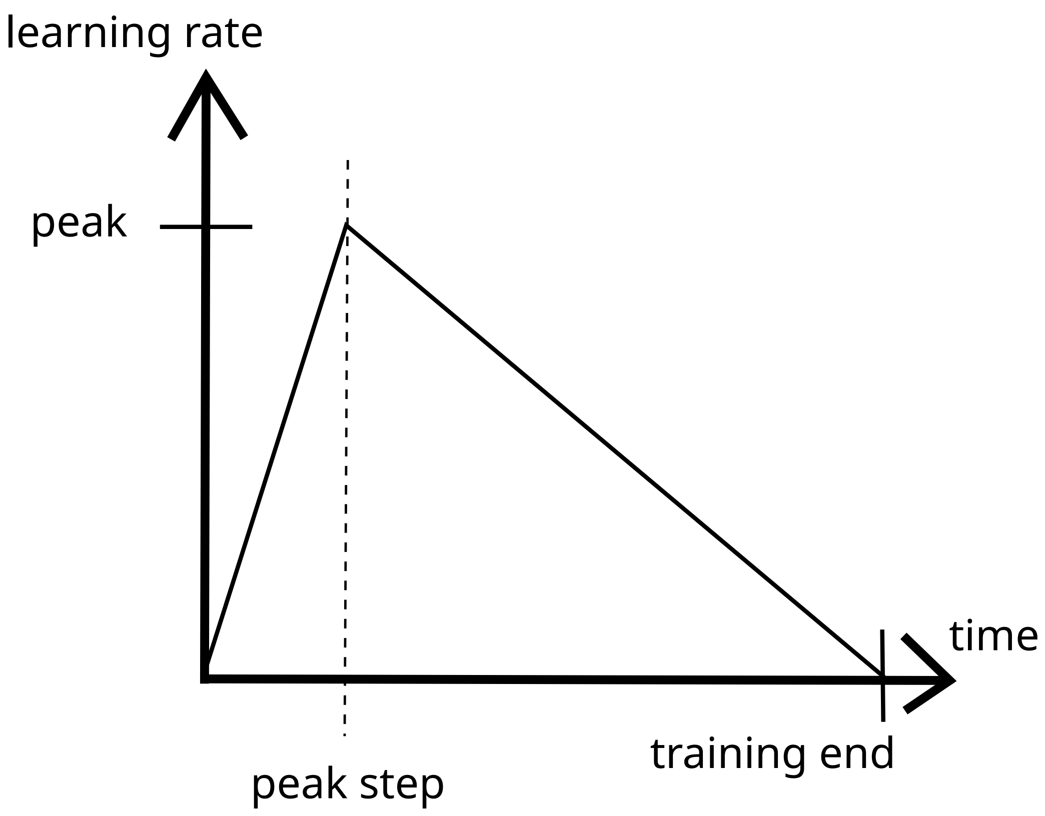


Figure 3.1: Linear learning late scheduler curve.

3.2.2 Inputs

Models that tokenize the entire set of input articles concatenate them into a single bulk of text without breaking their order. This – in theory – allows the models to attribute the significance of individual sentences to their positional IDs which now include the temporal information carried by each article.

3.2.3 Training

All models have been trained over 100 episodes using a linear learning rate scheduler (see Figure 3.1) with peak 10^{-5} at step 100 for STS and 500 for summarization. Our preliminary hyperparameter grid search showed that adjusting these values has negligible impact on model performance.

Unless specified otherwise, every model was trained *end-to-end*, that is, no layers were frozen during training.

3.2.4 Evaluation

No model in this thesis (with the exception of clustering, see Section 3.3.6 for details) provides the binary summarization classes directly. Instead, it provides real numbers (called *logits*) from which these classes are computed through comparison with a specific threshold. We have decided to experiment with the following two options for extracting a summary:

- a) using the aforementioned threshold for converting the logits to binary classes: $logit > threshold \to 1$, otherwise 0
- b) returning 1 for the *largest k* logits and 0 for the rest, where k is the expected summary length

Option b) assumes that the model receives the expected number of summary sentences from an *oracle*. We intended this to be the default mode of execution: the user inputs articles and selects the desired summary length, not enforcing a specific output length on the user. This makes the parameterless option a) worse, as no trivial solution to limiting the summary length after thresholding the logits exists.

In the end, all summarization models are evaluated using the metrics we have arrived at in Section 2.9. However, during training, the models' performance is monitored using F1 measure, as seen in Equation 3.1. This decision is motivated by the fact that extractive summarization is *de facto* a classification task and computing F1 between two binary vectors is much faster than converting these vectors to textual summaries and computing any of the aforementioned evaluation metrics. Furthermore, we wanted to avoid biasing the models towards optimizing any single one of these metrics, because our final measurements should serve as an unbiased description of the summarization dataset we created.

$$F1 = \frac{TP}{TP + 0.5(FP + FN)} \tag{3.1}$$

That being said, we did not make use of *early stopping*, which is a technique used to prevent *overfitting*. Instead, we kept track of the F1 measure on our *evaluation dataset* and whenever the model arrived at a new maximum, we saved its weights. After the model was done training, we computed ROUGE-N, ROUGE-L and BertScore on the saved model weights.

3.2.5 Activation Function and Dropout

After the BERT output is *projected*, it is almost always processed by more layers – either perceptrons or RNNs. Every such layer makes use of dropout with a probability of 0.5 and the Gaussian-error linear unit (GELU) activation function (Equation 3.2, Φ = normal distribution CDF, σ = sigmoid function) derived from the normal distribution. Dropout 0.5 introduces the maximum amount of neuron combinations ($\arg\max_n \binom{m}{n} = 0.5 \cdot m$). GELU has been shown to outperform the standard RELU as an activation function in many different disciplines [HG16].

$$GELU(x) = x \cdot \Phi(x) \approx x \cdot \sigma(1.702x) \tag{3.2}$$

3.3 STS model

We found the unsupervised approach described in Section 2.8.4 to fall short of the potential of a fine-tuned model. Therefore, we have decided to experiment with creating a model able to output distance matrices between article sentences according to their semantic textual similarity.

We assume two possible approaches for estimating the STS between pairs of sentences using CZERT. Both approaches make use of the same dataset and train/test split as described in Section 3.1.1.

3.3.1 Two-sentence Input into Regression

One way of having CZERT estimate the STS between two sentences is to input both of them separated by a [SEP] token, resulting in the following sequence: `[CLS] sentence_1 [SEP] sentence_2 [SEP]` and training the BERT model as a regressor. This approach has been shown to correlate slightly better with the test data [SSP+21].

3.3.2 Siamese One-sentence Input, Cosine Similarity

Another option is to input the sentences separately, let CZERT create their embedding (no regressive or any other head), and extract the resulting STS by computing cosine similarity (Equation 2.8) between their normalized embeddings. This architecture is the exact same as that of SentenceBERT [RG19].

3.3.3 Comparison

Even though the former model performs slightly better [SSP+21], it suffers from a massive disadvantage with respect to the system it would be imple-

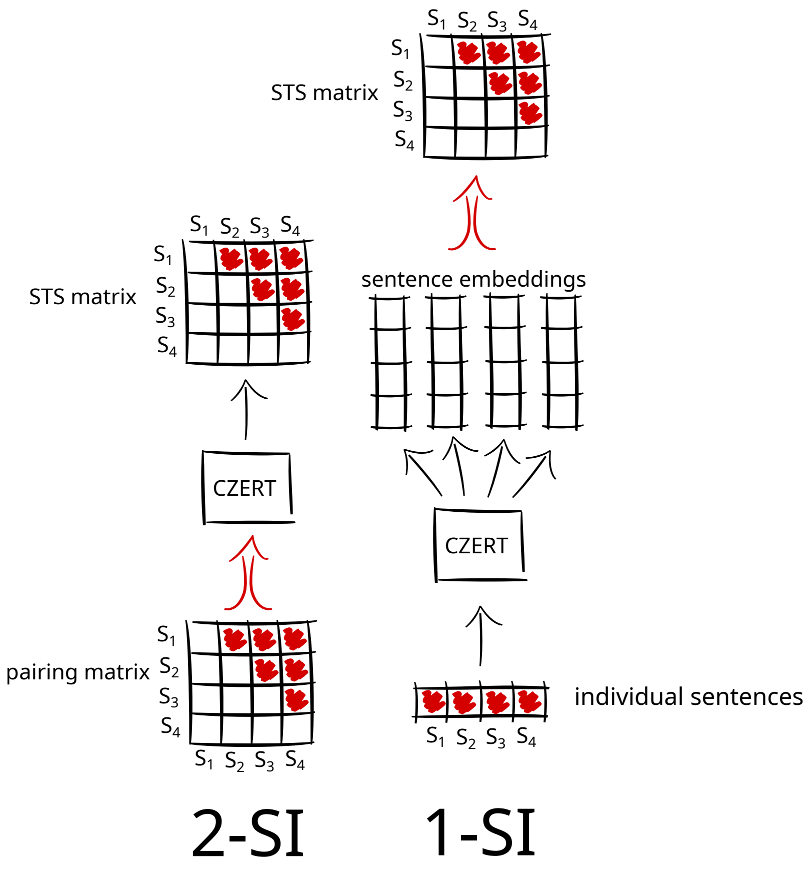


Figure 3.2: Overview of the difference between the two-sentence input (2-SI) approach and one-sentence input (1-SI) approach. The thicker arrow signalizes the operation which runs in quadratic time. *Pairing matrix* represents an imaginary matrix of sentence pairs for visualization purposes; the pairs are input separately. Although the creation of the STS matrix takes more steps for 1-SI, clearly, the 2-SI approach uses more computation time, as CZERT is the main performance bottleneck.

mented in. The essential difference between these two models is the stage during which the actual STS is calculated.

Using the two-sentence input model requires inputting every single pair of sentences and therefore performing $\frac{n\cdot(n-1)}{2} \in \mathcal{O}(n^2)$ CZERT forward passes [RG19]; see Figure 3.2. The distance matrix is then built trivially. However, using the one-sentence input Siamese model lets us embed each sentence separately ($n \in \mathcal{O}(n)$ CZERT forward passes) and then perform $\frac{n\cdot(n-1)}{2}$ dot product calculations (see Figure 3.2), which is significantly faster. This has been the main motivation behind deciding to use the Siamese model for any further STS estimations throughout this thesis.

3.3.4 Architecture

Input The input sequence length for the CZERT model used here was originally 128, which is the smallest power of two larger than the size of almost every single tokenized sentence in our summarization dataset (see Figure A.6a). However, reducing it to 64 still includes a vast majority of sentence content and has negligible impact on STS quality while speeding up the forward pass speed by a factor of ~ 4. Generally speaking, comparing the semantics of two sentences is much easier than selecting sentences for extractive summarization, which supports our decision to halve the available input size.

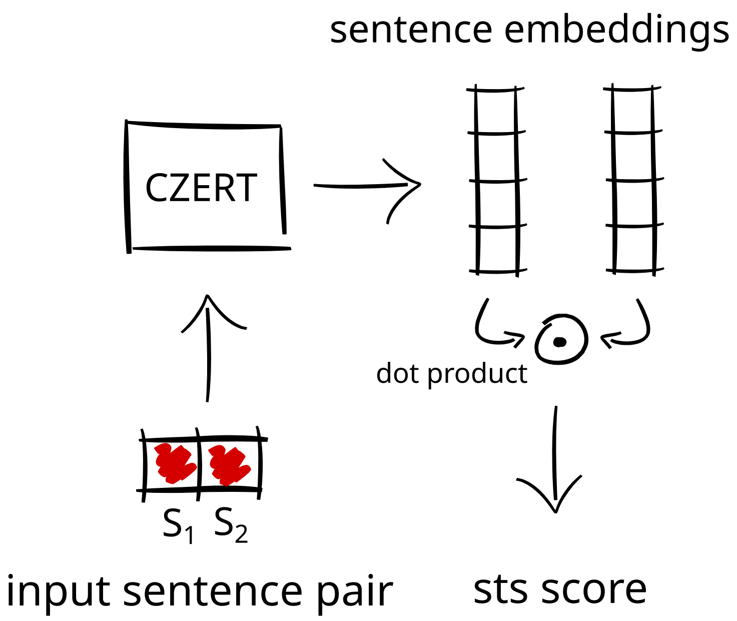


Figure 3.3: STS model architecture overview. The sentences s_1 and s_2 are input separately, as per the one-sentence approach, which results in two separate, independent embedding vectors. The *dot product* operation is preceded by vector normalization.

Forward pass First, the two input sentences are separately tokenized, treated as a full sequence. Both are then input to the CZERT model and *projected* (using a projection from Section 2.7). We then normalize both resulting vectors and compute their dot product (see Figure 3.3).

