Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Sparse Sentence Embeddings

Master's thesis

2026
Bc. Tomáš Varga

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Sparse Sentence Embeddings

Master's thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Computer Science
Supervisor:	Mgr. Vladimír Boža, PhD.

Bratislava, 2026
Bc. Tomáš Varga

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Tomáš Varga
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Sparse Sentence Embeddings
Riedke vetné vnorené vektory

Anotácia: Husté vetné vnorené vektory, ako napríklad tie z modelu Sentence-BERT, sú základným pilierom moderného spracovania prirodzeného jazyka (NLP) a poháňajú úlohy od sémantického vyhľadávania až po zhlukovanie. Nedávna práca [1] však naznačuje, že tieto husté vektory narážajú na limity reprezentačnej kapacity, čo obmedzuje ich schopnosť rozlišovať medzi rastúcim počtom sémantických konceptov.

Riedke vnorené vektory (napr. SPLADE) ponúkajú silnú alternatívu, ktorá je dobre kompatibilná s tradičnými vyhľadávacími indexmi. Tieto metódy však typicky vytvárajú riedke vektory využitím predikcií maskovaného jazykového modelu (MLM) na posúdenie dôležitosti tokenov v slovníku, čím viažu riedke vnorené vektory na pôvodné tokeny slovníka.

Hlavným cieľom tejto práce je navrhnúť a vyhodnotiť vrstvu pre riedky pooling (sparse pooling layer) pre modely typu BERT, ktorá sa priamo naučí konštruovať riedky vetný vnorený vektor z vnorených vektorov jednotlivých tokenov bez použitia predikčnej hlavy MLM.

[1] Weller, Orion, et al. "On the theoretical limitations of embedding-based retrieval." arXiv preprint arXiv:2508.21038 (2025).

Vedúci: Mgr. Vladimír Boža, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 01.12.2025

Dátum schválenia: 02.01.2026 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Tomáš Varga
Study programme:	Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Sparse Sentence Embeddings

Annotation: Dense sentence embeddings, such as those from Sentence-BERT, are a cornerstone of modern NLP, powering tasks from semantic search to clustering. However, recent work [1] suggests these dense vectors face a representational capacity bottleneck, limiting their ability to distinguish between a growing number of semantic concepts.
Sparse embeddings (e.g., SPLADE) offer a powerful alternative, aligning well with traditional search indices. However, these methods typically generate sparse vectors by repurposing Masked Language Model (MLM) predictions to assess vocabulary token importance, thereby tying sparse embeddings to the original vocabulary tokens.
The main goal of the thesis is to design and evaluate a sparse pooling layer for BERT-like models that directly learns to construct a sparse sentence embedding from individual token embeddings without using the MLM prediction head.
[1] Weller, Orion, et al. "On the theoretical limitations of embedding-based retrieval." arXiv preprint arXiv:2508.21038 (2025).

Supervisor:	Mgr. Vladimír Boža, PhD.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	doc. RNDr. Tatiana Jajcayová, PhD.
Assigned:	01.12.2025
Approved:	02.01.2026 prof. RNDr. Rastislav Kráľovič, PhD. Guarantor of Study Programme

.....
Student

.....
Supervisor

Declaration: I hereby declare that I have independently written this entire bachelor thesis entitled "Sparse Sentence Embeddings", including all its figures and appendices, using the literature listed in the attached bibliography.

In preparation of the thesis I have also used the following artifical infelligence tools: [TOOLS] for the follwing purposes: [PURPOSES]. I have used artificial intelligence tools in accordance with relevant legal regulations, academic rights and freedoms, ethical and moral principles while simultaneously maintaining academic integrity. I am aware that I am fully responsible for the accuracy of the final text.

Acknowledgments: Tu môžete poďakovať školiteľovi, prípadne ďalším osobám, ktoré vám s prácou nejako pomohli, poradili, poskytli dáta a podobne.

Abstrakt

Slovenský abstrakt v rozsahu 100-500 slov, jeden odstavec. Abstrakt stručne sumarizuje výsledky práce. Mal by byť pochopiteľný pre bežného informatika. Nemal by teda využívať skratky, termíny alebo označenie zavedené v práci, okrem tých, ktoré sú všeobecne známe.

Kľúčové slová: jedno, druhé, tretie (prípadne štvrté, piate)

Abstract

Abstract in the English language (translation of the abstract in the Slovak language).

