FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Bakalářská práce

Tvorba jazykového modelu pro vektorizaci textu

Studijní program: B0613A140005 – Informační technologie
Studijní obor: Inteligentní systémy
Autor práce: **Denis Tauchman**
Vedoucí práce: Ing. Martin Poláček

Liberec 2025

FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Zadání bakalářské práce

Tvorba jazykového modelu pro vektorizaci textu

Jméno a příjmení:	**Denis Tauchman**
Osobní číslo:	M22000192
Studijní program:	B0613A140005 Informační technologie
Specializace:	Inteligentní systémy
Zadávající katedra:	Ústav informačních technologií a elektroniky
Akademický rok:	2024/2025

Zásady pro vypracování:

1. Seznamte se s problematikou vektorizace textu pomocí velkých jazykových modelů a proveďte rešerši existujících metod a modelů.
2. Na základě provedené rešerše vyhodnoťte vybrané modely pro vektorizaci textu na vhodné vývojové sadě.
3. Vytvořte vlastní jazykový model optimalizovaný na úlohu vektorizace textu.
4. Výsledný model s nejlepším nastavením parametrů porovnejte s výsledky existujících modelů.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30-40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	čeština

Seznam odborné literatury:

[1] Vaswani, Ashish et al. "Attention is All you Need." *Neural Information Processing Systems* (2017).

[2] Devlin, Jacob et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." *North American Chapter of the Association for Computational Linguistics* (2019).

[3] Bednár, Jirí et al. "Some Like It Small: Czech Semantic Embedding Models for Industry Applications." *AAAI Conference on Artificial Intelligence* (2023).

Vedoucí práce:	Ing. Martin Poláček Ústav informačních technologií a elektroniky
Datum zadání práce:	14. října 2024
Předpokládaný termín odevzdání:	9. května 2025

L.S.

doc. Ing. Josef Černohorský, Ph.D.
děkan

doc. Ing. Josef Chaloupka, Ph.D.
garant studijního programu

V Liberci dne 14. října 2024

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.
Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.
Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.
Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.
Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS STAG se shoduje s textem tištěné podoby práce.
Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.
Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

3. 5. 2025 Denis Tauchman

Tvorba jazykového modelu pro vektorizaci textu

Abstrakt

Tato bakalářská práce se zabývá tvorbou jazykového modelu pro český jazyk, určeného k vektorizaci textu v rámci metody Retrieval-Augmented Generation (RAG). Cílem práce bylo navrhnout a natrénovat model, který umožní efektivní převod vstupních textových dotazů a dokumentů do vektorového prostoru, a tím zlepšit proces vyhledávání informací. Navržený model vychází z předtrénované architektury XLM-RoBERTa-base typu transformer, která byla dále doladěna (fine-tuned) na českých datech, včetně vlastního datasetu vytvořeného pro účely této práce. Experimentální část se zaměřuje na výběr základního modelu, úpravu hyperparametrů a přípravu trénovacích dat. Dosažené výsledky byly porovnány s alternativními přístupy běžně používanými pro podobné úlohy, přičemž navržený model dosáhl lepší přesnosti. V závěrečné části práce je model integrován do webové aplikace pro vektorizaci dokumentů v rámci techniky RAG, čímž je ověřena jeho praktická použitelnost.

Klíčová slova: Velké jazykové modely, RAG, vektorizace textu, RoBERTa, transformery

Abstract

This bachelor thesis deals with the creation of a language model for Czech, which is designed for text vectorization within the Retrieval--Augmented Generation (RAG) method. The aim of the thesis was to design and train a model that will enable efficient conversion of input text queries and documents into vector space, thus improving the information retrieval process. The proposed model is based on the pre-trained XLM-RoBERTa-base transformation architecture, which was further tuned on Czech data, including a custom dataset created for the purpose of this work. The experimental part focuses on the selection of the base model, the adjustment of the hyperparameters and the preparation of the training data. The results obtained were compared with alternative approaches commonly used for similar tasks, with the proposed model achieving higher accuracy. In the final part of the work, the model is integrated into a web application for document vectorization within the RAG technique, thus verifying its practical applicability.

Keywords: Large language models, RAG, text vectorization, RoBERTa, transfomers

Poděkování

Rád bych poděkoval panu inženýru Martinu Poláčkovi za odborné vedení, cenné rady, veškeré konzultace a podporu při tvorbě této bakalářské práce.

Obsah

Seznam zkratek . . . 12

1 Úvod **13**

2 Práce zabývající se vektorizací **14**
2.1 Word2Vec . . . 14
2.2 Transformery a jazykové modely . . . 15
2.3 České jazykové modely . . . 15
2.4 Jazykové modely E5 . . . 16
2.5 Cíle práce . . . 16

3 Vektorová podobnost a prohledávání dokumentů **17**
3.1 Kosinová podobnost . . . 17
3.2 RAG . . . 17
3.2.1 Princip RAG . . . 18
3.2.2 Předzpracování dokumentů . . . 18
3.2.3 Vyhledání relevantního kontextu (Retrieval) . . . 18
3.2.4 Rozšíření dotazu (Augmentation) . . . 18
3.2.5 Generování odpovědi (Generation) . . . 18

4 Současné architektury neuronových sítí pro jazykové modely **20**
4.1 Rekurentní neuronové sítě . . . 20
4.2 Long short-term memory (LSTM) . . . 21
4.3 Transformery . . . 22
4.3.1 Tokenizer . . . 23
4.3.2 Poziční enkódování . . . 23
4.3.3 Attention mechanismus . . . 24
4.3.4 Enkodér . . . 26
4.3.5 BERT . . . 28
4.3.6 RoBERTa . . . 29

5 Navržené řešení **31**
5.1 Použité metriky . . . 31
5.2 Základní architektura . . . 32
5.3 Využité datasety . . . 32
5.4 Doladění modelu . . . 33
5.5 Výsledky . . . 34

6 Experimentální vyhodnocení **36**
6.1 Výběr základního modelu 37
6.2 Výběr hyperparametrů 39
6.2.1 Learning rate 39
6.2.2 Batch size 40
6.3 Datasety 41
6.3.1 SQuAD 42
6.3.2 DaReCzech 43
6.3.3 iDNES 44

7 Tvorba aplikace **47**
7.1 Předzpracování dokumentů 47
7.2 Vektorizace a ukládání do databáze 47
7.3 Vyhledávání a generování odpovědí 48
7.4 Uživatelské rozhraní 48

Závěr **50**

Použitá literatura **51**

Seznam tabulek

5.1 Použité datasety 32
5.2 Porovnání s ostatními modely 35

6.1 Výsledky modelů na datasetu SQuAD dev2.0 38
6.2 Výsledná přesnost modelu při různých hodnotách learning rate 40
6.3 Výsledná přesnost modelu při různých hodnotách batch size 41

Seznam obrázků

3.1 Průběh RAG 19

4.1 Porovnání neuronové sítě a rekurentní neuronové sítě 21
4.2 Průběh attention 25
4.3 Převedení vstupních tokenů na vektory 26
4.4 Přidání informace o pozici 26
4.5 Enkodér 27

5.1 Použití metrik acc@1, acc@3, acc@10 32

6.1 Transformace datasetu SQuAD 43
6.2 Doladění pomocí různých kombinací datasetů - acc@10 45

7.1 Uživatelské rozhraní - položení dotazu 48
7.2 Uživatelské rozhraní - odpověď LLM 49

Seznam zkratek

API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
BiLSTM	Bidirectional Long Short-Term Memory
BOW	Bag of Words
BS	Batch Size
BPE	Byte Pair Encoding
CBOW	Continuous Bag of Words
CPU	Central Processing Unit
ELMO	Embeddings from Language Models
FFNN	Feed-Forward Neural Network
GPU	Graphics Processing Unit
GPT	Generative Pre-trained Transformer
HTML	HyperText Markup Language
LaBSE	Language-agnostic BERT Sentence Embedding
LLM	Large Language Model
LR	Learning Rate
LSTM	Long Short-Term Memory
ML	Machine Learning
MLM	Masked Language Model
NLP	Natural Language Processing
NSP	Next Sentence Prediction
QA	Question Answering
RAG	Retrieval-Augmented Generation
RAM	Random Access Memory
RNN	Recurrent Neural Network
RoBERTa	Robustly Optimized BERT Pretraining Approach
SQuAD	Stanford Question Answering Dataset
TF-IDF	Term Frequency–Inverse Document Frequency
ULM	Unigram Language Model
URL	Uniform Resource Locator
USE	Universal Sentence Encoder
W2V	Word2Vec

1 Úvod

V dnešní digitální éře jsme každý den zaplaveni obrovským množstvím textových dat - od příspěvků na sociálních sítích přes emaily, články a zprávy až po technickou dokumentaci a uživatelské manuály. Abychom dokázali z těchto dat získat smysluplné informace a efektivně je zpracovávat, je nezbytné používat pokročilé metody zpracování přirozeného jazyka (NLP). Jazykové modely, které stojí v centru těchto metod, se staly klíčovým nástrojem pro automatizaci analýzy textu, strojový překlad, generování textu nebo například automatickou sumarizaci.

Zpracování textu je však pro počítače velmi obtížné, protože na rozdíl od čísel, se kterými počítače pracují přirozeně, je jazyk pro ně nestrukturovaný a plný nejednoznačností. Slova mohou mít různé významy v závislosti na kontextu, gramatické struktury se liší napříč jazyky a běžná lidská komunikace často obsahuje slang, metafory nebo ironii. Převést text do formy, které by počítač rozuměl, tedy představuje zásadní výzvu, kterou se NLP snaží překonat.

NLP má dlouhou historii, kdy první přístupy ke zpracování textu byly založeny na pevných jazykových pravidlech definovaných lingvisty. S rozvojem strojového učení se objevily statistické metody, jako je bag-of-words (BoW) a TF-IDF, které reprezentovaly texty pomocí frekvence slov, ale nedokázaly zachytit významové vztahy mezi nimi.