Cosine similarity to STS However, since the STS scale from [SSP+21] is in the range $(0, 6)$ and a dot product of two normalized vectors is in the range $(-1, 1)$, the variety of ways in which to execute this mapping opened up more room for experimentation.

We have opted for choosing from three separate mappings, where p stands for the dot product of the two embedding vectors:

- **a)** $sts(p) = 6p$
- **b)** $sts(p) = 3(p + 1)$
- **c)** $sts(p) = 6|p|$

Depending on the exact implementation of SentenceBERT, either variant a) or variant b) is equivalent to their method of inference [RG19]. This depends on whether the authors re-scaled their STS datasets from range $(0, 1)$ to $(-1, 1)$ (resulting in variant b) or not (resulting in variant a), however, such details have been omitted from the paper.

We find variant b) suspicious, since it forces the model to embed unrelated ($sts = 0, p = -1$) sentences on the same line (see Figure A.9), even though there are theoretically infinitely many more unrelated sentences with respect to a given sentence.

Variant a) improves upon this, as the larger the angle between two sentence embeddings is, the smaller their STS score. However, if an angle between two sentences becomes greater than $\frac{\pi}{2}$, the STS score leaves the expected range $(0, 6)$. We expect this would mean that the model would try to force the entire embedding space to fit a single hyperoctant (a multidimensional analogy of a quadrant in 2D), shrinking the possible embedding space. Moreover, during inference, the STS score would have to be clipped.

These observations led to the proposal of variant c), which extends variant a) through symmetry. Using it states that all sentences, whose embeddings approximately lie on a single line, are equivalent, and sentences which are orthogonal to this line are unrelated, with varying degree of similarity in between.

STS distance matrix Of course, regarding summarization, to finally convert the STS similarity matrix to an STS distance matrix, one has to subtract the similarity matrix from a constant matrix of sixes: $STS_{dist}[i, j] = 6 - STS_{sim}[i, j]$.

3.3.5 Training

We have trained the model using the aforementioned STS dataset using the mean-squared error as our loss function (regression, Equation 3.3). The optimality of the STS model was measured by a correlation metric with respect to the evaluation dataset with 1200 elements.

$$J(y, \hat{y}) = \sum_{i=0}^{n-1} (y_i - \hat{y}_i)^2 \tag{3.3}$$

3.3.6 Summary inference

After executing our experiments, we have shown that the optimal configuration of this model is the usage of mean aggregation projection and variant c) for conversion between dot products and STS.

In order to use this STS model for summarization, we need to execute the following steps. First, the entire input article set is split into a set of individual sentences. These sentences are then embedded by the STS model, allowing us to compute a distance matrix (the output in Figure 3.2, 1-SI). This matrix is then used as an input to a *hierarchical agglomerative clustering algorithm* [Mül11], which separates the individual sentences into equivalence groups called *clusters*. Finally, a single *centroid* is selected from each of these clusters, identifying an extracted summary sentence.

Clustering In the context of agglomerative hiearchical clustering, there are two possible approaches: clustering by number of target clusters k, which forces the algorithm to identify precisely k clusters, or by element cutoff distance d, which claims that two elements lie in the same cluster iff the distance between them is less than d.

During inference, instead of evaluating *logit thresholding* (Section 3.2.4), which bears no meaning in clustering context, we simply perform clustering with **20 target clusters**. Similarly to the *top-k* approach (Section 3.2.4), we perform clustering with **k target clusters**.

Furthermore, in order to discover the performance ceiling of STS clustering, we experiment with distance-based clustering using an oracle for distance. The purpose of these oracles is to find such cutoff distance which maximizes summary quality.

Both oracle variants we have experimented with make use of a simple linear search over the cutoff distance space. In our setting, this space amounts to the set of all numbers between 0 and 6. With our chosen granularity 25, the values to evaluate form a discrete set $\{0, 0.25, 0.5, ..., 5.5, 5.75, 6\}$. For each incident, we have tried building a clustering with a cutoff distance d for each number from this set. One oracle variant, called **local distance oracle**, always simply returned the clustering which maximized the F1 measure. The other oracle variant, called **global distance oracle**, first computed which distance d maximized the *average* F1 and then performed distance clustering with this d for each incident. Figures A.7a, A.7b and A.7c visualize the average F1 values for each evaluated d, showing that the optimal value of d is 2.5.

Centroid selection Given a distance matrix, performing agglomerative clustering is simple. However, choosing an appropriate representative – a *centroid* – for every given cluster is not trivial. For this purpose, we define two measures of centroid quality.

If C is a set of sentence embeddings inside the currently processed cluster and S is the set of all sentences, we define measures *centrality* c and *uniqueness* u as shown in Equations 3.4 and 3.5:

$$c(p) = -\frac{\sum_{q \in C} dist(p, q)}{|C|} \tag{3.4}$$

$$u(p) = \frac{\sum_{q \in S \setminus C} dist(p, q)}{|S \setminus C|} \tag{3.5}$$

Both centrality and uniqueness are desirable qualities of a cluster centroid. High centrality signalizes the fact that the centroid candidate represents the cluster well, as it is semantically close to the remaining cluster members. High uniqueness, on the other hand, means that a cluster member is sufficiently semantically different from members of the other clusters.

Therefore, given a specific clustering, the three options for centroid extraction we have experimented with are:

- $\arg\max_p c(p) \sim$ (selecting according to) **centrality**
- $\arg\max_p u(p) \sim$ **uniqueness**
- $\arg\max_p c(p) + u(p) \sim$ **both centrality and uniqueness**

Further notes The unweighted, simple sum of c and u is justified, because both are sums of our STS distance matrix row subsets divided by the subset size. Since STS scores are in the range $(0, 6)$ and therefore the matrix elements are as well, both c and u values are limited to the same range regardless of the size of C or the distances themselves.

This setup does not guarantee the triangle inequality to hold; for example, with vectors $v_1 = [-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]$, $v_2 = [0, 1]$ and $v_3 = [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]$ (as seen in Figure A.8), it follows that $v_1 \cdot v_3 = 0$ and $v_1 \cdot v_2 = v_2 \cdot v_3 = \frac{\sqrt{2}}{2}$. Converting these values to STS simply scales them by a constant factor, so this step is skippable. Finally, subtracting these values from 1 (unscaled) leaves us with STS distances $1, 1 - \frac{\sqrt{2}}{2}$ and $1 - \frac{\sqrt{2}}{2}$ respectively, but $2 \cdot (1 - \frac{\sqrt{2}}{2}) < 1$, so the triangle cannot be built.

It should be noted as well that the clustering method makes no use of the summarization training data, which makes it unsupervised with respect to the summarization dataset.

3.4 CZERT classifiers

The summarization models which rely on direct BERT classification of article sentences have quite similar architectures, hence this section will discuss all of them at once.

3.4.1 Architecture

The key idea behind CZERT classifiers is inputting article sentences to CZERT or CZERT-long and directly receiving *logits* loosely correlating to the estimated probabilities of those sentences being part of the extractive summary. We experiment with three such models: single-sentence classifier (Section 2.8.1), single-sentence classifier with context (Section 2.8.2), and multi-sentence classifier (Section 2.8.3).

Single-sentence classifier (ssc) The simplest of each models: a single sentence is tokenized, input into CZERT, the output is *projected* and the output vector is dot-multiplied with a parameter vector $[\Theta_0, \Theta_1, ..., \Theta_{d_e-1}]$ corresponding to a single output neuron (see Figure 3.4). The input sequence length of this CZERT model is 128.

Single-sentence classifier with context and metadata (ssc-ctx-md) This model extends the above model through the introduction of context, as seen in Figure 3.5. The input sequence length is increased to 512, which is an empirically established value maximizing the input size before the quadratic full attention becomes unfeasibly complex.

However, instead of inputting a single sentence to the model, we tokenize the entire article which contains our target sentence. This tokenization differs from the standard by adding appending a [CLS] token after every [SEP] token for the purpose of classification. This results in a sequence
`[CLS] sentence_1 [SEP] [CLS] sentence_2 [SEP] [CLS] ... [SEP]`

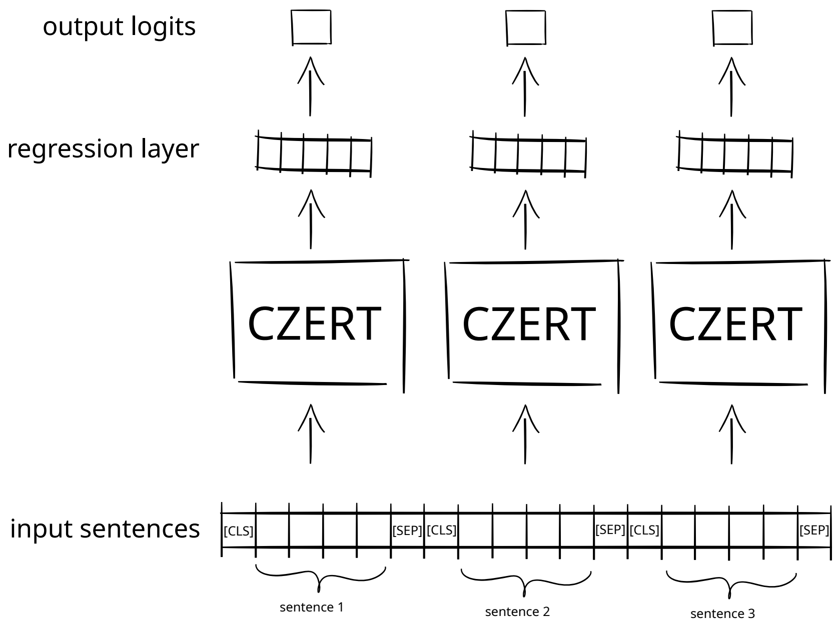


Figure 3.4: Architecture of the single-sentence classifier model.

We consequently reduce this tokenized sequence to 512 tokens by selecting the entire target sentence (which is always possible, see Figure A.6a) and centering it inside a window of size 512 (Figure 3.6a). If such centering would introduce the need for padding, the window is shifted towards the other sequence end until no padding is required (Figures 3.6b and 3.6c). However, if the entire sequence is shorter than 512 tokens, it is input as a whole and padded.

The target sentence, for which the classification happens, is marked using *token type IDs*. For every token belonging to the target sentence (including [CLS] and [SEP]), the token type ID is set to 1, otherwise it is set to 0.

The output is *projected* only with respect to the target sentence. Any remaining tokens are always ignored.

To improve this approach further, we have decided to add specific metadata as features to the output of CZERT before processing the vector by the final one-neuron layer. The metadata in question are the sentence position index with respect to the article which contains this sentence and the position of this article with respect to the entire incident. Basically, these are the two indices used to identify a specific sentence inside the "text" element object from Section 3.1.2. The inclusion of this metadata requires an extension of the final layer weight vector, increasing its size to $d_e + 2$. Choosing not to add the same metadata to the first model is in accordance to the original paper, where no such metadata inclusion is mentioned [GH19].

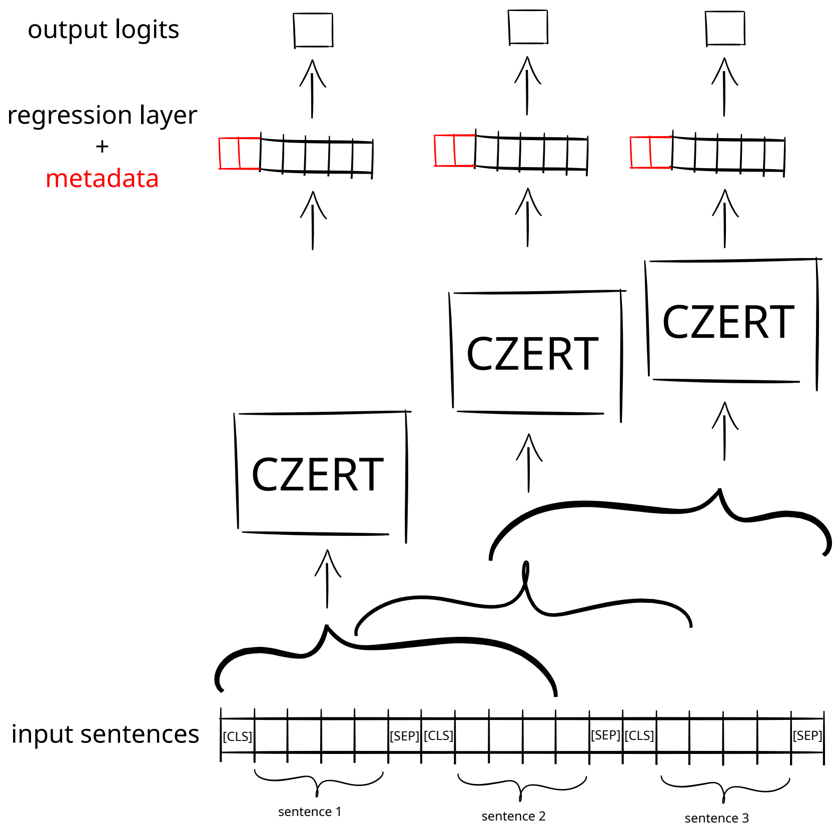


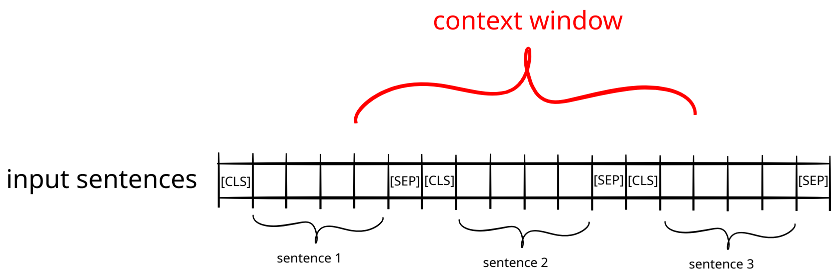
Figure 3.5: Architecture of the single-sentence classifier with context model. The key differences are that the surrounding tokens are included as well at the input, while adding metadata to the penultimate regression layer.