Keywords:

Contents

List of Figures

List of Tables

Introduction

Sentence embeddings have become an essential component of modern Natural Language Processing and represent a significant milestone in the field of text understanding. Their ability to map sentences into fixed dimensional vector spaces, where semantic similarity corresponds to geometrical proximity, has opened new possibilities in semantic search, information retrieval, text clustering, and question answering systems. Models like Sentence-BERT [8], built upon transformer architectures, have demonstrated that neural encoders can produce high-quality dense vector representations that capture semantic meaning effectively. Their widespread deployment across various domains, from customer support chatbots to academic search engines, highlights their growing importance and potential. Despite their impressive capabilities, however, dense sentence embeddings face several challenges, one of the most pressing being the representational capacity bottleneck.

The representational capacity bottleneck refers to a fundamental limitation of dense vector representations, as the number of distinct semantic concepts that need to be distinguished grows, the ability of fixed-dimensional dense vectors to reliably separate them declines. This phenomenon has been recently formalized in theoretical work by Weller et al. [13], who demonstrate that embedding based retrieval systems face inherent limitations when scaling to large document collections. The issue becomes particularly relevant in large-scale retrieval scenarios where millions of documents must be differentiated based on subtle semantic distinctions. In such settings, dense embeddings may struggle to maintain sufficient selective power, leading to degraded retrieval performance and reduced precision different applications.

Given the described limitations and the increasing scale of modern retrieval systems, exploring alternative representation strategies is a crucial task for improving the reliability and scalability of semantic search. Sparse embeddings offer a compelling alternative to dense representations. Unlike dense vectors where all dimensions carry information, sparse embeddings concentrate meaning in a small number of active dimensions, with most values being exactly zero. Methods like SPLADE [2] have demonstrated that learned sparse representations can achieve competitive or even superior performance on retrieval benchmarks while maintaining computational efficiency. However, current approaches to learning sparse sentence embeddings typically rely on

repurposing the Masked Language Model prediction head, which ties the sparse representation to the original tokenizer vocabulary.

The goal of this thesis is to design and evaluate a sparse pooling layer for BERT-like models that directly learns to construct a sparse sentence embedding from individual token embeddings, without using the Masked Language Model prediction head. Specifically, we hypothesize that a learnable sparse pooling mechanism can produce sentence embeddings that are both genuinely sparse and semantically meaningful, while being independent of vocabulary based projections. This sparsity should emerge through learned dimension selection at the embedding level rather than through token importance evaluation over the vocabulary.

FINISH UP HERE BASED ON THE WHOLE THESIS

Chapter 1

From Words to Sentences

Before we can represent entire sentences as vectors, we must first understand how individual pieces of text are processed by neural language models. This chapter introduces the fundamental concepts of tokenization and embeddings, which form the foundation for all modern sentence representation methods. We begin with tokenization, the process of breaking text into discrete units, then examine how these units are transformed into continuous vector representations. Finally, we trace the evolution from static word embeddings to contextual embeddings and discuss the challenge of aggregating token representations into sentence-level embeddings.

1.1 Tokenization

Neural language models do not process raw text directly. Instead, they operate on discrete units called *tokens* [12, 11]. A token may represent a complete word, a subword fragment, a punctuation mark, or a special character such as a newline. The process of splitting input text into such units is called tokenization, and it represents the first and arguably one of the most critical steps in any text processing pipeline [6].

When users interact with large language models through web interfaces, they observe that responses are generated incrementally rather than all at once. This incremental generation reflects the fundamental nature of these models: they predict and produce one token at a time, with each new token conditioned on all previous tokens [12, 11]. Understanding tokenization is therefore essential for understanding how these models process and generate language.

1.1.1 From Text to Tokens

The way we divide text into tokens significantly impacts model performance, vocabulary size, and the ability to handle different languages and domains [10, 4]. Several approaches exist, each with distinct trade-offs.

Word-level tokenization. The simplest approach to tokenization is splitting text by whitespace [10]. This method treats each space-separated unit as a token, which aligns with our intuitive understanding of words. However, this approach has significant drawbacks. Punctuation marks become attached to adjacent words, so "hello!" and "hello" would be treated as completely different tokens. More problematically, the resulting vocabulary can grow extremely large, as every unique word form requires its own entry [10, 7, 4].

A large vocabulary creates several problems. First, it increases memory requirements, as each token needs its own embedding vector. Second, rare words may not have enough training examples to learn good representations. Third, the model cannot handle words it has never seen during training, a problem known as the out-of-vocabulary (OOV) problem [10, 7].