Tento nedostatek vedl k vývoji distribučních reprezentací slov, založených na distribuční hypotéze, že slova s podobným významem se vyskytují v podobných kontextech. Průlom přinesl model Word2Vec (W2V) od Tomáše Mikolova (2013), který převádí slova do hustých vektorů a efektivněji zachycuje sémantické vztahy. Přestože W2V znamenal pokrok, měl omezení - nezohlednění dlouhodobého kontextu. To vedlo k vývoji pokročilejších modelů, jako jsou ELMo a BERT využívající architekturu transformer.

Tato bakalářská práce se nejdříve věnuje přehledu souvisejících prací, kde popisuje současné architektury jazykových modelů, jejich principy a trénování. Dále podrobně popisuje tvorbu vlastního jazykového modelu, včetně vybraného základního modelu, definici hyperparametrů a proces trénování. Experimentální část se zabývá analýzou a vyhodnocením současných jazykových modelů, výběrem základního modelu, který bude dotrénován na specifických datech. Popisuje proces výběru hyperparametrů, tvorbu vlastního datasetu a kombinaci s ostatními datatasety.

2 Práce zabývající se vektorizací

Vektorizace textu představuje klíčový krok ve zpracování přirozeného jazyka, neboť umožňuje převod textových dat do numerické podoby, se kterou mohou pracovat algoritmy strojového učení. V průběhu let došlo k velkému vývoji metod pro vektorizaci - od jednoduchých přístupů využívající pouze počet slov po pokročilé neuronové sítě schopné zachytit složité jazykové vztahy. Tato kapitola představuje hlavní milníky v oblasti vektorizace textu a shrnuje vybrané související práce.

Jak již bylo zmíněno, první přístupy k reprezentaci textu byly založeny na jednoduchých statistických metodách. BoW je jedním z nejstarších modelů, který text reprezentuje jako množinu slov. Tento přístup ovšem ztrácí informaci o pořadí jednotlivých slov a nezachycuje sémantické vztahy mezi slovy. Dalším zlepšení přineslo TF-IDF, které váží slova podle jejich důležitosti v rámci dokumentu a celém korpusu - eliminuje tím problém častých slov, která nenesou významovou informaci a umožňuje efektivnější prohledávání textů. I tak byl u této metody problém zachytit významové vztahy a kontextové informace.

2.1 Word2Vec

Významný průlom ve vektorizaci textu pak přinesl model W2V [1], který využívá neuronové sítě ke generování slovních vektorů, přičemž dokáže zachytit sémantické vztahy mezi slovy na základě jejich kontextu. Word2Vec využívá dvě hlavní architektury: CBOW (Continuous Bag-of-Words) a Skip-Gram [2].

- CBOW - model se snaží předpovědět cílové slovo na základě jeho okolních slov.
- Skip-Gram - model funguje opačně. Snaží se na základě daného slova předpovědět slova, která se vyskytují v jeho okolí.

Velkou výhodou Word2Vec oproti tradičním metodám, jako jsou one-hot vektory nebo počítání frekvence výskytu slov, je schopnost převést slova do spojitého vektorového prostoru. V tomto prostoru jsou podobná slova blízko sebe, což umožňuje výpočty sémantické podobnosti pomocí metrik jako kosinová podobnost. S vektory od W2V tak lze dělat vektorové operace jako např. ”král” - ”muž” + ”žena”, což vede k vektoru, který je blízký slovu ”královna”

2.2 Transformery a jazykové modely

Nejnovější přístup k reprezentaci jazyka přinesla architektura transformer 4.3 [3], které využívá mechanismus attention k efektivnímu zpracování textu bez nutnosti sekvenčního zpracování jako u RNN [4] (pořadí má jistý bias, který ovlivňuje výstup). Transformery umožnily vznik velmi výkonných jazykových modelů, jakou jsou BERT (Bidirectional Encoder Represntations from Transformers) [5], GPT (Generative Pre-trained Transfomer) [6, 7] a mnoho dalších, které dosahují vynikajících výsledků v úlohách zpracování přirozeného jazyka. [8, 9, 10]

2.3 České jazykové modely

Významný rozvoj v tvorbě vektorových repezentací pro české texty přinesla práce Some Like It Small: Czech Semantic Embedding Models for Industry Applications [11]. Ta je zaměřena na trénování slovních embeddingů pro češtinu s důrazem na efektivitu a použitelnost v průmyslových aplikacích díky menší velikosti modelu - oproti ostatním větším jazykovým modelům slibuje 5x vyšší rychlost a 8x menší velikost. V této práci se použily tři různé metodiky ze kterých vznikly různé předtrénované modely.

První metodikou bylo použití kombinace RetroMAE [12] a enkodérem BERT [5] v jeho menší podobě. Trénovací nastavení obsahovalo - 12 vrstev, 256 velikost skryté vrstvy, WordPiece tokenizer [13], velikost slovníku 57 226 - obsahující anglické i české tokeny. Pro optimalizaci se využila metoda AdamW [14] s learning rate 5e-4 a velikostí dávky 512. Model se trénoval na neveřejném českém korpusu (253 GB upravených českých textů) na dvě epochy.

Další metodika využila multilingual distillaiton [15], kde s použitím modelu all--mpnet-base-v2 jako učitele a modelem BERT s česko-anglickým slovníkem jako studenta vyústila v překonání předtrénovaných multilinguálních modelů. S využitím dvou datasetů tak vznikly dva modely - Dist-MPNetCzEng a Dist-MPNet-ParaCrawl.

Pro další zvýšení kvality byly všechny předtrénované modely následné optimalizovány pomocí doladění bez učitele. Tento krok měl zlepšit reprezentaci embeddingů bez potřeby anotovaných dat. Mezi použité přístupy patří SimCSE, RankCSE a InfoCSE. Pro trénink byla využita česká Wikipedie. Trénování všech modelů probíhalo po dobu 3 epoch, přičemž byl použit optimalizátor AdamW.

Vyhodnocení modelů probíhalo v těchto kategoriích - zero-shot evaluace, lineární klasifikace (linear probing) a dolaďování modelů na konkrétní úlohy. Ve výsledku modely vytvořené metodikou multilingual distillation dosáhly nejlepších výsledků napříč většinou úloh. Model RetroMAE-Small dosáhl překvapivě vysokého výkonu jak ve zero-shot evaluaci, tak v úlohách s doladěním na konkrétní úlohu. Na úloze relevance dotaz-dokument byly malé modely schopné dorovnat nebo překonat větší modely.

2.4 Jazykové modely E5

Dalším model pro tvorbu vektorových reprezentací je model Multilingual-E5 (mE5) [16], která v současnosti představuje state-of-the-art (SOTA) mezi opensource modely. Jsou k dispozici tři různé velikosti modelu (malý/běžný/velký), které rozšiřují anglické e5 modely z práce [17] (založené na architektuře RoBERTa [18]).

Tréninkový proces probíhal ve dvou krocích. První fáze představuje weakly-supervised contrastive pre-training, které probíhalo na miliardě dvojic textů získaných z ruznách zdrojů (Wikipedia, Reddit, StackExchange, ...). Byla použitá velká velikost dávky - 32 tisíc. Trénink trval 30 000 kroků s použitím ztrátové funkce InfoNCE [19]. Jako optimalizační metoda se využila AdamW.

Ve druhé fázi bylo provedeno doladění na kombinaci více než 1,6 milionu ručně anotovaných textových párů ze známých datasetů jako MS-MARCO [20] nebo SQuAD [21] a mnoho dalších.

Modely mE5 byly otestovány na široké škále úloh a prokázaly vysokou kvalitu jak v anglickém jazyce tak v mnoha ostatních. Na benchmarku MTEB model mE5--large-instruct překonal nejen předchozí nejlepší multilinguální model Cohere-multilingual-v3, ale i silný anglický model BGE-large-en-v1.5. V úloze multilinguálního vyhledávání modely výrazně předčily dříve používaný mDPR [22]. Také v úloze bitext mining model mE5-large-instruct překonal specializovaný model LaBSE [23] díky širšímu jazykovému pokrytí a instrukčnímu učení.

2.5 Cíle práce

Pro tuto bakalářskou práci byly stanoveny následující cíle:

1. Seznamte se s problematikou vektorizace textu pomocí velkých jazykových modelů a proveďte rešerši existujících metod a modelů.

2. Na základě provedené rešerše vyhodnoťte vybrané modely pro vektorizaci textu na vhodné vývojové sadě.

3. Vytvořte vlastní jazykový model optimalizovaný na úlohu vektorizace textu.

4. Výsledný model s nejlepším nastavením parametrů porovnejte s výsledky existujících modelů.

3 Vektorová podobnost a prohledávání dokumentů

3.1 Kosinová podobnost

V rámci práce s jazykovými modely určenými pro vyhledávání relevantních dokumentů, hraje kosinová podobnost důležitou roli. Po převedení uživatelského dotazu na vektor, se využije právě funkce kosinové podobnosti pro vyhledání dokumentů relevantních k dotazu uživatele.

Kosinová podobnost je metrika, která měří úhel mezi dvěma vektory v n-rozměrném prostoru. Čím menší je úhel mezi těmito vektory, tím je větší jejich podobnost. V kontextu vyhledávání dokumentů si lze pak každý dokument a uživatelský dotaz představit jako vektor ležící v prostoru, kde nejvíce relevantní dokument je ten, který svírá s dotazovým vektorem nejmenší úhel.

Matematicky je kosinová podobnost definována následně:

$$cos(\theta) = \frac{A \cdot B}{||A||\,||B||} \tag{3.1}$$

kde:

- **A** a **B** jsou dva vektory (v tomto případě vektorizovaný dotaz a dokument)
- **A B** je skalární součin těchto vektorů
- ||**A**|| a ||**B**|| jsou znormalizované vektory

Výsledná hodnota se pohybuje v rozmezí [-1, 1], kdy 1 znamená, že vektory jsou totožné (maximální podobnost), 0 značí vektorovou kolmost a tedy nemají žádnou podobnost a -1 znamená, že vektory jsou opačné

3.2 RAG

V současné době se velké jazykové modely stále častěji využívají v kombinaci s technikou zvanou RAG. Tato technika umožňuje modelům generovat přesnější a kontextuálně relevantnější odpovědi bez nutnosti jejich náročného dolaďování (fine-tuningu) na specifické domény či znalosti. RAG tak představuje elegantní způsob, jak efektivně kombinovat schopnosti jazykových modelů s externími znalostními zdroji.