Multi-sentence classifier (msc) This model expects the whole article concatenation to be input (see Figure 3.7), just like in BertSum [LL19]. However, given the fact that the tokenized lengths of concatenated articles reach much longer lengths than 512 almost all the time (see Figure A.6b), the underlying model has to be switched from CZERT to CZERT-long. Clipping the article concatenation does not help, because the model outputs should be extracted from [CLS] tokens at the beginning of each sentence.

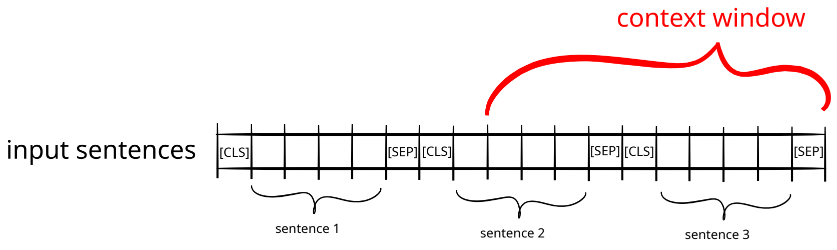
Apart from this change, the model still functions the same way as the previous two, except for the fact that the entire article set is processed in a single forward pass instead of one forward pass for each article sentence.

3.4.2 Training

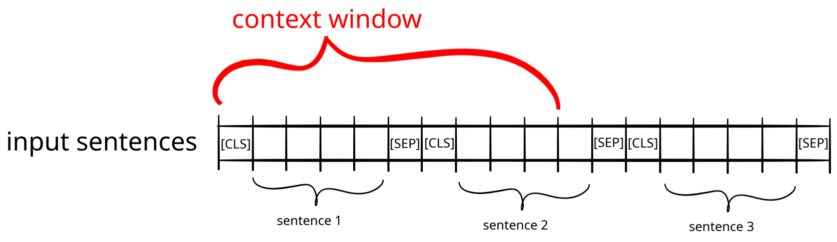
Each one of these models is trained on our summarization dataset using binary cross entropy loss (Equation 3.6). In order to balance the classes for the second model, for each incident, we have purposely left out a fraction



(a) Centering the context window around **sentence 2**.



(b) Centering around **sentence 3**, shifting the context window to the left to avoid padding.



(c) Centering around **sentence 1**, shifting the context window to the right to avoid padding.

Figure 3.6: Different ways of selecting context for the target sentence. Context window is not to scale.

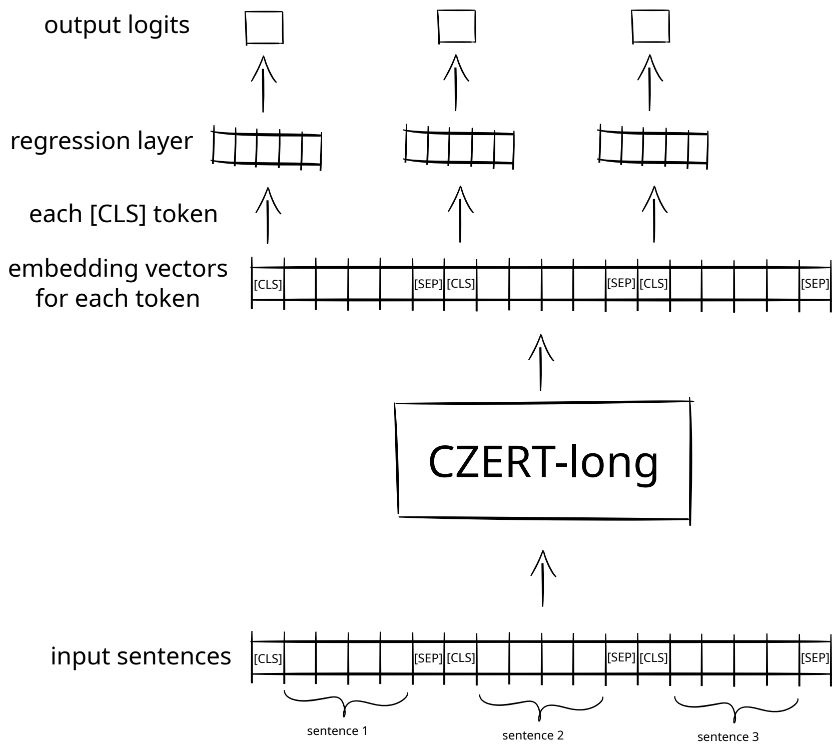


Figure 3.7: Architecture of the multi-sentence classifier model.

of negative samples, because the amount of summary sentences is equal to approximately 0.369 times the amount of non-summary sentences. Again, we chose not to perform the same balancing for the first model in order to more reliably replicate the training conditions proposed by Gu and Hu [GH19].

$$J(y, \hat{y}) = \sum_{i=0}^{n-1} \hat{y}_i \cdot \log y_i + (1 - \hat{y}_i) \cdot \log(1 - y_i) \tag{3.6}$$

3.5 CZERT RNN (rnn-pretr/fntn)

The final model we have decided to experiment on extends BERT by appending a RNN network to its final layer. The idea behind this approach is embedding each article sentence separately, *projecting* the results and passing them into an RNN model, which is suitable for processing sequential data.

3.5.1 Architecture

The vanilla RNN networks suffer from *gradient vanishing*, which is a problem tackled by using gated recurrent cells such as the LSTM [HS97] or GRU

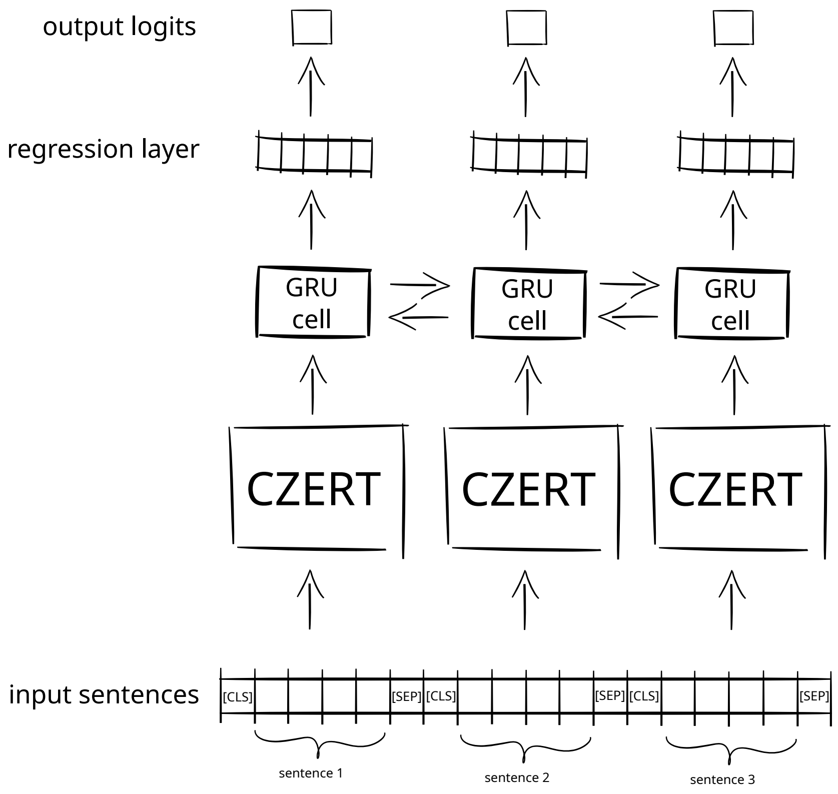


Figure 3.8: Architecture of the multi-sentence classifier model.

[Hoc98]. Since research suggests that these gated cell types are comparable in terms of performance [CGCB14] and our experiments in this environment confirm this assumption, we have opted for using GRU, since it is simpler and therefore faster.

We use the bidirectional variant of GRU, which allows us to utilize both left and right contexts at once, which is impossible in vanilla GRU (or LSTM).

The input sequence size for this CZERT model has been left as the default 128 for embedding sentences.

3.5.2 Training

The training procedure is exactly the same as for the CZERT classifiers in Section 3.4.2, however, we have encountered a major disadvantage of this model compared to the simple classifiers. Given the fact that they had their gradients computed with respect to a single sentence at a time, processing an entire incident requires constant memory with respect to the concatenated articles length. That is not the case for CZERT RNN, as the RNN gradient propagates to all input sentences at once. Such computation required large amounts of memory (over 16GiB for batch size = 1), forcing us to prevent the gradient from propagating to CZERT, which was left untrained. CZERT RNN is therefore the only model not trained end-to-end.

Since we were unable to train the model end-to-end and forced to freeze the CZERT model during training, for experimentation purposes, apart from simply using CZERT as the base of this model (*pretr*), we experimented with using the best performing fine-tuned instance of a single-sentence classifier with context instead (*fntn*). Both versions were evaluated separately.

3.6 Implementation and system architecture

The entire system has been implemented in Python 3.9.7, which is an interpreted language suitable for small projects and data science. This feature comes from the fact that Python is dynamically typed and supports a vast scale of data manipulation operations through libraries like *NumPy* (fast tensor operations, low level), *SciPy* (high level data analysis), *scikit-learn* (general machine learning), *PyTorch* (gpu/distributed tensor operations, gradient calculation, propagation, and descent), and many others. Our system makes great use of all of these libraries, including *Hugging Face*, which provides NLP tools for tokenization and processing inputs by BERT and similar models.

The motivation behind not using the latest Python version comes from the fact that each of these libraries is updated and maintained separately; there is no guarantee that the newest Python version is compatible with any of the newest library versions and *vice versa*.

We have designed this system with two purposes in mind: a) simple design and execution of training experiments, and b) fast and easy inference whilst ensuring availability of all pretrained/fine-tuned/fitted models. This section will discuss related architectural decisions we made in order to fulfill these purposes.