An improvement is to split on both whitespace and punctuation, treating punctuation marks as separate tokens [10]. This reduces vocabulary size while preserving meaningful distinctions between words and their surrounding punctuation. Further refinements include rule-based tokenization, which handles language-specific phenomena such as contractions [10, 4, 7]. In English, for example, treating the contraction suffix "n't" as a separate token allows words like *don't*, *won't*, and *can't* to share common components rather than requiring entirely separate vocabulary entries for each contracted form.

Character-level tokenization. At the opposite extreme, character-level tokenization reduces the vocabulary to just the alphabet and special characters [10, 4]. For English, this means a vocabulary of roughly 26 letters plus digits, punctuation, and special symbols. This approach completely eliminates the out-of-vocabulary problem, as any text can be represented as a sequence of known characters.

However, individual characters carry little semantic meaning on their own [10, 7]. The word "king" as a sequence of four character tokens ("k", "i", "n", "g") loses the rich semantic associations that the complete word carries. Models must learn to compose characters into meaningful units, which requires processing much longer sequences and makes training more difficult. Character-level models can be useful for specific applications such as spelling correction, but they generally underperform word-level or subword-level approaches for tasks requiring semantic understanding [10, 7].

1.1.2 Subword Tokenization

Modern language models typically use subword tokenization, which strikes a balance between word-level and character-level approaches [10, 4]. The key insight is that while we want to preserve whole words when possible, we can decompose rare or complex

words into smaller meaningful units. This keeps the vocabulary manageable while maintaining semantic information and handling previously unseen words.

For example, the word "woodcutter" might be split into "wood" and "cutter", preserving the semantic components (something related to wood, something that cuts) while avoiding the need for a separate vocabulary entry for this compound word. Similarly, "unhappiness" might become "un", "happi", and "ness", capturing the negation prefix, the root, and the noun-forming suffix. The target vocabulary size is typically around 30,000 tokens for base models, though this can vary depending on the application and the diversity of the training data [10, 4].

Byte-Pair Encoding. The most widely used subword algorithm is Byte-Pair Encoding (BPE) [9, 4, 10, 3]. Originally developed as a data compression algorithm, BPE was adapted for neural machine translation and has since become the standard tokenization approach for transformer models.

BPE works through an iterative merging process. Starting from individual characters, the algorithm counts all adjacent pairs of tokens in the training corpus and merges the most frequent pair into a new token. This process repeats until reaching a target vocabulary size. For example, if "t" and "h" frequently appear together, they would be merged into "th". Later iterations might merge "th" and "e" into "the" if this trigram is common enough.

The result is a vocabulary that includes common words as single tokens, while rare words are decomposed into frequent subword units. Importantly, the merge operations are learned from data, so the tokenization automatically adapts to the statistical properties of the training corpus [4, 10, 3].

Other subword algorithms. Several variants of subword tokenization exist. WordPiece [14], used by BERT, is similar to BPE but selects merges based on likelihood improvement rather than raw frequency. SentencePiece [5] treats the input as a raw stream of Unicode characters, avoiding the need for language-specific pre-tokenization rules. Unigram language modeling [**?**] takes a different approach, starting with a large vocabulary and iteratively removing tokens that least affect the likelihood of the training data.

1.1.3 Implications of Tokenization Choices

The choice of tokenization strategy affects model performance in subtle but important ways [1]. Poor tokenization can cause difficulties that might mistakenly be attributed to the model architecture rather than the preprocessing step.

One well-known issue is that tokenization can make simple arithmetic surprisingly

difficult for language models [1]. Numbers may be split inconsistently ("123" might become "12" and "3" in one context but "1" and "23" in another), making it hard for the model to learn numerical relationships. Similarly, spelling tasks become challenging when words are split into subword units that do not align with individual letters.

Non-English languages often suffer from less efficient tokenization [6]. Because most tokenizers are trained primarily on English text, they develop subword units optimized for English morphology. Other languages, especially those with different writing systems or richer morphology, may require more tokens to represent the same content, effectively reducing the model's capacity for those languages.

Chapter 2

Sparse embeddings

Chapter 3

Experiments

Before developing innovative sparse pooling mechanisms, it is essential to establish a solid experimental foundation. This chapter describes the implementation of a benchmarking framework that allows systematic comparison of different pooling strategies for sentence embeddings. We begin with an overview of the experimental setup, then describe the architecture of each pooling layer, and finally present the training and evaluation pipeline. This framework serves as the basis for all experiments.