3.2.1 Princip RAG

Technika RAG staví na konceptu porovnávání vektorových reprezentací dotazů a dokumentů pomocí kosinové podobnosti (viz. 3.1), která slouží k určení míry relevance mezi těmito vektory. Cílem je nalézt co nejrelevantnější kontextové informace vzhledem k položené otázce, které následně model využije při generování odpovědi.

Namísto práce s celými dokumenty se však obvykle používají pouze jejich části, tzv. chunky. Je tomu tak z několika důvodů: jednak mají jazykové modely omezenou kapacitu vstupních tokenů, a jednak celý dokument často obsahuje množství irelevantních nebo dokonce zavádějících informací. Práce s menšími segmenty textu umožňuje efektivnější a přesnější vyhledávání relevantního obsahu.

3.2.2 Předzpracování dokumentů

Prvním krokem v rámci RAG je předzpracování dokumentů. Ty jsou rozděleny na menší části o fixní velikosti, často se zavedením určitého překryvu mezi jednotlivými chunkami. Tento překryv slouží k zajištění, že nedojde ke ztrátě důležitého kontextu na hranicích rozdělení. Následně jsou jednotlivé části dokumentů transformovány do vektorové podoby pomocí jazykového modelu - v produkčním nasazení se tyto vektory ještě znormalizují pro zrychlení výpočtu kosinové podobnosti. Výsledné vektory jsou poté uloženy do vektorové databáze, která umožňuje rychlé a efektivní vyhledávání na základě podobnosti.

3.2.3 Vyhledání relevantního kontextu (Retrieval)

Po zadání dotazu uživatelem následuje první fáze – vyhledání relevantních částí dokumentů. Uživatelův dotaz je převeden do vektorové reprezentace pomocí stejného modelu, jaký byl použit pro dokumenty. Tento dotazový vektor se následně porovnává s vektory uloženými ve vektorové databázi za použití kosinové podobnosti. Vyberou se takové části dokumentů, jejichž vektory jsou nejpodobnější (svírají s dotazem nejmenší úhel), a ty jsou následně předány do další fáze.

3.2.4 Rozšíření dotazu (Augmentation)

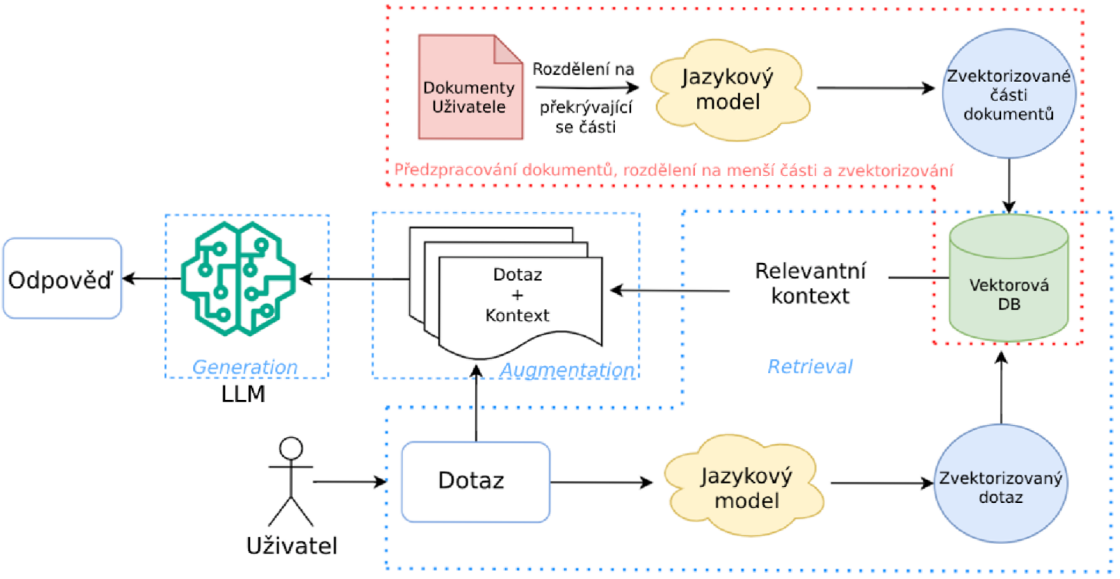
V této fázi se relevantní části dokumentů sloučí s původním dotazem uživatele. Výsledkem je rozšířený dotaz, který obsahuje jak samotný dotaz, tak i nezbytný kontext z dokumentů. Tento vstup je připraven do formátu, který je následně předložen LLM. V praxi se často využívají předem definované šablony , do nichž se doplní konkrétní prvky – např. předchozí konverzace, získané části dokumentů a samotný dotaz uživatele. Správně zvolená struktura dotazu může výrazně ovlivnit kvalitu výsledné odpovědi.

3.2.5 Generování odpovědi (Generation)

V posledním kroku je rozšířený dotaz předložen jazykovému modelu, který na jeho základě vygeneruje finální odpověď. Díky tomu, že model má k dispozici aktuální

a relevantní kontext, může poskytnout přesnější, informacemi podložené výstupy, které by jinak nebylo možné získat pouze na základě jeho tréninkových dat.

Následující obrázek ukazuje všechny fáze a celý průběh RAG:



Obrázek 3.1: Průběh RAG

4 Současné architektury neuronových sítí pro jazykové modely

V posledních desetiletích došlo k významnému pokroku v oblasti neuronových sítí, zejména v jejich aplikaci na zpracování přirozeného jazyka. Tradiční metody, jak jsou statistické modely a jednoduché neuronové sítě, měly omezenou schopnosti zachytit složitou strukturu jazyka a dlouhodobé závislosti v textu. S rozvojem hlubokého učení se objevily pokročilé architektury, které umožnily efektivnější práci s textovými daty a výrazně vylepšily výkon v úlohách - strojový překlad, shrnutí textu nebo generování odpovědí v chatbotových systémech.

Tato kapitola se zaměřuje na klíčové architektury neuronových sítí, které utvářely modení přístupy k NLP. Nejprve jsou popsány rekurentní neuronové sítě (RNN) [4], které umožnily modelům pracovat se sekvenčními daty, ale trpěly s dlouhodobou pamětí. Následně Long Short-Term Memory (LSTM) [4], vylepšenou verzi RNN, která lépe zachytává dlouhodobé závislosti v textu. Nakonec transformery [3], které představují revoluční změnu ve zpracování textu a dnes tvoří základ nejlepších jazykových modelů, jako jsou BERT [5] a GPT [7]. Podrobněji je věnován prostor klíčovým komponentám transformerů - tokenizace, enkodér, self-attention mechanismus a využití těchto modelů v praxi. Transformery umožnily nový přístup k NLP, kde se modely učí hlubší sémantické vztahy a efektivněji pracují s dlouhými texty, což vedlo k jejich vysokému nasazení v široké škále aplikací.

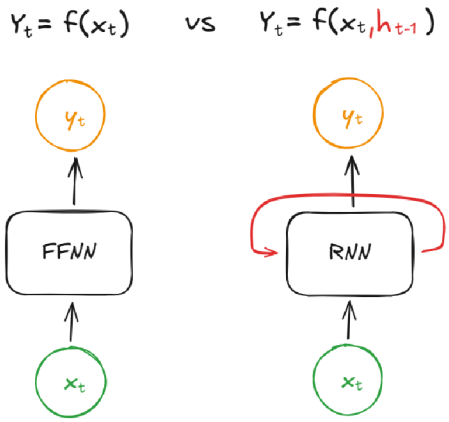
4.1 Rekurentní neuronové sítě

V tradičních neuronových sítí jsou vstupy a výstupy zpracovávány nezávisle. Úlohy, jako je predikce dalšího slova ve větě nebo jiné úlohy pracující se sekvenčními daty, však k přesným predikcím vyžadují informace o předchozích vstupech. Kvůli těmto limitacím byly vyvinuty RNN.

Rekurentní neuronové sítě používají mechanismus, kdy se výstup z předešlého kroku vrací jako vstup do dalšího, což jim umožňuje uchovávat informace o předchozích stavech a přenášet je napříč časovými kroky. Díky této konstrukci jsou RNN vhodné pro úlohy, kde je důležitý kontext z předchozích kroků, například pro strojový překlad, rozpoznáni řeči nebo generování textu.

RNN využívá skrytý stav (hidden state), nazývaný také stav paměti (memory state), který uchovává podstatné informace z předchozích vstupů v sekvenci. Základní RNN se tedy skládá z opakující se skryté vrstvy, které pro každý časový krok

t přijímá vstupní vektor x_t, přechozí skrytý stav h_{t-1} a produkuje nový stav h_t.



Obrázek 4.1: Porovnání neuronové sítě a rekurentní neuronové sítě

Ačkoliv RNN přinesly významný posun v práci se sekvenčními daty, jejich použití má několik základních omezení:

1. Vymizení a exploze gradientu - při trénování pomocí zpětné propagace může docházet k tomu, že gradienty pro aktualizaci vah buď extrémně rostou (exploding gradient) nebo se zmenšují natolik, že neuronová síť přestane efektivně trénovat (vanishing gradient). Tento problém tak zásadně limituje uchovávání závislostí u dlouhých sekvencí - krátkodobá paměť.

2. Sériové zpracování - na rozdíl od novějších architektur (viz. 4.3) je výpočet prováděn sekvenčně, což zabraňuje možnosti paralelizace a tím tak zvyšuje výpočetní náročnost.