3.6.1 Model swapping

Most models we have implemented in our system take up multiple gigabytes of memory on the GPU. Our system is meant to be run locally, on personal computers; it is therefore safe to assume that it will be very rare for more than one such model to fit on the GPU. However, even if it does, the GPU memory allocated by *PyTorch* (or its alternative, *TensorFlow*) is not automatically released, which can very quickly lead to out-of-memory errors. To

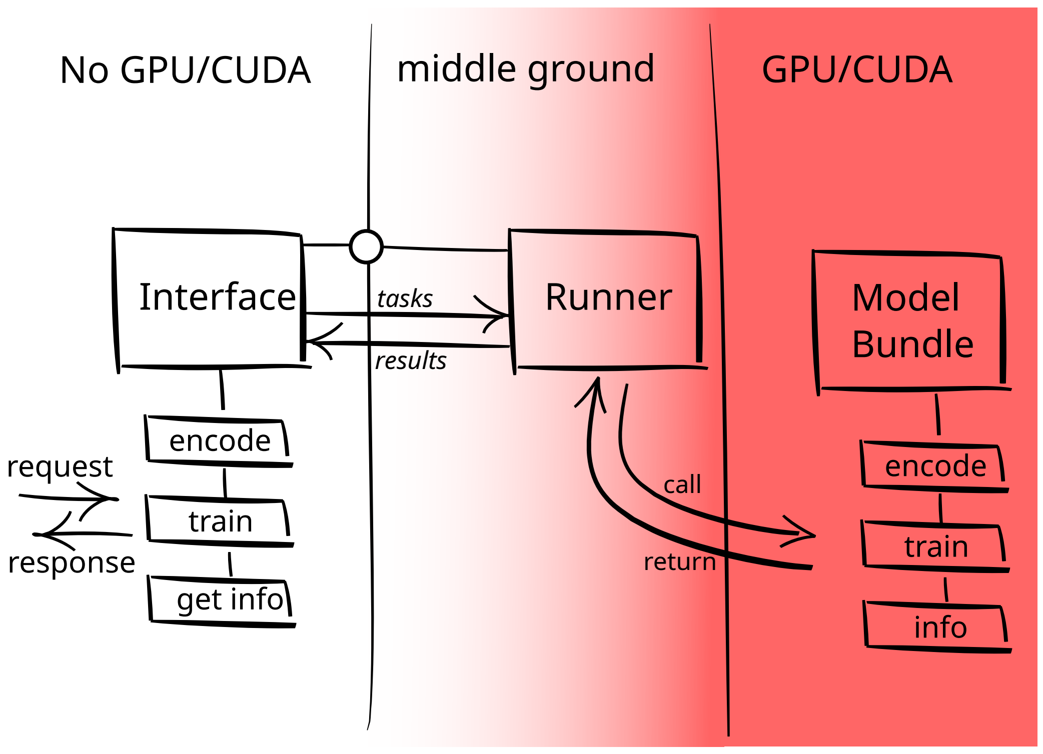


Figure 3.9: Model swapping system architecture. The diagram visualizes the communication process between the **Interface** and **Runner** modules through the CUDA context. The circular connector symbol between **Interface** and **Runner** signalizes our ability to disconnect the **Runner** (killing it) and connecting a new one.

solve this problem regardless of the library used for GPU gradient calculations, we have decided to split the system into two separate modules, the **Interface** and the **Runner**.

The **Runner** module isolates the entire subsystem that is dependent on *PyTorch* directly. Its purpose is to create a *separate process* containing a single **ModelBundle** instance, which bundles a tokenizer and a neural-network model together with relevant method handlers according to whether the bundle is intended for training or for prediction/inference. The **Runner** keeps this bundle running and awaits incoming **CzertTask**s, which are messengers holding target handle identifiers, input data and other required configuration arguments. These **CzertTask**s are read from a single-producer-single-consumer (SPSC) *task* queue suitable for multiprocessing. Each **CzertTask** is handled accordingly and the result is returned through another, *result* SPSC queue.

The **Interface**, as a module, is a set of high-level functions, such as `compute_sts_matrix`, `embed_sentences`, etc., which make use of GPU neural-network data operations. It is responsible for communicating with the **Runner**. Whenever a new request for data processing is created, the **Interface** checks whether the appropriate **ModelBundle** currently runs or not. If it is running, the **Interface** simply generates a **CzertTask** instance and passes it to the **Runner** for processing; if it is not, the **Runner** is killed and therefore, without exceptions, all allocated GPU memory is released. A new **Runner** instance is then created with the appropriate **ModelBundle**, a **CzertTask** instance is generated, and so on. The results of **ModelBundle** handlers are retrieved from the *result* queue and returned to the **Interface** function caller, as shown in Figure 3.9.

Since *Hugging Face* and *PyTorch* models return tensors which are still occupying GPU memory, in order to return them to the **Interface** without introducing its (main) process to the CUDA context, the values are *cleansed*; usually, this means simply converting them from tensors to *NumPy* arrays, which are allocated on RAM, however, when the method handler returns a more complex structure containing references to such tensors, these tensors have to be identified and converted separately.

3.6.2 Data processing

Each summarization model we have evaluated shares a significant portion of its training and evaluation procedures with the other models, from training files to output values. In order to speed up the process of declaring new model experiments and avoid duplicating code, we utilized a pipeline

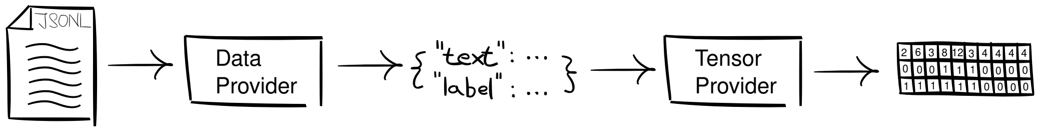


Figure 3.10: The data processing pipeline.

approach for data processing (Figure 3.10).

If a model needs to convert input files to training/evaluation element instances, it uses a subclass of **DataProvider**. **DataProvider** is an abstract class which serves data from an input file according to its implementation. It takes care of properly opening and closing the file.

If a model needs to convert training/evaluation element instances to tensors (in order to process them), it uses a subclass of **TensorProvider**. The **TensorProvider**, too, is an abstract class, which handles sequence tokenization, padding, clipping, pairing elements with labels, batch serving, shuffling and converting data types for compatibility purposes.

Generally speaking, each dataset has its own **DataProvider** derivate, which is shared among all models making use of it. Regarding **TensorProviders**, models are split according to the input format they expect; each such format is implemented in a separate **TensorProvider** derivate. For example, models which predict a value from a document (such as article concatenation) make use of the **EncodeDocumentInputNoOutput** tensor provider. In most cases, the tensor provider for model training and prediction is the same, with the exception of the clustering model from Section 3.3, which expects two sentence inputs during training (Figure 3.3) and one sentence input during inference (Figure 3.2).

3.6.3 Model configuration

In order to make use of this data pipeline pattern, we allocate one source file for *model configuration*. This file defines a **ModelConfig** class, which fully describes each distinct **ModelBundle**, including but not limited to its type (prediction/training), save/restoration paths, model and tokenizer classes, arguments for model and tokenizer initialization, data and tensor providers, training arguments, loss functions, metrics, etc. The **ModelConfig** class itself contains the descriptions of individual **ModelBundles** as attributes and, as such, functions as a factory for **ModelConfig** instances. Defining a brand-new **ModelBundle** is then simply reduced to filling out the required **ModelConfig** descriptions and the bundle is ready to be trained or evaluated, without having to create new code.

Thanks to this approach, performing searches over hyperparameter spaces is also incredibly simple, as one just has to create a specific **ModelConfig** and adjust the attributes one wants to optimize.

3.6.4 Evaluation details

Since the language of our summarization dataset is Czech, which includes many morphological (syntactic) word variants, such as inflection, having little impact on semantics, for the purpose of ROUGE calculations, we have decided to parse the summaries through a *stemmer*, which extracts word stems to a certain degree of precision. Stop words were ignored as well. We have opted for using a stemmer instead of a *lemmatizer* because stemmers are much faster, which makes them the optimal choice for our already heavily time-consuming evaluation pipeline.

All experiments were executed on a MetaCentrum cluster called *adan* using the *gpu* queue. Consequently, they were evaluated on the same summarization evaluation dataset as described in Section 3.1.

4 Discussion

After implementing all models in our system, we have proceeded to run the aforementioned experiments, tracking their progress. Each experiment has been run 15 times, all instances have been evaluated, and the average results, which represent the general summarization capabilities of the underlying models, are presented in the following sections. For an insight into the actual result distributions, see Figures A.1 through A.5.

4.1 Baselines

In order to show the performance of these models in stronger contrast, we have opted for implementing two simple untrained baseline models which select the initial or random sentences respectively. Both baselines have two versions: one which selects 20 sentences and the other selects k sentences of the article concatenation, where k is the expected summary length. The baselines that select a fixed amount of sentences – particularly 20 – exist for the purpose of comparison with other baselines and evaluated summarization approaches; to highlight the difference between knowing and not knowing the amount of reference summary sentences.

method	*rouge-1*	*rouge-2*	*rouge-3*	*rouge-l*	*bertscore*
topk-initial	**0.540**	**0.402**	**0.362**	**0.383**	**0.781**
20-initial	0.525	0.388	0.348	0.368	0.780
topk-random	0.511	0.340	0.291	0.347	0.773
20-random	0.491	0.320	0.272	0.328	0.768

Table 4.1: Overview of the baseline methods, sorted by performance. Bold values are maxima.

It is unsurprising to find that *top-k* baselines outperform fixed, *20* baselines, and that *initial* baselines outperform *random* baselines. However, as we will later infer from Tables 4.2 and 4.3, the *topk-initial* baseline outperforms most of our evaluated models as well, which makes it a very strong baseline given its simplicity. In performance-heavy applications, using it as a main approach should be considered.

4.2 Clustering

method	*rouge-1*	*rouge-2*	*rouge-3*	*rouge-l*	*bertscore*
20-cent	<u>0.330</u>	<u>0.200</u>	<u>0.165</u>	<u>0.209</u>	<u>0.593</u>
20-uniq	0.280	0.158	0.130	0.175	0.552
20-comb	0.288	0.162	0.133	0.182	0.563
topk-cent	<u>0.518</u>	<u>0.364</u>	<u>0.319</u>	<u>0.357</u>	<u>0.776</u>
topk-uniq	0.517	0.363	0.317	0.355	0.775
topk-comb	0.517	0.362	0.317	0.356	0.775
loc-orcl-cent	<u>0.504</u>	<u>0.371</u>	<u>0.327</u>	<u>0.359</u>	<u>0.726</u>
loc-orcl-uniq	0.459	0.321	0.278	0.313	0.685
loc-orcl-comb	0.487	0.352	0.308	0.345	0.716
glob-orcl-cent	<u>**0.541**</u>	<u>**0.383**</u>	<u>**0.335**</u>	<u>**0.375**</u>	<u>**0.788**</u>
glob-orcl-uniq	0.528	0.363	0.313	0.359	0.779
glob-orcl-comb	0.534	0.373	0.324	0.369	0.786

Table 4.2: Clustering results grouped into four groups by clustering type: *20*, *topk*, *loc-orcl* and *glob-orcl*, which stand for 1. 20 clusters, 2. k clusters (equal to expected summary size), 3. local distance oracle and 4. global distance oracle (cutoff distance = 2.5) respectively; see Section 3.3.6, paragraph *Clustering*. Each clustering method then examines three ways of centroid selection: *cent*, *uniq*, and *comb*, which stand for "by centrality", "by uniqueness" and "by centrality and uniqueness (combined)", as described in Section 3.3.6, paragraph *Centroid selection*. Underlined values are group maxima, bold values are global maxima.

Centroid selection Our clustering experiments show that *selection by centrality* (Section 3.3.6) is the optimal centroid selection approach, as it improves the results across all four clustering approaches in Table 4.2. However, the difference is very slight, as shown in both the results table and across Figures A.7a, A.7b and A.7c, compared to the impact the clustering method itself has on the final results.

Clustering by cluster count Trying to perform clustering given target cluster count 20, which is an approach comparable to our baselines, significantly under-performs in comparison to the *topk* cluster count approach, which uses the same amount of clusters as the amount of sentences in our reference summary. This tells us that if we wanted to use clustering for summarization according to cluster count, specifying the summary size would be absolutely necessary.

Clustering by cutoff distance On the other hand, the data in Table 4.2 suggest that clustering by cutoff distance may be the best approach. Since we were unable to estimate a good cutoff distance empirically, we have developed two distance oracle variants as described in Section 3.3.6, which are to be regarded as performance ceiling estimators for summarization through clustering. The global distance oracle, which is equivalent to clustering by cutoff STS distance = 2.5, slightly outperforms the local distance oracle, which selects the cutoff STS distance individually in order to maximize binary F1 measure (BF1). (The local oracle has around 0.09 better BF1 on average.) This means that the BF1 does not necessarily correlate with our other evaluation metrics.

Furthermore, since the meaningful cutoff distance space is limited to the range $(0, 6)$, claiming that a best specific cutoff distance exists makes much more sense than, for example, claiming that a best amount of clusters exists. This leads us to the conclusion that clustering by cutoff distance, specifically 2.5, may be the best clustering method for summarization, but evaluation on more datasets is required to confirm this.