3.1 Experimental Setup

3.1.1 Base Encoder

We use `bert-base-uncased` as the base encoder for all experiments. This model consists of 12 transformer layers and produces 768-dimensional token embeddings. By keeping the encoder constant across all experiments, we ensure that any differences in performance can be attributed entirely to the pooling strategy. The only component that varies between experiments is the pooling layer.

3.1.2 Training Data

For training, we use the MNLI (Multi-Genre Natural Language Inference) dataset. Specifically, we extract sentence pairs labelled as *entailment*, that is, cases where one sentence logically follows from another. These pairs are semantically related and provide a good training signal for learning sentence similarity. We use up to 50,000 such pairs for training.

We employ Multiple Negatives Ranking Loss for training. The idea is straightforward, for each pair of related sentences, the model should produce embeddings that are more similar to each other than to embeddings of unrelated sentences in the same batch. This contrastive objective encourages the model to learn meaningful sentence

representations.

3.1.3 Evaluation

We evaluate all methods on the STS-B (Semantic Textual Similarity Benchmark) validation set, which contains 1,500 sentence pairs with human-annotated similarity scores ranging from 0, which represents complete difference, to 5 what represents essential equivalence.

Our primary metric is Spearman correlation, which measures how well the ranking of pairs by the model matches the human ranking. We also report Pearson correlation, which measures the linear correlation between model scores and human scores. For sparse methods, we additionally compute sparsity statistics, including the percentage of dimensions that are zero and the average number of active dimensions per embedding.

3.2 Pooling Strategies

We compare three categories of pooling strategies. Baseline methods without learnable parameters, learnable pooling layers, and sparse pooling layers.

3.2.1 Baseline Pooling

These strategies have no learnable parameters and serve as reference points.

Mean pooling computes the average of all token embeddings. This is often the default choice and works surprisingly well in practice. **CLS pooling** uses the embedding of the special `[CLS]` token as the sentence representation. While this token was designed for classification tasks, it is sometimes used for sentence-level representations. **Max pooling** takes the element-wise maximum across all token embeddings, which can capture the most prominent features.

3.2.2 Learnable Pooling

These strategies have parameters that are trained alongside or instead of the encoder.

Attention Pooling. This layer learns to assign different weights to different tokens, recognizing that some words are more important for the sentence meaning than others. It consists of a single linear layer that computes an importance score for each token. These scores are normalized using softmax, and the final embedding is a weighted average of token embeddings according to these learned weights.

Formally, given token embeddings $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_n]$ where $\mathbf{h}_i \in \mathbb{R}^d$, attention pooling computes:

$$\alpha_i = \mathbf{w}^\top \mathbf{h}_i \tag{3.1}$$

$$a_i = \frac{\exp(\alpha_i)}{\sum_{j=1}^{n} \exp(\alpha_j)} \tag{3.2}$$

$$\mathbf{s} = \sum_{i=1}^{n} a_i \mathbf{h}_i \tag{3.3}$$

where $\mathbf{w} \in \mathbb{R}^d$ is a learnable weight vector and $\mathbf{s}$ is the resulting sentence embedding.

Weighted Pooling. Instead of weighting tokens, this approach learns weights for each dimension of the embedding. It first computes the standard mean of token embeddings and then multiplies each dimension by a learned weight. The hypothesis is that some dimensions may be more useful than others for capturing sentence-level semantics.

Hierarchical Pooling. This more complex approach uses two levels of attention. First, a multi-head self-attention layer allows tokens to exchange information with each other, producing enhanced token representations. Then, a global attention mechanism combines these enhanced representations into a single sentence embedding. This architecture can capture more complex interactions between tokens before aggregating them.

3.2.3 Sparse Pooling

These strategies produce embeddings where most dimensions are exactly zero.

Top-K Sparse Pooling. After computing a dense embedding through mean pooling and a learned projection, this layer retains only the K dimensions with the largest absolute values. All other dimensions are set to zero. We experiment with $K \in \{50, 100, 200\}$ out of the total 768 dimensions.

The forward pass proceeds as follows. First, we compute the mean of token embeddings to obtain a dense representation. This representation is then passed through a learned linear projection, which we call the activation head. Next, we identify the K dimensions with largest absolute values and zero out all other dimensions. This approach is simple but effective, forcing the model to concentrate information in a small subset of dimensions.