4.2 Long short-term memory (LSTM)

Pro zmírnění nevýhod základních rekurentních neuronových sítí (především problémy s gradientem), byly vyvinuty pokročilejší varianty jako je právě Long Short-Term Memory (LSTM). Tato architektura obsahuje mechanismus, který pomáhá efektivněji uchovávat informace o dlouhodobých závislostech a řeší problém vymizení gradientu.

LSTM se skládá z:

1. Zapomínací brána (forget gate) - určuje, které informace z paměťové buňky C_{t-1} budou zapomenuty.

2. Vstupní brána (input gate) - rozhoduje, jaké nové informace budou přidány do paměti.

3. Paměťová buňka (aktualizace paměti) - nový stav paměti C_t je kombinací zapomenutých starých informací a nových vstupů.

4. Výstupní brána (output gate) - ovlivňuje, jaké informace z paměťové buňky budou použity pro výstup.

Díky těmto bránám LSTM reguluje tok informací, což umožňuje modelu zpracovat delší sekvence.

Dalším vylepšením pro LSTM sítě jsou Bidirectional LSTM (BiLSTM). Zatímco standardní LSTM zpracovává sekvence pouze jedním směrem (od minulosti k budoucnosti), BiLSTM umožňuje modelu se učit informace z obou směrů - jak z minulosti, tak z budoucnosti. Tento přístup je zvláště užitečný v úlohách, kde je důležitý kontext z obou stran, například při strojovém překladu nebo analýze sentimentu.

Další důležitý krok v evoluci textové vektorizace přivedl ELMo (Embeddings from Language Models) [24]. Ten oproti tradičním metodám jako W2V zavádí kontextově závislé vektorové reprezentace. ELMo používá dvousměrné jazykové modely (BiLM), které se skládají z vrstvy výše zmíněného BiLSTM a pro trénování tak využívá vlastnosti učit se z obou směrů. ELMo proto trénuje dva jazykové modely současně:

- Dopředné LSTM - předpovídání dalšího slova na základě dosud přečtených
- Zpětné LSTM - predikce předchozího slova na základě následujících

Výstupy obou modelů jsou následně zkombinovány a vytváří tak kontextovou reprezentaci každého slova v dané větě. Díky tomu tak stejné slovo může mít v jiné větě jinou reprezentaci. ELMo navíc využívá všechny vrstvy BiLSTM (nejen poslední skrytou jako u tradičních LSTM), kdy každé vrstvě je přiřazena váha, která určuje důležitost pro konkrétní úlohu. tyto váhy jsou upravovány během finetuningu na konkrétním úkolu, díky čemu je ELMo flexibilní a dá se snadno adaptovat na různé aplikace.

4.3 Transformery

Zásadní změnu ve zpracování jazyka přinesla architektura transformer, která byla představena v roce 2017 článkem Attention is all you need [3]. Na rozdíl od rekurentních sítích (RNN i LTSM), které data zpracovávají sekvenčně (kvůli rekurentní struktuře), transformer využívá attention mechanismus (viz. 4.3.3). Ten umožňuje sekvenční data zpracovávat paralelně, díky čemu jsou transformery efektivnější při trénování na velkých objemech dat a tím tak dosahují mnohem lepších výsledků než ostatní modely. Architektura se skládá ze dvou hlavních komponent (obrázek odkaz na obrázek):

- Enkodér - převádí vstupní sekvenci na sekvenci vektorových reprezentací, které zachycují význam každého tokenu v kontextu celé sekvence.
- Dekodér - generuje výstupní sekvenci na základě reprezentací z enkodéru a předchozích vstupů

Každá tato část obsahuje několik identických modulů, které kombinují attention mechanismus a feedforward vrstvy, reziduální spoje a normalizaci vrstev pro zlepšení stability. Z hlediska vektorizace textu nás zajímá především enkodér pro generování vektorové reprezentace slov.

4.3.1 Tokenizer

Tokenizace je klíčový krok v předzpracování vstupních textových dat pro jazykové modely. Úkol tokenizéru je převedení vstupního textu na sekvenci tokenů, které jsou následně mapovány na číselné hodnoty (token ID). Těmto hodnotám odpovídá embedding vektor, aby mohly být zpracovány neuronovou sítí. Tokenizace hraje zásadní roli, jelikož efektivní rozdělení textů ovlivňuje kvalitu výsledného modelu. Zároveň jsou pro tvorbu tokenizeru použité takové metody, které následné umožní tokenizovat i dosud neviděné vstupní sekvence.

- Word-based tokenizace - rozděluje text na jednotlivá slova podle mezer. Nevýhodou je pokud narazí na slovo, které není ve slovníku, tak musí použít speciální token označující neznámé slovo, což vede ke ztrátě informace.
- Subword tokenizace - rozděluje slova na menší známé části a dokáže tak pokrýt i slova, které nezná. Tento způsob tokenizace se může rozdělit na další typy jako jsou:
  - WordPiece [13] - častá slova zůstávají celá, zatímco méně častá se rozkládají. Tento způsob využívá BERT.
  - Byte Pair Encoding (BPE) [25] - nahrazování nejčastější sousední znaky nebo sekvence znaků novými tokeny. Je často používaný v modelech jako GPT nebo RoBERTa.
  - SentencePiece [26] - využívá BPE a Unigram Language Model (pravděpodobnostní metoda optimalizující sadu tokenů) jako dvě hlavní metody segmentace. Díky svojí flexibilitě se stal velmi rozšířeným.

Zvolená metoda tokenizace ovlivní to, jaké tokeny se ve slovníku budou nacházet, což ovlivňuje délku vstupní sekvence do modelu. Velikost slovníku se nastavuje předem jako hyperparametr a není přímo určena metodou tokenizace.

Při volbě velikosti slovníku je důležité zohlednit velikost a rozmanitost trénovacích dat - příliš velký slovník vede k tomu, že je třeba méně rozkladů na menší části, ale vyžaduje u modelu více parametrů pro zpracování většího množství tokenů. Naopak malý slovník vede k nadměrnému rozkladu slov na menší části, čímž prodlužuje délku vstupní sekvence. [27]

4.3.2 Poziční enkódování

Poziční enkódování je nezbytnou součástí architektury transformerů, která umožňuje modelu pracovat s informací o pořadí slov v sekvenci. Transformery na rozdíl od rekurentních neuronových sítí zpracovávají vstupní tokeny paralelně, což znamená,

že samy o sobě nedisponují žádným mechanismem, jak rozlišit pořadí slov. Aby se tato informace do modelu dostala, je ke každému tokenu přičten poziční vektor, který reprezentuje jeho pozici v celé sekvenci.

Poziční informace může být zakódována různými způsoby. Tradiční metodou je absolutní nebo relativní poziční kódování, kdy každé pozici v sekvenci odpovídá unikátní vektor. Nejčastěji se tyto vektory generují pomocí funkcí sinus nebo kosinus s různými frekvencemi. Tento přístup umožňuje modelu zachytit nejen pořadí slov, ale i jejich vzájemné vzdálenosti a vztahy.

Další způsoby zahrnují trénovatelné poziční vektory nebo jiné matematické metody, které splňují podmínky rozlišitelnosti pozic v sekvenci. Cílem všech těchto metod je umožnit modelu efektivně zachytit sekvenční strukturu textu a zvýšit jeho schopnost porozumět jazykovým souvislostem.

4.3.3 Attention mechanismus

Attention (někdy nazývaný jako scaled dot-product attention) je nejdůležitější prvek enkodéru transformeru, který umožňuje modelu zachytávat vztahy mezi jednotlivými slovy (tokeny) ve vstupní sekvenci, bez ohledu na jejich vzdálenost. Tento mechanismus je hlavním důvodem, proč transformery dosahují vynikajících výsledků v úlohách zpracování přirozeného jazyka. Každý token ve vstupní sekvenci získává různé váhy (attention score) vůči ostatním vzorům, což určuje, jak moc by se při zpracování mělo zaměřit na jiné části sekvence.

Na výpočet Attention se nejdříve převedou prvky vstupní sekvence na tři vektory:

- Query (Q) - dotaz: co hledáme v jiných částech sekvence.
- Key (K) - klíč: co ostatní tokeny nabízejí.
- Value (V) - hodnota: co skutečně získáme, pokus se zaměříme na daný token.

Tento převod probíhá za pomocí vynásobení vstupu x_t s příslušnou váhovou maticí (váhy jsou naučené při procesu trénování):

$$(q|k|v)_t = W^{(Q|K|T)} * x_t \tag{4.1}$$

Výpočet následně probíhá takto:

1. Vypočtení skóre pozornosti (attention score) - každý token (Query) je spárován se všemi ostatními tokeny (Keys) pomocí skalárního součinu, což určuje míru podobnosti mezi nimi:
$$score(Q, K) = Q \times K^T \tag{4.2}$$
2. Škálování skóre - pro zabránění příliš vysokým hodnotám (u dlouhých sekvencí), výsledky se vydělí odmocninou z dimenze vektoru (d_k):
$$\frac{Q \times K^T}{\sqrt{d_k}} \tag{4.3}$$

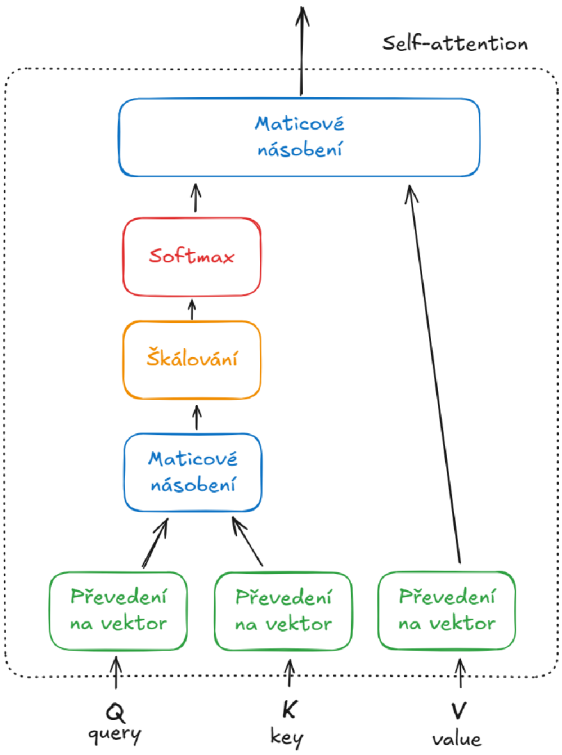
3. Aplikace softmax - výsledné skóre se převede na pravděpodobnostní distribuci pomocí softmax funkce:

$$\text{Váhy pozornosti} = softmax(\frac{Q \times K^T}{\sqrt{d_k}}) \tag{4.4}$$

4. Vynásobení vektorem Value - každý token dostane své konečné reprezentace jako vážený součet hodnot všech tokenů:

$$\text{Attention(Q,K,V)} = softmax(\frac{Q \times K^T}{\sqrt{d_k}}) \times V \tag{4.5}$$