4.3 Supervised approaches

Thresholding vs. top-k Our supervised summarization experiments in Table 4.3 show that, regarding the method of logit-to-classes conversion (Section 3.2.4), the difference between *thresholding* and *top-k* approaches is significant. Almost every single model improves across the entire scale of metrics when it knows how many sentences its output summary should have. The sole exception is the best-performing model *ssc-ctx* – the single-sentence classifier with context – which actually performs *worse*. We have examined the cause of this and found that the reason the ROUGE-N, ROUGE-L, and BertScore F1 measures (not to be confused with classification, binary F1 measure) are so high for *ssc-ctx-md* is that the model has very high *thresholding* recall at the cost of precision; see Figures A.1 through A.5 for more details. Essentially, this means that the model outputs very similar logits, most of which are positive. Consequently, the model tends to create overly long summaries, which human judges would not consider very good, even though the model maximizes all of the aforementioned metrics with the

method	*rouge-1*	*rouge-2*	*rouge-3*	*rouge-l*	*bertscore*
thr-ssc	0.372	0.270	0.240	0.281	0.610
thr-ssc-ctx-md	<u>**0.605**</u>	<u>**0.485**</u>	<u>**0.447**</u>	<u>**0.464**</u>	<u>0.790</u>
thr-msc	0.495	0.337	0.291	0.339	0.754
thr-rnn-pretr	0.437	0.286	0.246	0.290	0.713
thr-rnn-fntn	0.057	0.026	0.020	0.036	0.371
topk-ssc	<u>0.565</u>	<u>0.420</u>	<u>0.376</u>	<u>0.429</u>	<u>**0.801**</u>
topk-ssc-ctx-md	0.542	0.405	0.366	0.406	0.790
topk-msc	0.510	0.345	0.297	0.349	0.772
topk-rnn-pretr	0.508	0.346	0.300	0.345	0.770
topk-rnn-fntn	0.520	0.371	0.328	0.361	0.770

Table 4.3: Supervised summarization results grouped into two groups by logits-to-classes conversion: *thr* and *topk*, which stand for "thresholding" and "top-k" respectively. The individual approaches are *ssc*, *ssc-ctx-md*, *msc*, *rnn-pretr* and *rnn-fntn* which stand for 1. single-sentence classifier (Section 3.4), 2. single-sentence classifier with context and metadata (Section 3.4), 3. multi-sentence classifier (Section 3.4), 4. GRU with CZERT embeddings (Section 3.5) and 5. GRU with the best *ssc-ctx* embeddings (Section 3.5) respectively. Underlined values are group maxima, bold values are global maxima.

exception of BertScore.

method	*rouge-1*	*rouge-2*	*rouge-3*	*rouge-l*	*bertscore*
thr-ssc	0.372	0.270	0.240	0.281	0.610
thr-ssc-ctx-md	<u>**0.605**</u>	**0.485**	**0.447**	<u>**0.464**</u>	<u>0.790</u>
thr-ssc-ctx-nomd	**0.597**	<u>**0.489**</u>	<u>**0.453**</u>	**0.461**	<u>0.790</u>
topk-ssc	<u>0.565</u>	<u>0.420</u>	<u>0.376</u>	<u>0.429</u>	<u>**0.801**</u>
topk-ssc-ctx-md	0.542	0.405	0.366	0.406	0.790
topk-ssc-ctx-nomd	0.531	0.380	0.336	0.376	0.777

Table 4.4: Comparison between single-sentence classifiers methods where context is added (*ctx*) and positional metadata is added (*md*) or excluded (*nomd*). Underlined values are local maxima, bold values are either global maxima or near global maxima.

Comparison between single-sentence classifiers The same pattern – *thresholding* outpeforming *top-k* – is not observed in the vanilla single-sentence classifier *ssc*. Therefore, adding positional metadata and/or context to a single sentence causes the model's to correctly decrease its amount of false negatives significantly, but at the cost of a decreased true negative rate. In order to narrow down the cause of this performance shift, we ran an additional experiment, *ssc-ctx-nomd*, which excludes the addition of positional metadata to the penultimate model layer. The comparison between all three models is captured by Table 4.4. The first thing to notice is the fact that *both ctx* methods perform better under *thresholding* than *top-k*, as opposed to the simple *ssc* method. This means that it is actually the addition of *context*, not *positional metadata*, to the *ssc*, which causes the model to have such low false-negative rate.

Furthermore, regarding thresholding only, the models are comparable – almost equivalent – in performance. This would – without further investigation – make us conclude that the inclusion of positional metadata in *thr-ssc-ctx* is an irrelevant feature. However, Figures A.1 through A.5 show us that without metadata, the model's recall becomes 1.0 and its precision drops further; the model stops being able to differentiate between individual sentences and the logits it outputs all become positive, rendering them unusable for thresholding. We assume the reason the inclusion of context hinders the model's ability to distinguish between individual sentences is that marking the central sentence by updated token type IDs (Section 3.4) is insufficient. Regardless, the *thr-ssc-ctx-md* remains to be the best approach for summarization with unknown amount of target sentences.

The observed results lead us to two conclusions about the vanilla ssc – without context. First, the ssc is not capable of deciding whether a given sentence belongs to a summary or not, confirming the results obtained by [GH19], but they are effective at comparing whether one sentence is more suitable for a summary than another (*topk-ssc*). Second, the information that the model bases this comparison upon must clearly be contained within the individual sentences. This means that there are certain phrases or words in our CNA summarization dataset which are more likely to appear in a summary than not.

GRU models The improvement of *top-k* over *thresholding* is most significant in the GRU model with a fine-tuned *ssc-ctx* as an embedder. In Figures A.1 through A.5, we can observe that the cause of such low metric scores is its very low *recall*. This is caused by the model's tendency to output negative logits in general. However, when selecting the *top-k* logits as positive answers instead, the GRU with a fine-tuned *ssc-ctx* performed best out of all its variants. Hence, if the GRU model cannot be trained end-to-end, finetuning the underlying CZERT embedder is recommended.

method	*rouge-1*	*rouge-2*	*rouge-3*	*rouge-l*	*bertscore*
thr-msc	0.495	0.337	0.291	0.339	0.754
thr-msc-md	0.338	0.238	0.210	0.243	0.594
thr-msc-mean-nomd	0.464	0.342	0.306	0.349	0.695
thr-msc-mean-md	0.386	0.283	0.252	0.288	0.618
topk-msc	0.510	0.345	0.297	0.349	0.772
topk-msc-md	0.555	0.409	0.365	0.411	0.798
topk-msc-mean-nomd	**0.563**	**0.416**	**0.371**	**0.424**	**0.802**
topk-msc-mean-md	0.561	0.415	0.371	0.421	0.801

Table 4.5: Comparison between multi-sentence classifier methods where we compare 1. the default *msc* architecture as described in Section 3.4, 2. *msc* with added positional metadata to the penultimate layer (*msc-md*), 3. *msc* with *mean aggregative projection* without (*msc-mean-nomd*) and 4. with (*msc-mean-md*) metadata. Underlined values are group maxima, bold values are global maxima.

Multi-sentence classifiers (Longformer) We were surprised by the apparent lack of quality of our *msc* model in comparison to the *ssc* models in Table 4.3, given the fact that the *msc* has the entire article context ready for processing. We have identified two significant differences between these two approaches. First, the *ssc-ctx* includes positional metadata. This – we theorized – should be irrelevant for *msc* thanks to its positional encodings. It should be noted that the positional metadata in the context of *msc* are always the same vectors $[0, 0], [0, 1], [0, 2], ..., [1, 0], ..., [m, n]$, where $m + 1$ is the amount of input articles and $n + 1$ is the amount of setences in the last article. Second, both *ssc* and *ssc-ctx* make use of the *mean aggregative projection* over sentences when converting the CZERT output matrix to a single vector, as opposed to *msc*, which selects the [CLS] tokens corresponding to each sentence instead. In order to compare these two model architectures in detail, we have designed and performed additional experiments with the *msc*; the results can be seen in Table 4.5.

Regarding *thresholding*, which we have already shown to be generally a sub-optimal way to extract binary classes from logits in this domain, neither the updated projection nor adding metadata helped improve the model's performance significantly. Figures A.1 through A.5 show that the training of these models tends to result in them returning similar logits, which in turn causes them either exhibit high recall and small precision or *vice versa.*

On the other hand, the *top-k* approaches improved significantly, particularly by the employment of the *mean aggregative projection* over individual sentences. Moreover, when this projection is utilized, the metadata addition becomes obsolete. Indeed, using the aforementioned projection instead of [CLS] token representations *should* be better in general, as all [CLS] token representations are very similar because of the way global attention works in Longformer (Figure 2.3).

4.4 Final comparison

type	*method*	*rouge-1*	*rouge-2*	*rouge-3*	*rouge-l*	*bertscore*
base	*20-initial*	0.525	0.388	0.348	0.368	0.780
	topk-initial	0.540	0.402	0.362	0.383	0.781
	20-random	0.491	0.320	0.272	0.328	0.768
	topk-random	0.511	0.340	0.291	0.347	0.773
unsup.	*glob-orcl-cent*	0.541	0.383	0.335	0.375	0.788
	loc-orcl-cent	0.504	0.371	0.327	0.359	0.726
	topk-cent	0.518	0.364	0.319	0.357	0.776
sup.	*thr-ssc-ctx-md*	**0.605**	**0.485**	**0.447**	**0.464**	0.790
	topk-ssc	0.565	0.420	0.376	0.429	0.801
	topk-msc-mean-nomd	0.563	0.416	0.371	0.424	**0.802**

Table 4.6: Overview of the best methods compared to our baselines. Underlined values are instances where an evaluated method was better than all of our baselines. Bold values are global maxima.

The best model The results in Table 4.6 show clear dominance of the supervised approaches over the unsupervised approaches, leaving us to conclude that the multi-sentence classifier with mean aggregative projection is the best model overall, as it maximizes BertScore, which has been shown to correlate better with human annotators. Suffice to say, we used *topk-msc-mean-nomd* for implementing our summarization system. However, this approach is applicable only if the amount of summary sentences is known. Therefore, to complement this model, we make use of the *thr-ssc-ctx-md*, which tends to return large summaries, but it turns out to be the strongest model for solving the problem of unknown target summary size.

Clustering We can also see that the *topk-initial* baseline, which selects the initial k sentences, is stronger than clustering with an oracle for distance according to some of the evaluation metrics, and therefore clustering in general. Such revelation reflects very negatively on any neural-network-based method, since their execution takes significantly more time (Table 4.7).

type	*archetype*	*speedup*
supervised	*ssc-ctx*	1.0x
	ssc	3.1x
	rnn	4.8x
	msc	7.7x
unsupervised	*dist-clust*	9.6x
	top-k-clust	9.6x
	20-clust	9.6x
baseline	*random*	520,000x
	initial	2,200,200x

Table 4.7: Speedup of individual model archetypes in comparison to the slowest one: single-sentence classifier with context. We have let each model archetype to summarize every element from our test summarization dataset and measured the time it took. The performance benchmark has been executed on an Nvidia GeForce GTX 1050 Ti GPU and Intel(R) Core(TM) i5-4440 CPU @ 3.10GHz. The benchmark conditions were loose; its purpose was not to arrive at exact results, but to provide us with an approximate relative performance overview.

Speedup To assess the relative execution time of each summarization model, we have performed a simple benchmark whose results are displayed in Table 4.7. Notice that all clustering methods take almost the same time; clearly, the CZERT pass is the bottleneck here. The single-sentence classifiers take the longest time, as they process the entire article concatenation as individual sentences separately. The GRU processes the input similarly, however, the sentence embeddings it produces as a middle step are dependent on one another. Therefore, the entire article concatenation must be input to the model as a whole, hence it can calculate the sentence embeddings batch-wise, which causes a minor speedup compared to *ssc*, even though the embeddings are passed through a GRU layer. Overall, *msc* is the fastest supervised method, as the underlying Longformer processes the article concatenation in a single – albeit complex – forward pass.

4.5 Conclusion

During the preparation of this thesis, we have sufficiently analyzed the state of the art of multi-document summarization, focusing primarily on the applications of neural-network models, such as BERT and Longformer. We have successfully prepared a summarization dataset from CNA incidents, which we used for the training and evaluation of these models, all of which were trained using the MetaCentrum grid service. Consequently, we have compared and critically evaluated the quality of their summarization capabilities in comparison to a set of simple summarization baseline approaches we have implemented. The results we obtained have led us to perform additional experiments in order to truly understand the reasons behind the under/over-performance of models compared to our expectations. The conclusion of said experiments lead us to implement our summarization system using the multi-sentence classifier based on CZERT-long [SPP$^+$21], which is inspired by BertSum [LL19] and represents sentences by reducing all sentence token vectors into a single vector by averaging. Hence, we have fulfilled all requirements for successfull completion of this thesis. Future work should primarily focus on abstractive summarization methods using Transformer-based architectures such as BART.

Bibliography

[ACM$^+$18] Flora Amato, Aniello Castiglione, Vincenzo Moscato, Antonio Picariello, and Giancarlo Sperlì. Multimedia summarization using social media content. *Multimedia Tools and Applications*, 77(14):17803–17827, 2018.

[AL08] Abhaya Agarwal and Alon Lavie. Meteor, m-bleu and m-ter: Evaluation metrics for high-correlation with human rankings of machine translation output. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 115–118, 2008.