3.3 Implementation Details

3.3.1 Project Structure

The implementation is organized into several Python modules within a `src` directory. The `learnable_pooling.py` file contains all pooling layer definitions, while `modeling.py` handles model assembly. The `log_results.py` module defines the result data structure and handles appending results to CSV files, while `eval_stsb.py` implements the evaluation script. Training is handled by `train_sparse_pooling.py`. A shell script `run_full_experiment.sh` runs all experiments. Results are stored in a `results` directory containing `results.csv` for the main results table and `sparsity_stats.json` for sparsity statistics. Trained models are saved in an `outputs` directory.

3.3.2 Pooling Layer Implementation

All learnable pooling layers are implemented as PyTorch modules in `learnable_pooling.py`. Each layer takes token embeddings and an attention mask as input and returns a sentence embedding.

Code seen in 3.1 shows the implementation of attention pooling. The module contains a single linear layer that projects each token embedding to a scalar attention score. During the forward pass, these scores are computed for all tokens, masked to ignore padding tokens, normalized with softmax to obtain attention weights, and finally used to compute a weighted sum of token embeddings.

Algorithm 3.1: Attention pooling implementation

```
class AttentionPooling(nn.Module):
    def __init__(self, hidden_dim):
        super().__init__()
        self.attention = nn.Linear(hidden_dim, 1)

    def forward(self, token_embeddings, attention_mask):
        scores = self.attention(token_embeddings).squeeze(-1)
        scores = scores.masked_fill(
            attention_mask == 0, float('-inf')
        )
        weights = F.softmax(scores, dim=1)
        sentence_embedding = torch.bmm(
            weights.unsqueeze(1), token_embeddings
        ).squeeze(1)
        return sentence_embedding
```

The sparse pooling implementation, shown in 3.2, includes a learned projection layer and top-K selection. The module first computes mean pooling over the token embeddings, accounting for the attention mask to ignore padding. The result is passed through a linear projection layer. Finally, only the K dimensions with the largest absolute values are retained, while all other dimensions are set to zero using scatter operations.

Algorithm 3.2: Sparse top-K pooling implementation

```
class SparsePooling(nn.Module):
    def __init__(self, hidden_dim, k=None):
        super().__init__()
        self.k = k if k is not None else hidden_dim // 10
        self.activation_head = nn.Linear(hidden_dim, hidden_dim)

    def forward(self, token_embeddings, attention_mask):
        mask_expanded = attention_mask.unsqueeze(-1).float()
        sum_emb = torch.sum(
            token_embeddings * mask_expanded, dim=1
        )
        sum_mask = torch.clamp(
            mask_expanded.sum(dim=1), min=1e-9
        )
        dense_embedding = sum_emb / sum_mask
        activation_scores = self.activation_head(dense_embedding)
        topk_vals, topk_idx = torch.topk(
            torch.abs(activation_scores), k=self.k, dim=1
        )
        sparse_embedding = torch.zeros_like(activation_scores)
        sparse_embedding.scatter_(
            1, topk_idx,
            activation_scores.gather(1, topk_idx)
        )
        return sparse_embedding
```

3.3.3 Model Assembly

The `modeling.py` file connects the BERT encoder with pooling layers to create a complete SentenceTransformer model. The main function `build_sbert_from_hf` accepts a model name and pooling type, and returns a ready-to-use model.

For standard pooling strategies such as mean, CLS, and max pooling, we use the built-in implementation from the `sentence-transformers` library. For custom pooling

layers, we create a `CustomPoolingModule` wrapper that ensures compatibility with the SentenceTransformer API and handles saving and loading of model weights.

3.3.4 Training Pipeline

The training script `train_sparse_pooling.py` implements the complete training procedure. The process begins by loading the BERT model and attaching the selected pooling layer. Next, entailment pairs are extracted from the MNLI dataset. The model is then trained using Multiple Negatives Ranking Loss, with periodic evaluation on the STS-B validation set. Finally, the trained model is saved to disk.

The script supports two training modes. In the frozen encoder mode, only the pooling layer is trained while BERT weights remain fixed. This is faster and tests whether the pooling layer alone can improve results. In full training mode, both the pooling layer and BERT encoder are trained together, allowing the encoder to adapt to the new pooling strategy.

Training hyperparameters used in our experiments are as follows. We use a batch size of 64 and a learning rate of 2×10^{-5}. The training set consists of 50,000 pairs from MNLI, and we train for one epoch with a warmup period covering 10% of training steps.