Transformer ovšem nevyužívá pouze jeden mechanismus attention, ale využívá několik hlav najednou (multi-head attention). Každá tato hlava používá jiné váhy, což modelu umožňuje zachytit různé vztahy mezi slovy. Právě díky náhodné inicializaci na začátku, dokáže pak každá hlava zpracovat jiné závislosti. Před vstupem do multi-head attention se musí vektory rozšířit pomocí lineární transformace na matice, které jsou rozděleny mezi jednotlivé hlavy. Po výpočtu jsou výsledky z každé hlavy spojené dohromady a projdou lineární transformací. Tento krok umožňuje, aby se informace ze všech hlav zkombinovaly do jednoho reprezentativního vektoru. Následný obrázek zachycuje popsaný postup pouze pro jednu hlavu:

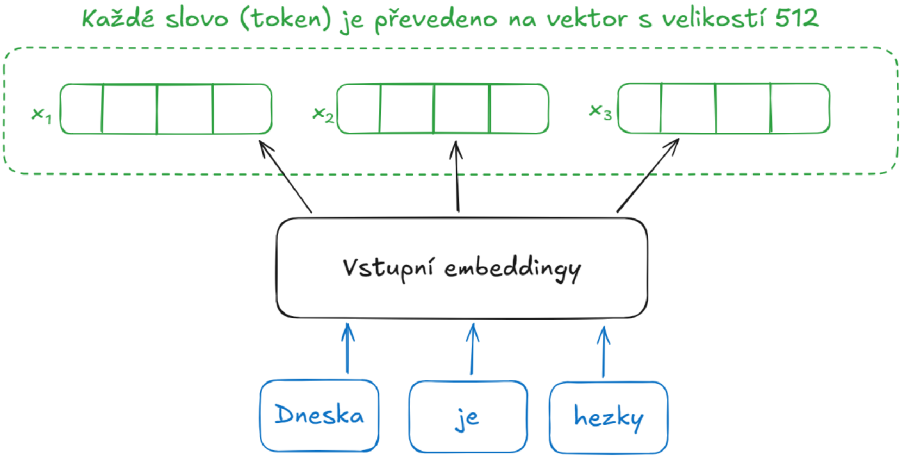


Obrázek 4.2: Průběh attention

4.3.4 Enkodér

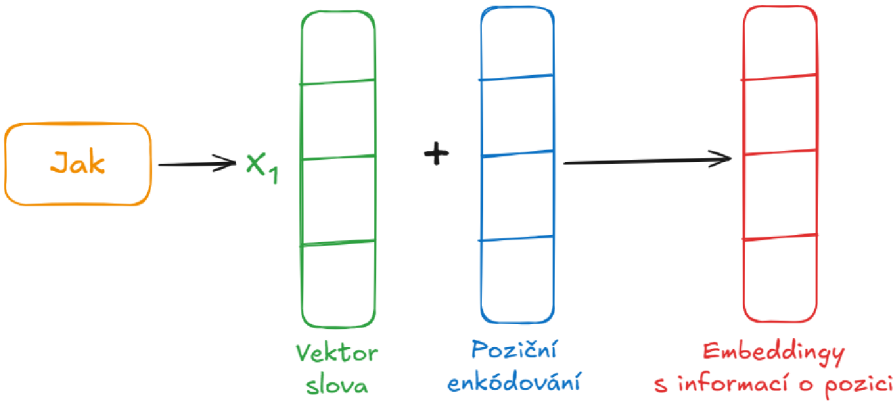
Část architektury transformerů, která je v této práci nejdůležitější se nazývá enkodér. Ten má na starost tvorbu vektorové reprezentace vstupních tokenů, která se využívá například v technice RAG 3.2 nebo směřuje dál do dekodéru.

Prvním krokem enkodéru je převedení slov na tokeny a z tokenů na vektory pomocí embedding vrstvy. Tyto vektory si lze představit jako prosté Word2Vec vektory, které zatím neobsahují kontextovou informaci. Enkodér následně přijme seznam vektorů, kde každý má fixní velikost.



Obrázek 4.3: Převedení vstupních tokenů na vektory

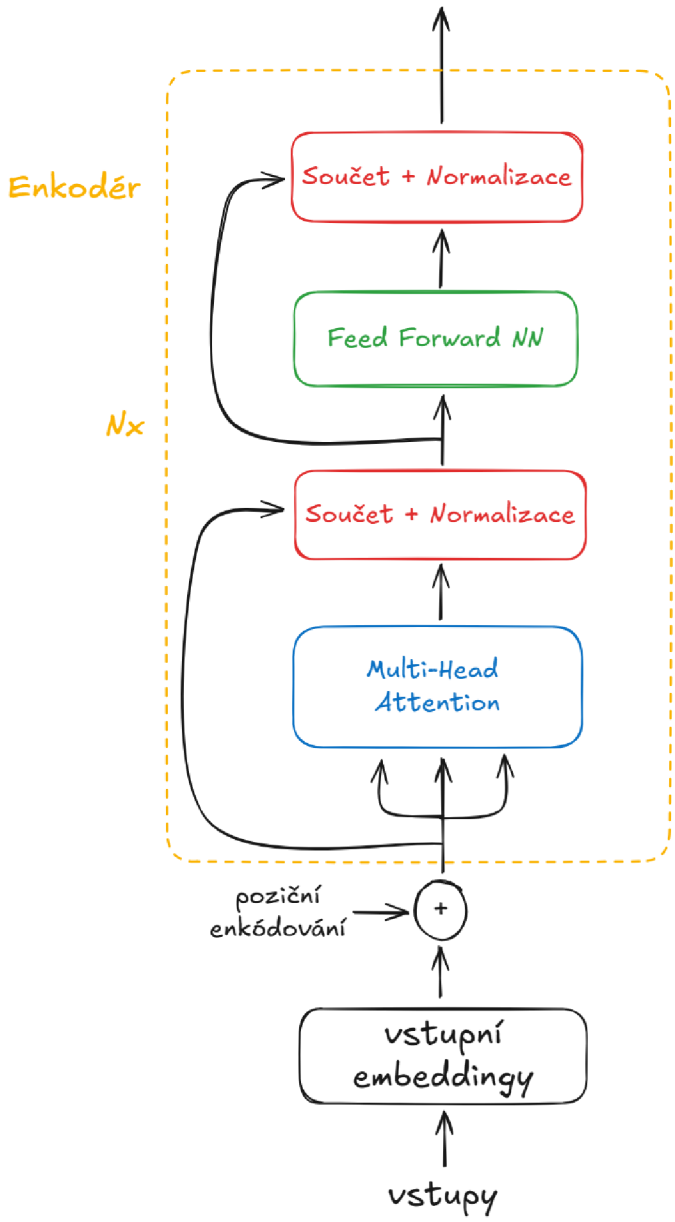
Následuje poziční enkódování pro přidání informace o pozici každého tokenu ze vstupní sekvence.



Obrázek 4.4: Přidání informace o pozici

Následně enkodér obsahuje několik identický modulů za sebou (6 v originálním modelu [3]) - tyto moduly slouží pro převedení vstupní sekvence na vektorovou reprezentaci, která zahrnuje význam tokenů v kontextu celé sekvence. Tato vrstva se skládá ze dvou dílčích částí:

- Multi-head attention mechanismus
- Feedforward (Plně propojená síť)



Obrázek 4.5: Enkodér

Namísto jednoho attention mechanismus 4.3.3, se zde využívá více hlav (Multi--Head Attention). Každá hlava má svoje vlastní matice W_Q, W_K, W_V, čímž model zachycuje různé významové vztahy mezi slovy. [28]

Kromě toho zahrnuje reziduální spojení kolem každého dílčího modulu, které je spojeno se vstupem a normalizačním krokem pro zmírnění problému mizejícího gradientu a umožňuje tak vytvářet hlubší modely. Stejný přístup se aplikuje i po

FeedForward vrstvě. Tuto síť si lze představit jako dvojici lineárních vrstev, mezi nimiž se nachází aktivační funkce ReLU:

$$FFN(X) = max(0, XW_1 + b_1)W_2 + b_2 \tag{4.6}$$

Kde W_1, W_2 jsou matice vah, b_1 a b_2 jsou biasy. Vstup této sítě se nejdříve rozšíří v první vrstvě a následně se opět zmenší. Výstupem je další transformovaná reprezentace tokenu.

Výsledným výstupem z posledního enkodéru je matice ve tvaru:

$$H = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_N \end{bmatrix} \tag{4.7}$$

Kde h_i je vektor reprezentující i-tý token ve vstupní sekvenci a každý sloupec odpovídá skrytým rozměrům modelu. Nejedná se pouze o obyčejný embedding slova, ale obsahuje kontext získaný z celého vstupního textu díky self-attention mechanismu.

4.3.5 BERT

BERT (Bidirectional Encoder Representations from Transformers) [5] je jeden z nejvíce významných jazykových modelů, který vychází ze zmíněné architektury transformer, ale s několika klíčovými úpravami, které ho odlišují od původního návrhu.