[APA$^+$17] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saeid Safaei, Elizabeth D Trippe, Juan B Gutierrez, and Krys Kochut. Text summarization techniques: a brief survey. *arXiv preprint arXiv:1707.02268*, 2017.

[APPH16] Andry Alamsyah, Marisa Paryasto, Feriza J Putra, and Rizal Himmawan. Network text analysis to summarize online conversations for marketing intelligence efforts in telecommunication industry. In *2016 4th International Conference on Information and Communication Technology (ICoICT)*, pages 1–5. IEEE, 2016.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[BLL$^+$15] Lidong Bing, Piji Li, Yi Liao, Wai Lam, Weiwei Guo, and Rebecca J Passonneau. Abstractive multi-document summarization via phrase selection and merging. *arXiv preprint arXiv:1506.01597*, 2015.

[BPC20] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[CA13] Freddy Chong Tat Chua and Sitaram Asur. Automatic summarization of events from social media. In *Seventh international AAAI conference on weblogs and social media*, 2013.

[CB18] Yen-Chun Chen and Mohit Bansal. Fast abstractive summarization with reinforce-selected sentence rewriting. *arXiv preprint arXiv:1805.11080*, 2018.

[CBOK06] Chris Callison-Burch, Miles Osborne, and Philipp Koehn. Re-evaluating the role of bleu in machine translation research. In

11th Conference of the European Chapter of the Association for Computational Linguistics, 2006.

[CGCB14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[DHD20] Esin Durmus, He He, and Mona Diab. Feqa: A question answering evaluation framework for faithfulness assessment in abstractive summarization. *arXiv preprint arXiv:2005.03754*, 2020.

[Dug21] Pranay Dugar. Attention, seq2seq models, Jul 2021.

[EBE19] Matan Eyal, Tal Baumel, and Michael Elhadad. Question answering as an automatic evaluation metric for news article summarization. *arXiv preprint arXiv:1906.00318*, 2019.

[FRK04] Marcelo Fiszman, Thomas C Rindflesch, and Halil Kilicoglu. Abstraction summarization for managing the biomedical research literature. In *Proceedings of the computational lexical semantics workshop at HLT-NAACL 2004*, pages 76–83, 2004.

[GH19] Yang Gu and Yanke Hu. Extractive summarization with very deep pretrained language model. *International Journal of Artificial Intelligence and Applications*, 10(2):27–32, 2019.

[GL12] Pierre-Etienne Genest and Guy Lapalme. Fully abstractive approach to guided summarization. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 354–358, 2012.

[GMCK00] Jade Goldstein, Vibhu O Mittal, Jaime G Carbonell, and Mark Kantrowitz. Multi-document summarization by sentence extraction. In *NAACL-ANLP 2000 Workshop: Automatic Summarization*, 2000.

[Gre11] Charles Greenbacker. Towards a framework for abstractive summarization of multimodal documents. In *Proceedings of the ACL 2011 Student Session*, pages 75–80, 2011.

[HAM10] Sonia Haiduc, Jairo Aponte, and Andrian Marcus. Supporting program comprehension with source code summarization. In *2010 acm/ieee 32nd international conference on software engineering*, volume 2, pages 223–226. IEEE, 2010.

[HG16] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[HLL$^+$18] Wan-Ting Hsu, Chieh-Kai Lin, Ming-Ying Lee, Kerui Min, Jing Tang, and Min Sun. A unified model for extractive and abstractive summarization using inconsistency loss. *arXiv preprint arXiv:1805.06266*, 2018.

[Hoc98] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[Hor18] Rani Horev. Bert explained: State of the art language model for nlp, November 2018.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[JR18] Aishwarya Jadhav and Vaibhav Rajan. Extractive summarization with swap-net: Sentences and words from alternating pointer networks. In *Proceedings of the 56th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 142–151, 2018.

[Lin04] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[LL19] Yang Liu and Mirella Lapata. Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345*, 2019.

[LN19] Hui Lin and Vincent Ng. Abstractive summarization: A survey of the state of the art. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9815–9822, 2019.

[LRFP13] Elena Lloret, María Teresa Romá-Ferri, and Manuel Palomar. Compendium: A text summarization system for generating abstracts of research papers. *Data & Knowledge Engineering*, 88:164–175, 2013.

[LSL18] Logan Lebanoff, Kaiqiang Song, and Fei Liu. Adapting the neural encoder-decoder framework from single to multi-document summarization. *arXiv preprint arXiv:1808.06218*, 2018.

[Mat18] Goran Matošević. Text summarization techniques for meta description generation in process of search engine optimization. In *Computer Science On-line Conference*, pages 165–173. Springer, 2018.

[MCN14] Yashar Mehdad, Giuseppe Carenini, and Raymond Ng. Abstractive summarization of spoken and written conversations based on phrasal queries. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1220–1230, 2014.

[MHK⁺99] Inderjeet Mani, David House, Gary Klein, Lynette Hirschman, Therese Firmin, and Beth M Sundheim. The tipster summac text summarization evaluation. In *Ninth Conference of the European Chapter of the Association for Computational Linguistics*, pages 77–85, 1999.

[Mil19] Derek Miller. Leveraging bert for extractive text summarization on lectures. *arXiv preprint arXiv:1906.04165*, 2019.

[Mül11] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*, 2011.

[OMCN14] Tatsuro Oya, Yashar Mehdad, Giuseppe Carenini, and Raymond Ng. A template-based abstractive meeting summarization: Leveraging summary and source text relationships. In *Proceedings of the 8th International Natural Language Generation Conference (INLG)*, pages 45–53, 2014.

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[Pop17] Igor Popov. Malware detection using machine learning based on word2vec embeddings of machine code instructions. In *2017 Siberian symposium on data science and engineering (SSDSE)*, pages 1–4. IEEE, 2017.

[PRWZ02] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[PY09] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[RF00] Dragomir Radev and Weiguo Fan. Automatic summarization of search engine hit lists. In *ACL-2000 Workshop on Recent Advances in Natural Language Processing and Information Retrieval*, pages 99–109, 2000.

[RG19] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[Ros00] Ronald Rosenfeld. Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE*, 88(8):1270–1278, 2000.

[SCJ+20] Kaushik Sekaran, P Chandana, J Rethna Virgil Jeny, Maytham N Meqdad, and Seifedine Kadry. Design of optimal search engine using text summarization through artificial intelligence techniques. *Telkomnika*, 18(3):1268–1274, 2020.

[SPP+21] Jakub Sido, Ondřej Pražák, Pavel Přibáň, Jan Pašek, Michal Seják, and Miloslav Konopík. Czert–czech bert-like model for language representation. *arXiv preprint arXiv:2103.13031*, 2021.

[SSP+21] Jakub Sido, Michal Seják, Ondřej Pražák, Miloslav Konopík, and Václav Moravec. Czech news dataset for semantic textual similarity. *arXiv preprint arXiv:2108.08708*, 2021.

[SSTW01] Michael J Shaw, Chandrasekar Subramaniam, Gek Woo Tan, and Michael E Welge. Knowledge management and data mining for marketing. *Decision support systems*, 31(1):127–137, 2001.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[WC13] Lu Wang and Claire Cardie. Domain-independent abstract generation for focused meeting summarization. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1395–1405, 2013.

[WLZ+19] Qicai Wang, Peiyu Liu, Zhenfang Zhu, Hongxia Yin, Qiuyue Zhang, and Lindong Zhang. A text abstraction summary model based on bert word embedding and reinforcement learning. *Applied Sciences*, 9(21):4701, 2019.

[WSC$^+$16] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[YM02] Mark D Yandell and William H Majoros. Genomics and natural language processing. *Nature Reviews Genetics*, 3(8):601–610, 2002.

[ZKW$^+$19] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2019.

5 Common abbreviations

abbreviation	*meaning*
ssc	single-sentence classifier
ssc-ctx	single-sentence classifier with context
ssc-ctx-md	single-sentence classifier with context and positional metadata
msc	multi-sentence classifier
rnn-pretr	GRU with CZERT embeddings
rnn-fntn	GRU with finetuned *ssc-ctx* embeddings
cent	centrality
uniq	uniqueness
comb	centrality and uniqueness combined
thr	thresholding (if logit > 0 true, else false)
topk	if logit is one of the largest k true, else false;
BERT	Bidirectional Encoder Representations from Transformers
CNA/ČTK	Czech News Agency / Česká tisková kancelář
CZERT	Czech BERT
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
SDS	single-document summarization
MDS	multi-documents summarization
LSTM	long-short term memory
GRU	gated recurrent unit
RELU	rectified linear unit
STS	semantic textual similarity

A Appendix

A.1 User manual

1. Make sure to have Python 3.9+ and `pip` installed on your system.
2. Navigate to `Applications_and_libraries`.
3. Change `target_device.txt` content to `cuda` if you wish to use GPU for forward passes.
4. Run `run.sh`.
5. Open `localhost:5000/static/index.html` in your browser.

If the above approach does not work for you, try using the latest Python 3.9 and `requirements_old.txt`, which include specific versions of the required libraries.

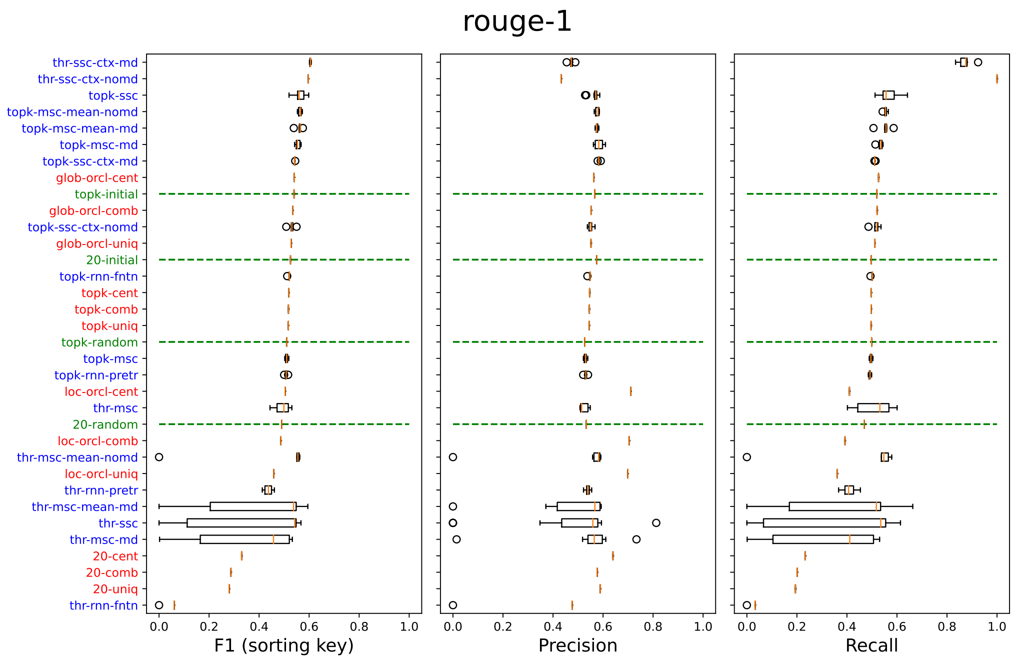


Figure A.1: Sorted ROUGE-1 boxplots. Red color signalizes clustering, blue color signalizes a supervised approach, and green color signalizes a baseline.

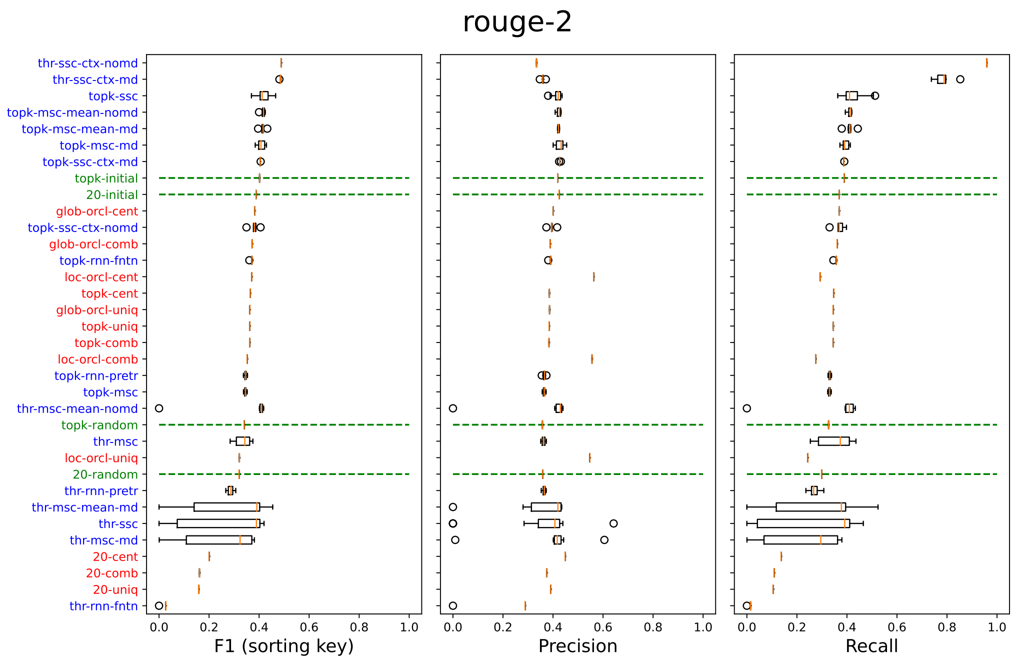


Figure A.2: Sorted ROUGE-2 boxplots. Red color signalizes clustering, blue color signalizes a supervised approach, and green color signalizes a baseline.