3.3.5 Evaluation Pipeline

The evaluation script `eval_stsb.py` measures embedding quality on the STS-B benchmark. The process starts by loading the model, either from Hugging Face or a trained checkpoint. All sentences from STS-B are then encoded, and cosine similarity is computed between each sentence pair. Spearman and Pearson correlations with human judgments are calculated, and for sparse methods, sparsity statistics are also computed. All results are logged to a CSV file.

Sparsity statistics include the sparsity ratio, which represents the percentage of zero dimensions, the average number of non-zero dimensions, and the Gini coefficient of activation magnitudes.

3.4 Summary

This chapter presented the implementation of a benchmarking framework for comparing pooling strategies in sentence embeddings. We described the experimental setup, including the base encoder, training data, and evaluation metrics. We then detailed the architecture of baseline, learnable, and sparse pooling layers, along with the complete training and evaluation pipeline.

This framework establishes a reproducible baseline against which we can measure the impact of future improvements. In the following chapters, we will use this infrastructure to evaluate innovative sparse pooling mechanisms and analyse the trade-offs between sparsity and embedding quality. Any new method we develop will be compared directly against the baselines established here.

Conclusion

See file `zaver.tex` for information about recommended contents of the Conclusion chapter.

Bibliography

[1] ChristopherGS. The technical user's introduction to llm tokenization, 2024. [Citované 2025-11-4] Dostupné z `https://christophergs.com/blog/understanding-llm-tokenization`.

[2] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. Splade: Sparse lexical and expansion model for first stage ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2288–2292, 2021.

[3] Tayyib Ul Hassan Gondal. All you need to know about tokenization in llms, 2024. [Citované 2025-13-4] Dostupné z `https://medium.com/thedeephub/all-you-need-to-know-about-tokenization-in-llms-7a801302cf54`.

[4] Gwyneth Iredale. Tokenization algorithms in nlp, 2021. [Citované 2025-11-4] Dostupné z `https://101blockchains.com/tokenization-nlp/`.

[5] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing, 2018.

[6] Dmitrii Lukianov. A closer look at large language models, 2024. [Citované 2025-11-4] Dostupné z `https://akvelon.com/a-closer-look-at-large-language-models/`.

[7] Sujatha Mudadla. What are the types of tokenizer algorithm s in nlp?, 2023. [Citované 2025-11-4] Dostupné z `https://medium.com/@sujathamudadla1213/what-are-the-types-of-tokenizer-algorithm-s-in-nlp-6f61da33cb4d`.

[8] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[9] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 1715–1725, 2016.

[10] Sharvil. Tokenization algorithms explained, 2021. [Citované 2025-11-4] Dostupné z `https://medium.com/data-science/tokenization-algorithms-explained-e25d5f4322ac`.

[11] Sean Trott. Tokenization in large language models, explained, 2024. [Citované 2025-11-4] Dostupné z `https://seantrott.substack.com/p/tokenization-in-large-language-models`.

[12] vinay kumar. Llm building blocks -tokens and embeddings, 2024. [Citované 2025-11-4] Dostupné z `https://vinaykuma201.medium.com/llm-building-blocks-tokens-and-embeddings-part-2-a18a531b4bdc`.

[13] Orion Weller, Michael Boratko, Iftekhar Naim, and Jinhyuk Lee. On the theoretical limitations of embedding-based retrieval, 2025.

[14] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

Appendix A: Contents of the Online Appendix

The online appendix attached to this work contains the source code data files used in the analysis. The source code is available also at `http://mojadresa.com/`.

Ak uznáte za vhodné, môžete tu aj podrobnejšie rozpísať obsah elektronickej prílohy, prípadne poskytnúť návod na inštaláciu programu. Alternatívou je tieto informácie zahrnúť do samotnej prílohy, alebo ich uviesť na obidvoch miestach.

Appendix B: User Manual

V tejto prílohe uvádzame používateľskú príručku k nášmu softvéru. Tu by ďalej pokračoval text príručky. V práci nie je potrebné uvádzať používateľskú príručku, pokiaľ je používanie softvéru intuitívne alebo ak výsledkom práce nie je ucelený softvér určený pre používateľov.

V prílohách môžete uviesť aj ďalšie materiály, ktoré by mohli pôsobiť rušivo v hlavnom texte, ako napríklad rozsiahle tabuľky a podobne. Materiály, ktoré sú príliš dlhé na ich tlač, odovzdajte len v elektronickej prílohe.