BERT je založen pouzen na enkodérové části a existuje ve dvou hlavních variantách:

1. BERT-base: 12 vrstev (12 hlav self-attention), 768 skrytých rozměrů a zhruba 110 miliónů parametrů

2. BERT-large: 24 vrstev (16 hlav self-attention), 1024 skrytých rozměrů a zhruba 340 miliónů parametrů

Vstupní i výstupní dimenze jsou pevně stanovené, což umožňuje efektivní přenos znalostí do dalších úloh, zároveň více vrstev dovoluje modelu zachytávat vztahy v delších sekvencích.

Hlavní odlišností BERT od standardního transformeru spočívá v jeho způsobu trénování. Většina modelů (především dekodérové modely) generují slova autoregresivně - po jednom slovu zleva doprava, což omezuje jejich schopnost využívat informace z celé sekvence. Vstupní kontext zpracují na začátku celý, ale výstupní sekvenci generují postupně. BERT nepracuje sekvenčně, ale místo toho využívá obousměrný kontext - zpracovává celý text najednou a to z obou směrů současně a učí se tak vztahy mezi všemi slovy v sekvenci. To umožňuje modelu modelu zachytit hlubší vztahy mezi slovy a získat tak bohatší vektorové reprezentace.

BERT má unikátní přístupy k trénování, jeho první přístup je Masked Language Model (MLM), kdy se náhodně maskují tokeny ve vstupní sekvenci a model se je

snaží zpět rekonstruovat - do modelu se přidá token <MASK>, který značí zamaskované slovo. Druhým přístupem je Next Sentence Prediction (NSP), kdy je cílem předpovědět, zda druhá věta logicky navazuje na první - model dostává páry vět a musí určit, jestli první věta opravdu následovala tu druhé. Učí se tak vazby mezi jednotlivými větami.

BERT produkuje vektory pro každý token ve vstupu, které lze využít v různých NLP úlohách:

- Klasifikace text - používá se speciální token [CLS], jehož výstupní vektor se předává do klasifikátoru - analýza sentimentu, kategorizace textu
- Srovnání textů - CLS vektor nebo průměr tokenových vektorů lze použít pro výpočet kosinové podobnosti mezi větami
- Vyhledávání informací - lze využít pro vyhledávání relevantních dokumentů nebo hledání odpovědí na otázky
- Generování odpovědí (quesiton answering) - model dostane otázku + kontextový text a musí najít správnou odpověď

4.3.6 RoBERTa

RoBERTa (Robustly Optimized BERT Pretraining Approach) [18] je vylepšená verze modelu BERT [5] (4.3.5), která byla představena týmem z MetaAI. Model zachovává architekturu BERT, ale přináší několik klíčových optimalizací v procesu trénování, které vedou k vyšší přesnosti a lepší generalizaci ve zpracování přirozeného jazyka.

Hlavní vylepšení RoBERTa oproti BERT:

- Delší a efektivnější trénování na větším množství dat
- Odstranění Next Sentence Prediction
- Dynamické maskování v MLM
- Větší dávky a sekvence pří trénování
- Větší trénovací dataset

RoBERTa nabízí dva hlavní modely:

- RoBERTa-base: 12 vrstev (12 hlav), velikost skrytého stavu 768 a zhruba 125 miliónů parametrů
- RoBERTa-large: 24 vrstev (16 hlav), velikost skrytého stavu 1024 a zhruba 335 miliónů parametrů

BERT byl učen na úloze NSP, což nutilo model určovat, zda dvě věty na sebe navazují, výzkum ovšem ukázal, že NSP není nutné a jeho odstranění zlepšuje výkon modelu. RoBERTa je proto trénována pouze pomocí MLM a dosahuje tak lepších výsledků než BERT. Dále využívá 5x více trénovacích dat než BERT:

- BERT dataset (16GB):
  - Toronto BooksCorpus (800M slov)
  - English Wikipedia (2,500M slov)
- RoBERTA dataset (160GB):
  - BERT dataset
  - OpenWebText
  - Common Crawl - CC-News

Další změna u modelu RoBERTa je dynamické maskování. BERT využívá statické maskování, což znamená, že model vidí stejný maskovaný text ve všech epochách trénování. To může zapříčinit, že se model může naučit zapamatovat konkrétní maskované tokeny místo porozumění širším jazykovým vztahům. Dynamické maskování znamená, že maskované tokeny se mění v každé epoše, což vede k lepší generalizaci modelu a díky tomu tak efektivněji využívá trénovací data.

Další vylepšení souvisí s většími dávkami a delšími sekvencemi. BERT byl trénován s maximální sekvencí 512 tokenů, ale většinou používal kratší sekvence. RoBERTa využívá delší sekvence (512 tokenů) po celou dobu trénování a má větší batch size (8x) než je u modelu BERT.

RoBERTa, stejně jako BERT, produkuje vektory pro každý token ve vstupu a lze ji tak využít v různých NLP úlohách. Díky efektivnějšímu využití trénovacích dat dosahuje vyšší přesnosti a lepších výsledků.

5 Navržené řešení

Navržený jazykový model byl vyvinut s primárním cílem efektivní vektorizace českých textových dat, což umožňuje jeho využití při vyhledávání relevantních dokumentů na základě podobnosti. Klíčovým prvkem tohoto přístupu je aplikace kosinové podobnosti, která umožňuje porovnání textových reprezentací a určení míry shody mezi uživatelským dotazem a jednotlivými dokumenty. Díky tomuto mechanismu lze model efektivně nasadit pro vyhledávání informací v rozsáhlých textových databázích, čímž se usnadňuje a zpřesňuje identifikace nejrelevantnějších výsledků.

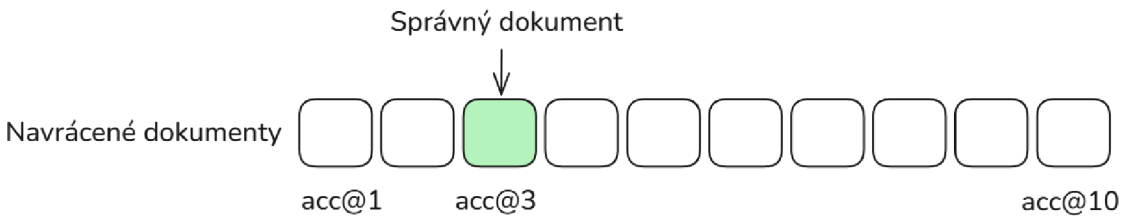
5.1 Použité metriky

Pro objektivní vyhodnocení výkonu jazykového modelu pro navrácení relevantních dokumentů byly použity standardní metriky přesnosti známé jako Top@k accuracy, konkrétně ve variantách acc@1, acc@3 a acc@10. Tyto metriky měří, zda se očekávaný (správný) výstup nachází mezi prvními k výsledky navrácenými modelem. Hodnota metriky se pohybuje v rozmezí od 0 do 1 (nebo 0–100 %), kde vyšší číslo indikuje lepší výkon modelu.

- acc@1 (Top1 accuracy): Měří, zda je správný výsledek hned na první pozici. Jedná se o velmi přísné kritérium, protože model musí přesně odhadnout nejlepší odpověď.
- acc@3 (Top3 accuracy): Umožňuje modelu chybu na prvních dvou pozicích, ale stále hodnotí úspěch, pokud je správná odpověď mezi prvními třemi výsledky.
- acc@10 (Top10 accuracy): Hodnotí, zda se správná odpověď nachází mezi prvními deseti návrhy. Tato metrika lépe reflektuje praktické použití v technice RAG, jelikož se z vektorové databáze vrací více částí dokumentů najednou, pro lepší přenost a kontext.

Použití více metrik umožňuje komplexnější pohled na kvalitu výsledků. Například dva modely mohou mít podobné acc@1, ale výrazně se lišit v acc@10 – to může znamenat, že jeden model je konzistentnější ve vracení relevantních výsledků i při nižší přesnosti na prvním místě.

Na následujícím obrázku 5.1 je znázorněna situace, kdy správná odpověď není na první pozici (acc@1 = 0), ale je na třetí pozici (acc@3 = 1) a současně spadá mezi prvních deset (acc@10 = 1). Taková situace je v praxi běžná a ukazuje, proč je vhodné posuzovat výkon modelu více metrikami zároveň.



Obrázek 5.1: Použití metrik acc@1, acc@3, acc@10

5.2 Základní architektura

Architektura použitá v navrženém řešení vychází z modelu XLM-RoBERTa-base, který je vícejazyčnou variantou modelu RoBERTa [18, 29]. Výběr této architektury byl podpořen experimentálním vyhodnocením, popsaným v kapitole 6.1, kde RoBERTa překonala ostatní porovnávané modely jak v přesnosti, tak v obecné schopnosti modelovat sémantiku textu.

Zvolený model XLM-RoBERTa-base je předtrénován na rozsáhlém multilinguálním korpusu, zahrnujícím více než 100 jazyků, včetně češtiny. Tím získává schopnost zachytit nejen jazykové vzory jednotlivých jazyků, ale také jejich vzájemné vztahy. Tato vlastnost je klíčová při vektorizaci textů, které mohou obsahovat jazykové interference nebo prvky více jazyků. Díky tomu model disponuje vyšší mírou generalizace a robustnosti při práci s reálnými textovými daty.

Pro účely této práce nebyl model trénován od začátku, ale využilo se přístupů fine-tuningu a následné optimalizace embedding funkcí nad specifickým trénovacím datasetem, zaměřeným na úlohu vektorizace textu a vyhledávání relevantních dokumentů.