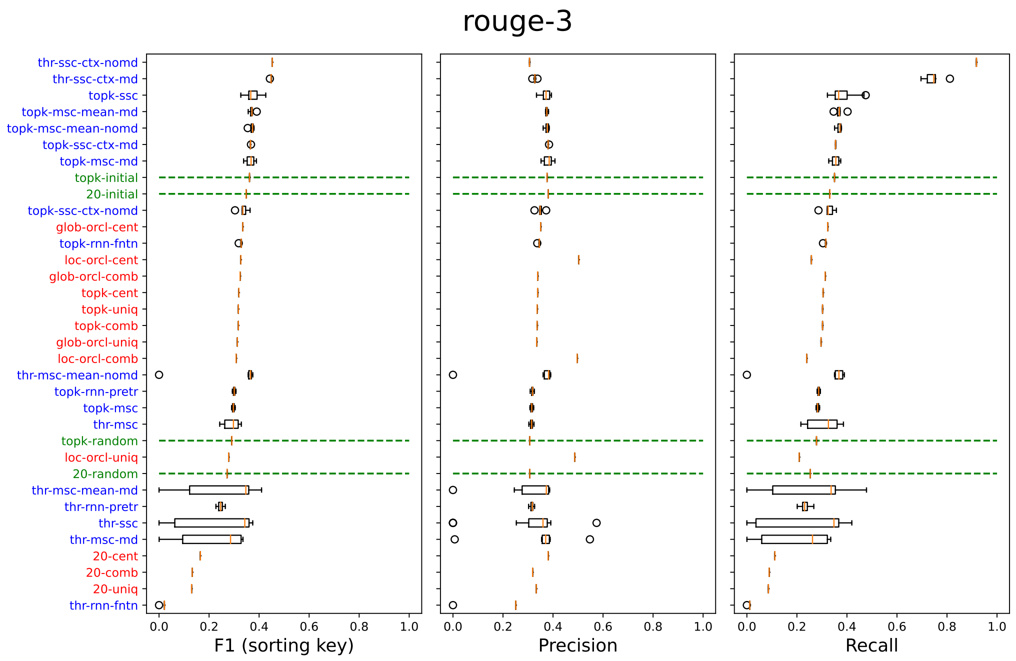


Figure A.3: Sorted ROUGE-3 boxplots. Red color signalizes clustering, blue color signalizes a supervised approach, and green color signalizes a baseline.

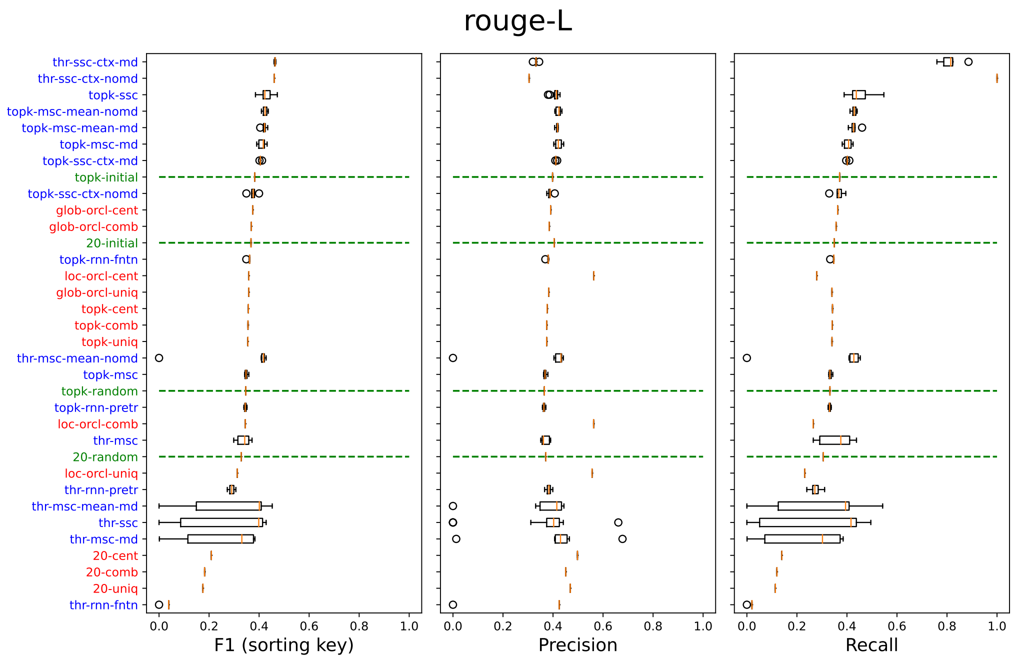


Figure A.4: Sorted ROUGE-L boxplots. Red color signalizes clustering, blue color signalizes a supervised approach, and green color signalizes a baseline.

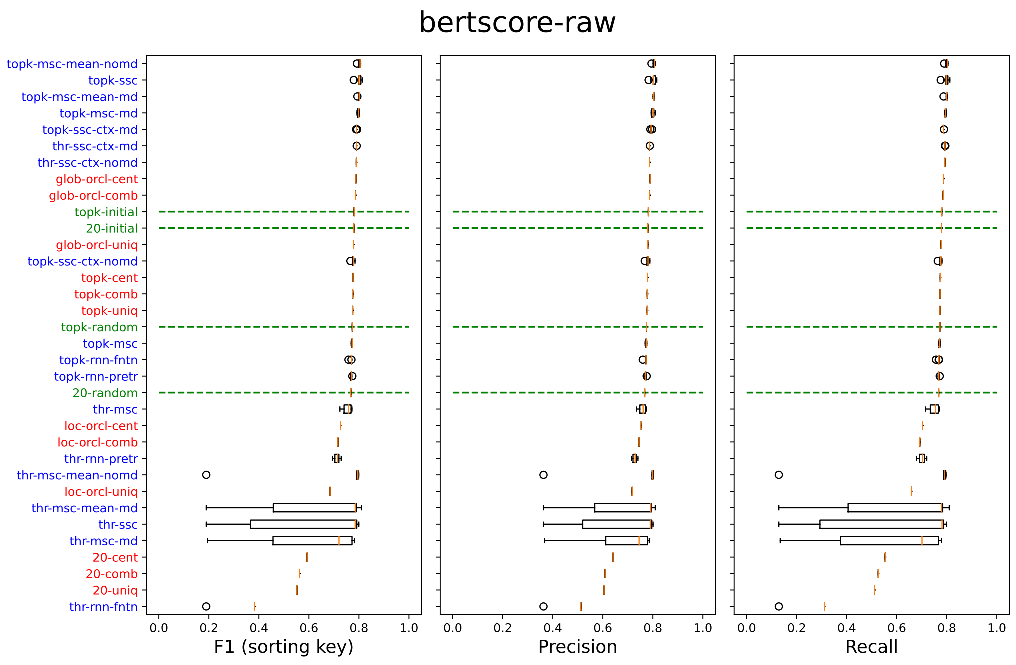
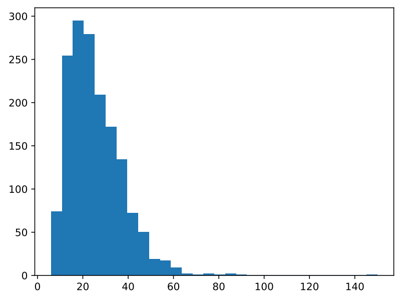
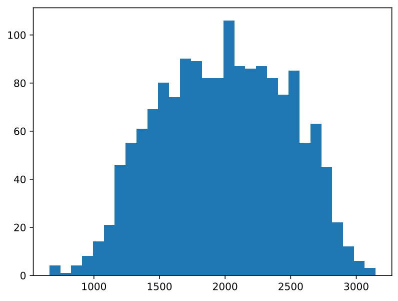


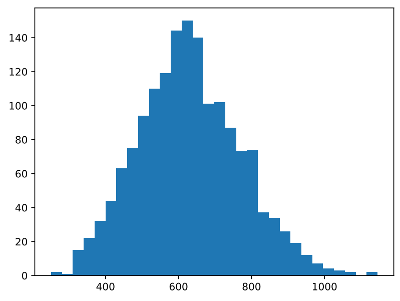
Figure A.5: Sorted BertScore boxplots. Red color signalizes clustering, blue color signalizes a supervised approach, and green color signalizes a baseline.



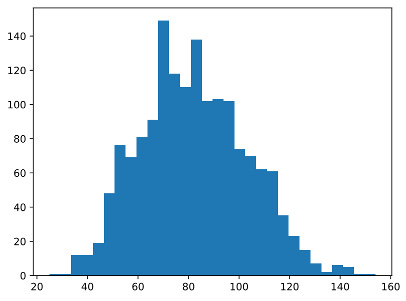
(a) Amount of tokens in each sentence.



(b) Amount of tokens in an article concatenation.

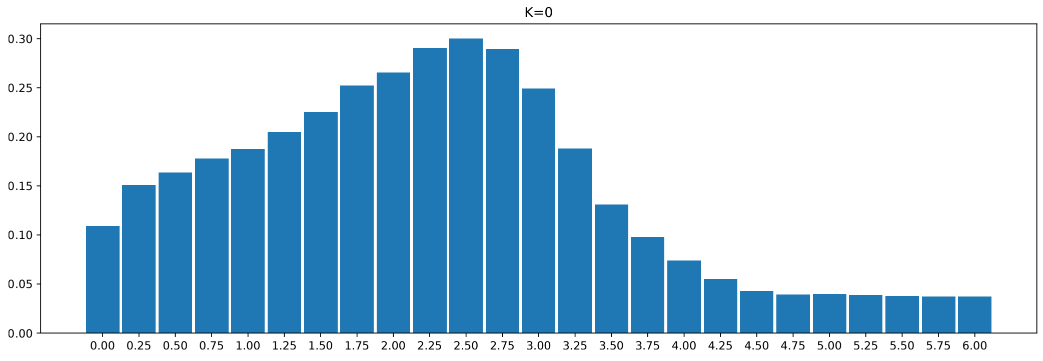


(c) Amount of tokens in the original abstractive summaries.

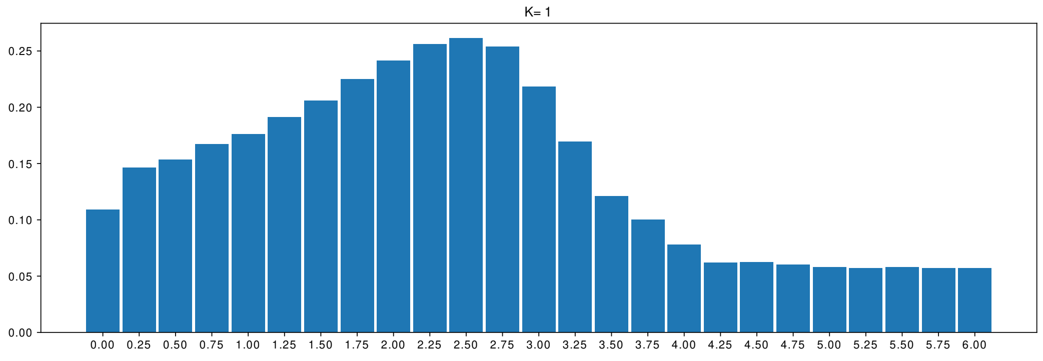


(d) Amount of sentences in an article concatenation.