5.3 Využité datasety

Jelikož byla snaha výsledný jazykový model zaměřit na vektorizaci textu a následné vyhledávání relevantních dokumentů pomocí kosinové podobnosti bylo třeba využít speciálně upravené datasety. Pro doladění základního modelu se využily celkem 4:

Dataset	**Počet párů [tisíce]**
DaReCzech	97
SQuAD v1.1	18,5
SQuAD v2.0	18,6
iDnes	500
Celkem	634,3

Tabulka 5.1: Použité datasety

Prvním datasetem je DaReCzech [30], což je sada pro hodnocení relevance textu

v češtině. Obsahuje více než 1,6 milionu anotovaných datových záznamů ve formátu: uživatelský dotaz, URL anotovaného dokumentu a reprezentace dokumentu (obsah). Tato datová sada byla vytvořena pro trénink a vyhodnocení modelů na úlohách vyhledávání a hodnocení relevance textů. Pro potřeby této práce se však využila pouze menší část datasetu nazvaná train-small, která obsahuje 97 tisíc záznamů. Z této podmnožiny se následně pracovalo pouze s dvojicí dotaz – obsah dokumentu, čímž se dataset zjednodušil pro trénink jazykového modelu zaměřeného na vyhledávání.

Další dva použité datasety jsou různé verze českého SQuADu [31] (The Stanford Question Answering Dataset [21]). Tento dataset je určen primárně pro úlohu question answering, kde obsahuje několik korpusů textů, ke kterým jsou přiřazeny otázky s odpověďmi. Tento formát umožňuje modelu učit se extrahovat relevantní informace z textu na základě uživatelského dotazu. Pro potřeby této práce byly z těchto datasetů vybrány pouze dvojice korpus – otázka, kde otázka slouží jako simulovaný uživatelský dotaz a korpus jako reprezentace potenciálně relevantního dokumentu. Tímto způsobem byl dataset adaptován pro trénink vyhledávacího modelu.

Posledním datasetem využitým pro doladění jazykového modelu je iDnes dataset, který byl vytvořen vlastní extrakcí dat ze zpravodajského portálu iDnes. Tento dataset byl získán scrapováním článků, přičemž bylo analyzováno přes půl milionu článků. Z každého článku byly extrahovány dvě klíčové části: název článku a anotace. Název článku zde slouží jako analogie k uživatelskému dotazu a anotace jako reprezentace relevantního dokumentu, jelikož zpravidla rozšiřuje kontext nadpisu a shrnuje obsah článku. Tento dataset byl vytvořen za účelem rozšíření trénovací množiny a přizpůsobení modelu reálným dotazům a textům v češtině.

Využitím těchto datasetů se podařilo vytvořit robustní trénovací množinu, která modelu umožňuje efektivně se učit, jak propojit uživatelské dotazy s relevantními dokumenty v češtině. Každý dataset přispěl k lepšímu pokrytí různých aspektů vyhledávání – od anotovaných dotazů, přes otázky a odpovědi, až po reálná novinová data. Díky tomu lze očekávat, že výsledný model bude schopný efektivně odpovídat na uživatelské dotazy a nalézat odpovídající dokumenty s vysokou přesností.

5.4 Doladění modelu

Základní předtrénované jazykové modely vykazují velmi dobré výsledky v různých úlohách zpracování přirozeného jazyka. Nicméně jejich výkonnost lze výrazně zlepšit pomocí doladění (fine-tuningu) na konkrétní úlohu.

V této práci bylo doladění provedeno pomocí knihovny Sentence Transformers, která poskytuje nástroje pro efektivní práci s jazykovými modely, včetně jejich doladění na úlohy srovnávání vět či hledání podobných textů. Vzhledem k vysokým výpočetním nárokům spojeným s doladěním větších modelů byla využita výpočetní infrastruktura MetaCentra Cesnet, která umožňuje přístup k výkonným GPU clusterům.

Doladění modelu probíhalo na GPU clusteru Glador v MetaCentru Cesnet, který poskytuje výpočetní uzly s následujícími komponentami:

- CPU: 64x AMD EPYC 7543
- RAM: 512 GiB
- GPU: 4x nVidia A40

Celkový proces doladění trval 168 hodin, přičemž hlavní výpočetní zátěž byla přenesena na grafické karty. Pro sledování průběhu doladění byla použita platforma Comet ML, která umožnila zaznamenávání klíčových metrik, jako je ztrátová funkce (loss), přesnost modelu a další parametry.

Pro doladění modelu byly nastaveny následující hyperparametry:

- Velikost dávky: 56
- Počet epoch: 38
- Learning rate: 2e-5
- Počet evaluačních kroků: 50
- Ztrátová funkce: MultipleNegativesRankingLoss

Velikost dávky určuje počet trénovacích vzorků, které jsou zpracovány najednou před aktualizací vah modelu. Hodnota 56 byla zvolena na základě experimentálního vyhodnocení v kapitole 6.2.2. Epochy představují počet kompletních průchodů trénovacími daty. Zvolená hodnota 38 byla nastavena experimentálně tak, aby bylo dosaženo konvergence modelu bez přetrénování. Learning rate určuje rychlost, jakou se model učí během aktualizace vah. Příliš vysoká hodnota by mohla vést k nestabilitě trénování, zatímco příliš nízká by zpomalila konvergenci. Na základě experimentálního vyhodnocení v kapitole 6.2.1 je hodnota nastavena na 2e-5. Finální vektor se získává průměrem matice. Každých 50 trénovacích kroků byl model vyhodnocen na validační sadě, aby bylo možné sledovat jeho výkon a případně včas detekovat přetrénování. MultipleNegativesRankingLoss je specifická ztrátová funkce používaná pro úlohy srovnávání textů, zejména při vyhledávání relevantních dokumentů nebo párování dotazů s odpověďmi. Funguje na principu minimalizace vzdálenosti mezi správným párem (dotaz-odpověď) a maximalizace vzdálenosti mezi nesprávnými kombinacemi. To znamená, že model se učí přiřazovat vyšší skóre relevantním odpovědím a nižší skóre nerelevantním. Tato ztrátová funkce je efektivní zejména při trénování modelů pro úlohy sémantického vyhledávání a podobnosti textů.

5.5 Výsledky

Výsledky doladěného modelu jsou porovnány s několika dalšími existujícími modely a shrnuty v tabulce 5.2. Tato tabulka ukazuje, jak si navržené řešení vede ve srovnání s jinými modely, včetně široce používaných modelů pro vícejazyčné vektorové reprezentace.

model	**acc@1** [%]	**acc@3** [%]	**acc@10** [%]
Multilingual-E5-base [16]	65.4	86.8	91.7
Multilingual-E5-small [16]	63.0	85.0	90.5
distiluse-base-multilingual-cased-v2 [32]	42.6	68.5	77.3
LaBSE [23]	42.3	68.2	76.9
Seznam/simcse-dist-mpnet-czeng-cs-en [11]	36.1	62.1	71.8
Seznam/RetroMAE-small-cs [11]	27.0	49.9	59.9
Seznam/simcse-small-e-czech [11]	3.3	9.9	14.8
E5-Czech-Base	63.2	81.2	**92.0**

Tabulka 5.2: Porovnání s ostatními modely

Modely otestované v tabulce 5.2 byly testované na evaluační sadě dat složené převážně z českého datasetu SQuAD (jiná sada než byla použita v procesu doladění). Z tabulky lze vidět, že navržený model dosahuje velmi dobrých výsledků. Zatímco model Multilingual-E5-base má nejlepší skóre v acc@1 a acc@3, doladěný model ho překonal v acc@10. Oproti menší verzi Multilingual-E5-small, si doladěný model vede také dobře a strádá pouze v acc@3.

V porovnání s dalšími modely dosahuje doladěný model výrazně lepších výsledků, často s rozdílem až 20 % ve všech metrikách. Zvláště zajímavé je, že navržený model překonává modely od společnosti Seznam, které byly trénovány na českých datech. To naznačuje, že doladění na specifickou úlohu může být efektivnější než použití obecného modelu trénovaného na českých textech bez další optimalizace pro konkrétní doménu.

6 Experimentální vyhodnocení

Pro potřeby trénování a vyhodnocování jazykového modelu bylo nutné využít výpočetně výkonné prostředí. Vzhledem k náročnosti těchto operací se využilo prostředí MetaCentrum CESNET, které umožňuje spouštění vlastních skriptů jak interaktivně, tak dávkově.

V rámci této práce byly experimenty realizovány převážně dávkovým způsobem pomocí systému qsub, který slouží k zařazení úloh do výpočetní fronty a jejich přiřazení na odpovídající výpočetní uzly podle specifikovaných parametrů. Pro trénování modelu byly použity následující hardwarové požadavky:

- 1 výpočetní jádro (CPU)
- 64 GB RAM
- 48 GB paměti na GPU
- 15 GB scratch diskového prostoru

Díky těmto parametrům byly skripty typicky přiřazovány na stroj galdor, který je osazen grafickou kartou nVidia A40 s 48 GB paměti, což umožnilo bezproblémové trénování větších modelů a dávkové zpracování dat.

Ještě před samotným spouštěním úloh bylo zapotřebí připravit odpovídající virtuální prostředí. Pro správu balíčků a vytváření virtuálních prostředí jazyka Python se na MetaCentru používá balíčkový manažer Mamba.

Pro vytvoření prostředí byla využita služba Open OnDemand, která umožňuje přístup k prostředkům MetaCentra prostřednictvím webového rozhraní. Tato služba poskytuje uživatelsky přívětivé grafické rozhraní, pomocí kterého lze spouštět terminálové relace, spravovat soubory a jednoduše provádět konfiguraci prostředí bez nutnosti přímé práce přes SSH. Právě díky OnDemand bylo možné snadno a rychle vytvořit požadované prostředí v domovském adresáři, aniž by bylo potřeba řešit nízkoúrovňové připojení nebo ruční správu modulů.

Pomocí Mamby bylo následně sestaveno vlastní virtuální prostředí obsahující všechny klíčové knihovny potřebné pro trénování modelu a provádění experimentů. Mezi nejdůležitější balíčky patřily:

- Python 3.11
- PyTorch 2.5.1 [33]

- PyTorch CUDA 12.4
- Sentence Transformers 3.3.1
- Accelerate 1.2.0
- Comet ML 3.47.4

Virtuální prostředí bylo vytvořeno jen jednou a dále připojováno ke každé dávkové úloze, aby se předešlo opakovanému znovu instalování knihoven, což výrazně šetřilo čas a urychlovalo celkový průběh experimentů.