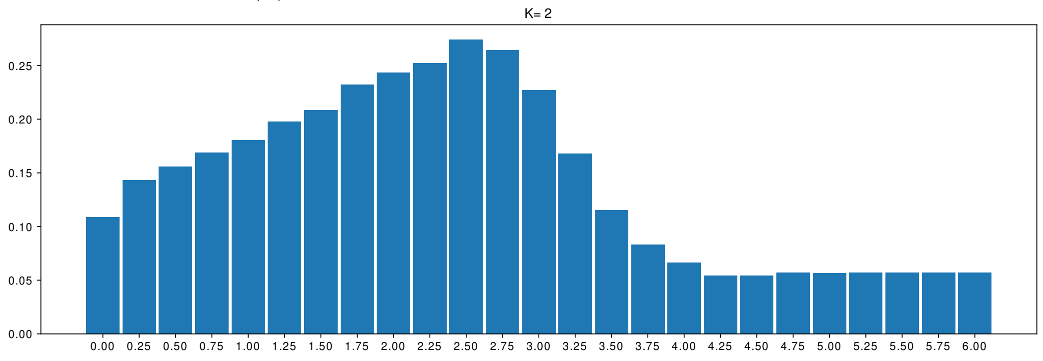
Figure A.6: A compilation of histograms describing the summarization dataset. The X axis represents bucketized token lengths, the Y axis represents the number of elements in individual buckets.



(a) Average F1 with respect to *centrality*.



(b) Average F1 with respect to *uniqueness*.



(c) Average F1 with respect to the combination of *centrality* and *uniqueness*.

Figure A.7: The average F1 measure between ground-truth labels and labels obtained from performing agglomerative hieararchical clustering with distance cutoff *d* and centroid selection by *centrality* using our STS distance matrices.. Other centroid selection approaches yielded a similar shape and shared the same *argmax*, see Section 3.3.6, paragraph Centroid selection for further details. The X axis represents values of *d*, the Y axis shows the average F1 measure.

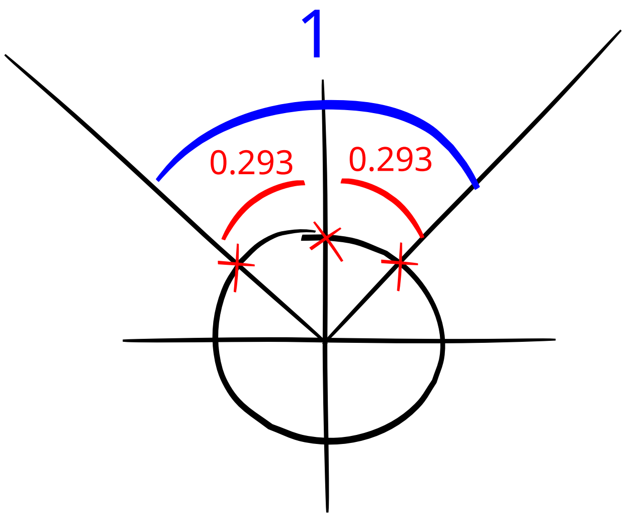


Figure A.8: Visualization of example vectors v_1, v_2 and v_3 and the cosine (STS) distances between them for the purpose of showing that cosine distance does not satisfy the triangle inequality.

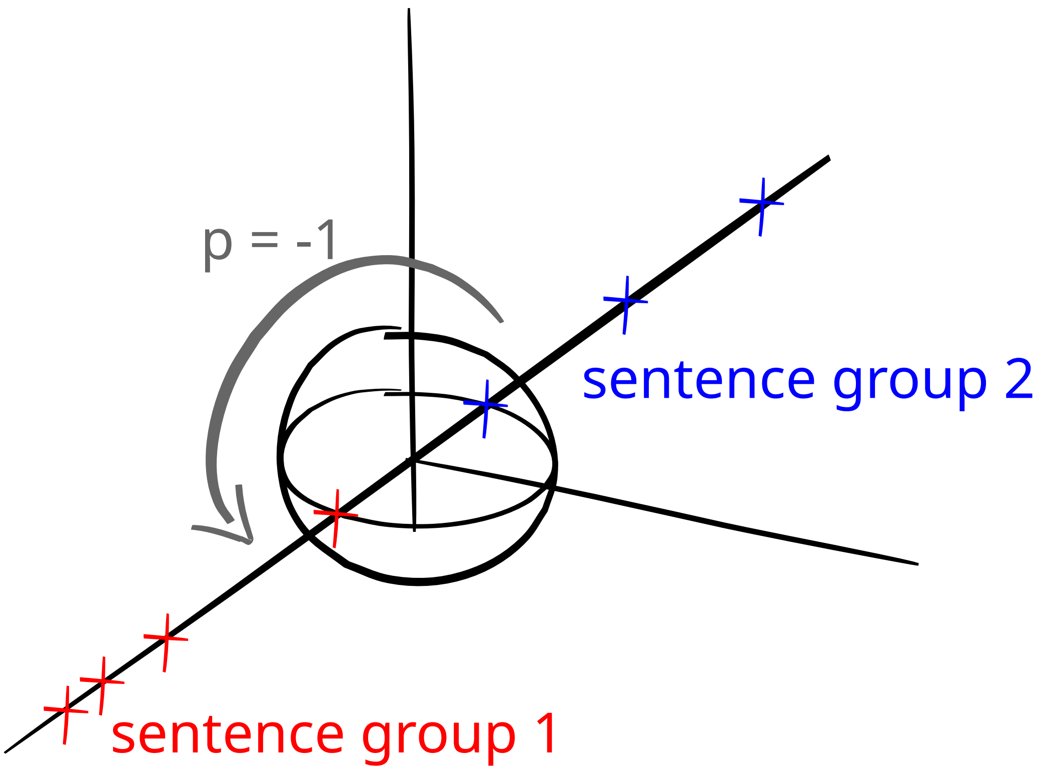


Figure A.9: Visualization of the problem with STS = 0 for variant b). The only point on the unit hypersphere with a dot product equal to -1 with respect to another point is its polar opposite. This results in the creation of two groups of sentence embeddings on the opposite line ends which – according to the model – would group equivalent sentences and claim that all sentences in the opposing group are unrelated to any given sentence. This clearly fails in the case where three or more unrelated sentences exist.

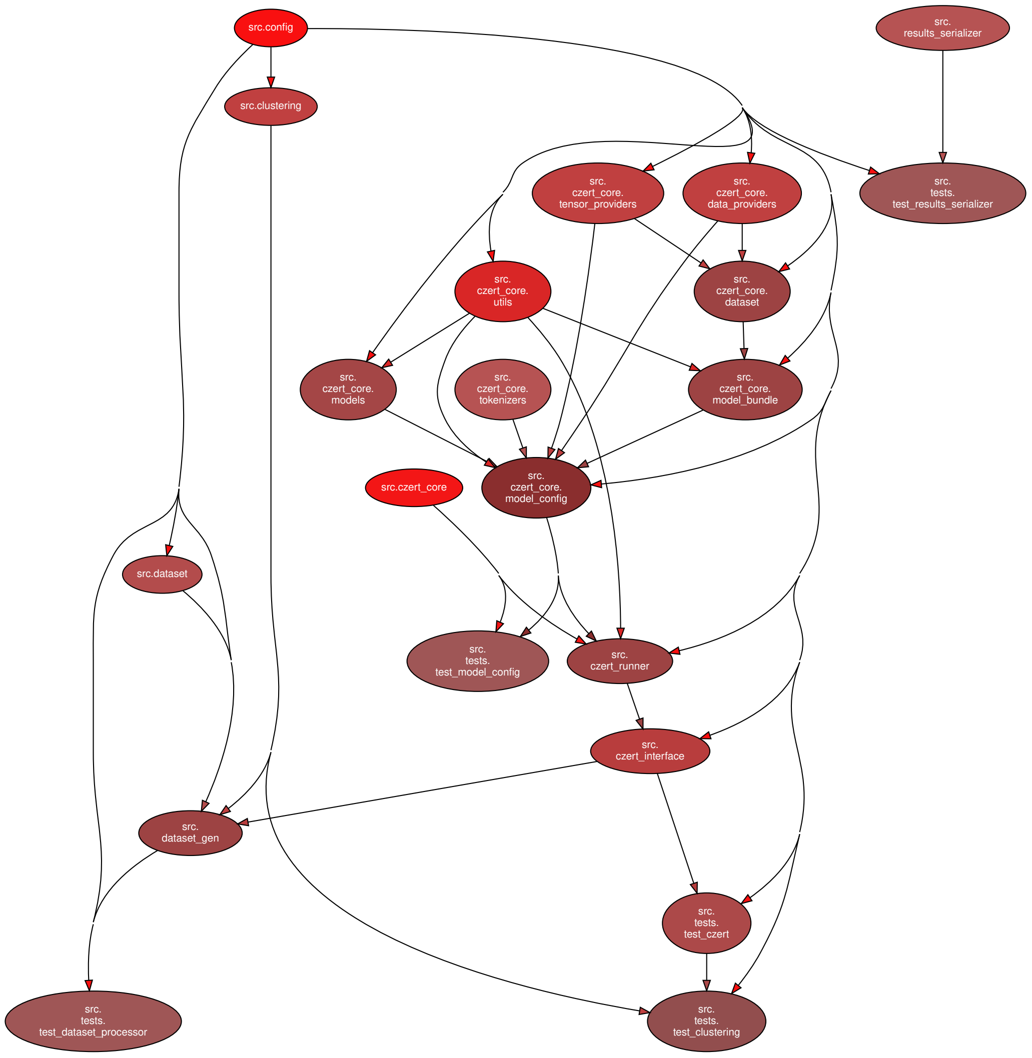

src.config
src. results_serializer
src.clustering
src. czert_core. tensor_providers
src. czert_core. data_providers
src. tests. test_results_serializer
src. czert_core. utils
src. czert_core. dataset
src. czert_core. models
src. czert_core. tokenizers
src. czert_core. model_bundle
src.czert_core
src. czert_core. model_config
src.dataset
src. tests. test_model_config
src. czert_runner
src. czert_interface
src. dataset_gen
src. tests. test_czert
src. tests. test_dataset_processor
src. tests. test_clustering

Figure A.10: A module dependency diagram of our system without external dependencies. Notice how the *src.czert_interface* is isolated from *src.czert_core* modules by *src.czert_runner*. This is the direct result of our design choices described in Section 3.6.1.

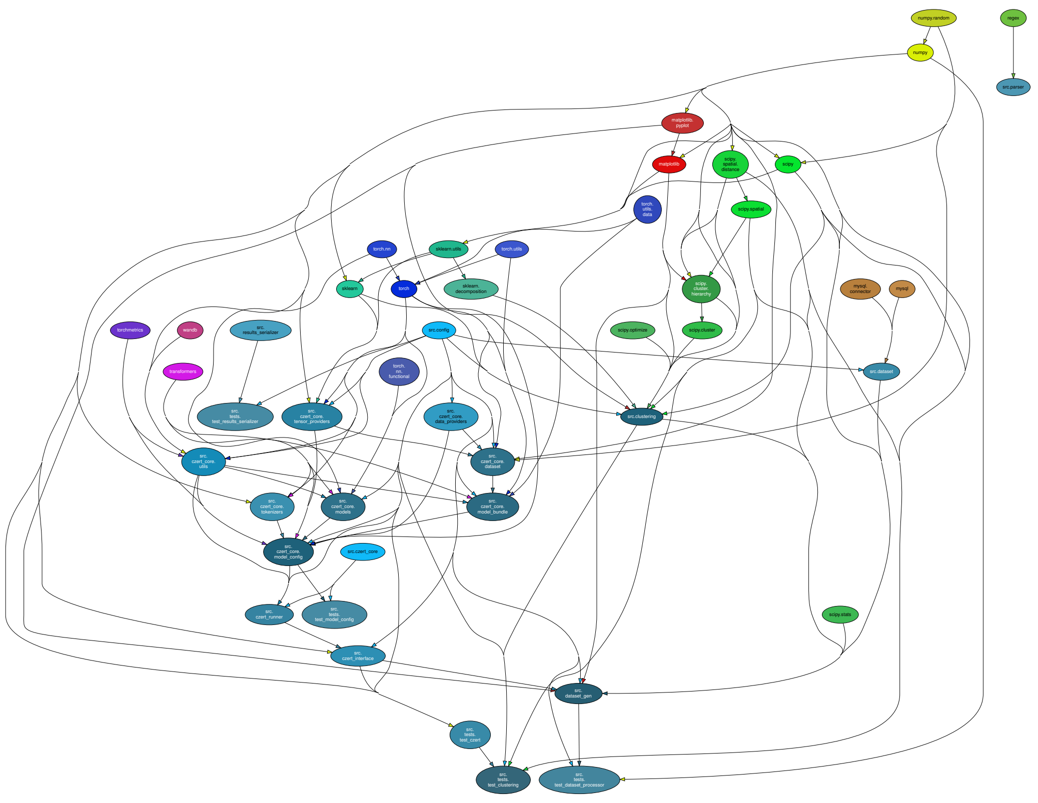


Figure A.11: A full module dependency diagram of our system, including external dependencies.