Pro spouštění experimentů byl využit dávkový režim, kdy byly jednotlivé skripty připraveny jako spustitelné úlohy ve formátu PBS. Tyto úlohy byly vkládány do fronty přes příkaz qsub. Každý experiment měl vlastní skript s definovanými zdroji a parametry, který se následně odesílal na výpočetní uzel.

Během trénování modelu bylo důležité průběžně sledovat metriky jako ztrátová funkce, přesnost a další parametry. K tomuto účelu byla použita platforma Comet ML , která poskytuje prostředí pro sledování a vizualizaci experimentů ve webovém rozhraní.

Pro každý typ experimentu byl v rámci Comet založen vlastní dashboard. Po vytvoření dashboardu se vygeneroval unikátní API klíč, který se přidal do skriptu, aby bylo možné propojit běžící experiment s online rozhraním. Následně se všechny logované metriky automaticky odesílaly na server, kde byly zobrazovány v reálném čase formou grafů a přehledných tabulek.

Tímto způsobem bylo možné jednotlivé experimenty detailně sledovat, porovnávat výsledky mezi různými konfiguracemi a rozhodovat se na základě vizualizovaných dat o dalším postupu nebo nutnosti změn v architektuře modelu.

6.1 Výběr základního modelu

Prvním krokem při tvorbě jazykového modelu bylo zvolení vhodného základního modelu, na kterém následně proběhne doladění. Pro výběr byla provedena praktická evaluace několika vybraných předtrénovaných modelů s cílem nalézt ten, který podává nejlepší výsledky na českých datech.

Při výběru byly záměrně zastoupeny modely různých architektur a přístupů k trénování. Následující přehled stručně popisuje jednotlivé zástupce:

- Multilingual-E5-base / small - moderní modely od Google Research vycházející z architektury XLM-RoBERTa. Jsou trénovány specificky na úlohy retrievalu a vektorizace textu. Podporují více než 100 jazyků.
- LaBSE (Language-agnostic BERT Sentence Embedding) – víceúčelový model navržený pro vícejazyčnou reprezentaci textu s důrazem na sémantickou podobnost.
- DistilUSE – zjednodušená verze USE (Universal Sentence Encoder), postavená na BERT architektuře s podporou více jazyků [32].

- paraphrase-multilingual-mpnet-base-v2 – model optimalizovaný pro detekci parafrází a sémantické podobnosti, vychází z modelu BERT.
- Seznam/simcse-dist-mpnet-czeng-cs-en - vychází z modelu BERT, trénovaný na česko-anglickém datasetu.
- Seznam/retromae-small-cs - vychází z modelu BERT, využívá slovník obsahující české i anglické tokeny, je trénovaný na českém datasetu
- Seznam/simcse-small-e-czech - vychází z modelu ELECTRA [34], trénovaný na českém datasetu.

Pro účely testování byl vytvořen skript využívající knihovnu SentenceTransformers, dostupnou na portálu HuggingFace [35]. Skript zajišťuje načtení a inicializaci všech vybraných modelů, přípravu evaluačních dat z české verze SQuAD 1.1 a 2.0, převedení do formátu dvojice otázka-korpus (viz. 6.3.1) a vyhodnocení pomocí třídy InformationRetrievalEvaluator, která pro každou otázku vyhledá nejpodobnější korpusy.

Praktická část výběru základního modelu probíhala čistě experimentálně – modely byly testovány a seřazeny podle jejich výkonu na dvou variantách evaluačního datasetu. Výsledky metrik acc@1, acc@3 a acc@10 pro SQuAD 2.0 jsou uvedeny v tabulce 6.1.

Model	**acc@1 (%)**	**acc@3 (%)**	**acc@10 (%)**
multilingual-e5-base	**65.4**	**86.8**	**91.7**
multilingual-e5-small	63.1	85.1	90.6
paraphrase-multilingual-mpnet-base-v2	44.1	69.6	79.1
distiluse-base-multilingual-cased-v2	42.6	68.6	77.3
LaBSE	42.3	68.3	77.0
Seznam/simcse-dist-mpnet-czeng-cs-en	36.1	62.1	71.8
Seznam/retromae-small-cs	27.1	49.9	59.9
Seznam/simcse-small-e-czech	3.4	9.9	14.8

Tabulka 6.1: Výsledky modelů na datasetu SQuAD dev2.0

Z dosažených výsledků je zřejmé, že modely Multilingual-E5 jednoznačně dominují všem metrikám. Dokonce i zmenšená verze small zaostává za základní variantou base jen o cca 1,5%.

Naopak modely od společnosti Seznam překvapivě podávaly výrazně horší výsledky, přičemž některé (například simcse-small-e-czech) nedosahovaly ani 15% úspěšnosti v top-10 výsledcích. To naznačuje, že čistě české modely bez rozsáhlého předtrénování na vícejazyčných datech nemusí být pro tuto úlohu nejvhodnější.

Na základě těchto praktických testů byl jako základní model zvolen multilingual-e5-base. Tento model nabízí:

- dostatečnou podporu češtiny díky předtrénováníu na datech ve více než 100 jazycích,

- vhodné vlastnosti pro doladění na specializované úloze.

Výběr modelu tak vycházel čistě z praktického srovnání výkonu na českých datech a používané architektuře. [36]

6.2 Výběr hyperparametrů

Po výběru a otestování základního modelu následovalo jeho doladění na konkrétní úlohu. Nedílnou součástí tohoto procesu je volba vhodných trénovacích hyperparametrů, které mají zásadní vliv na výslednou přesnost a stabilitu modelu. Cílem této fáze bylo experimentálně určit takové hodnoty hyperparametrů, které maximalizují přesnost bez nadměrného přeučení modelu. Hledání probíhalo spuštěním sady experimentů s různými kombinacemi hodnot.

Aby bylo možné efektivně testovat různé hodnoty hyperparametrů, bylo nutné upravit skript pro doladění modelu. Současně byl vytvořen nový řídicí skript, který automatizuje spuštění více instancí trénování s různými kombinacemi hyperparametrů. Tato automatizace byla důležitá z následujících důvodů:

- Pokud by se jednotlivé experimenty spouštěly ručně, hrozilo by, že se spustí všechny instance se stejným (posledně nastaveným) hyperparametrem kvůli frontě výpočetních úloh.
- Automatické spuštění zaručilo konzistentní přiřazení parametrů ke konkrétním instancím.
- Pro každý testovaný hyperparametr byl v prostředí Comet.ml vytvořen samostatný panel, což umožnilo snadné sledování průběhu trénování a srovnávání výsledků mezi jednotlivými experimenty.

6.2.1 Learning rate

Jedním z nejdůležitějších hyperparametrů při trénování neuronových sítí je learning rate (LR), tedy velikost kroku, o který se mění parametry modelu při každé iteraci optimalizačního algoritmu.

- Při trénování modelu od nuly se běžně používají vyšší hodnoty (např. 10^{-3} až 10^{-4}), které modelu umožňují rychleji se učit ze vstupních dat.
- V případě doladění se ale model již nachází v relativně stabilním stavu (je předtrénován na rozsáhlém korpusu), a proto je vhodné zvolit menší hodnoty LR, které jemně doladí váhy bez rizika destabilizace modelu.

V rámci experimentů byly otestovány hodnoty LR v rozsahu $[1e{-}5, 1e{-}6]$. Všechny experimenty měly shodné následující parametry:

- Batch size: 56 (zvolená jako základní hodnota)

- Doba trénování: 48 hodin (dostatečné pro konvergenci a vytvoření validní trénovací křivky)
- Learning rate schedule: aktivní decaying learning rate, která zajišťovala postupné snižování hodnoty LR v průběhu trénování. Tím docházelo k jemnějším úpravám modelu v pozdějších fázích trénování.

Každý experiment byl zaznamenán do prostředí Comet.ml, kde bylo možné sledovat a porovnávat vývoj metrik jako přesnost (acc@1), ztráta (loss) nebo rychlost konvergence. Pro každou testovanou hodnotu LR byla následně vyhodnocena výsledná přesnost modelu. Cílem tohoto testování bylo určit hodnotu learning rate, která vede k nejlepší rovnováze mezi rychlostí učení a přesností výsledného modelu.

Následující tabulka znázorňuje vývoj přesnosti v závislosti na zvolené hodnotě learning rate.

Learning rate	**acc@1 [%]**
1e-6	49.3
2e-6	52.2
3e-6	53.3
4e-6	53.4
5e-6	54.2
6e-6	54.1
7e-6	54.4
8e-6	54.3
9e-6	54.5
1e-5	54.5
2e-5	**55.3**
3e-5	53.6
4e-5	54.1
5e-5	53.1
6e-5	52.6

Tabulka 6.2: Výsledná přesnost modelu při různých hodnotách learning rate

Na základě těchto experimentů byla jako optimální zvolena hodnota learning rate - 2e-5, která dosáhla nejvyšší přesnosti 55,3%.

6.2.2 Batch size

Dalším důležitým hyperparametrem je velikost dávky tzv. batch size, tedy velikost dávky trénovacích příkladů, která se zpracovává najednou v rámci jednoho kroku optimalizace. Tento parametr výrazně ovlivňuje rychlost a stabilitu trénování, ale i výslednou přesnost modelu.

- Menší batch size vede k častějším aktualizacím vah, ale zároveň přináší větší šum do učení a může zpomalit konvergenci.

- Větší batch size zpravidla umožňuje rychlejší trénování a stabilnější gradienty, avšak může vést k horší schopnosti modelu generalizovat.

Testování probíhalo podobně jako u learning rate – pro každý experiment byla zachována stejná konfigurace s výjimkou hodnoty batch size. Learning rate byla během těchto experimentů nastavena na již optimalizovanou hodnotu 2e−5.

Při volbě konkrétních hodnot byly testovány dávky o velikostech, které odpovídají mocninám čísla 2 (4, 8, 16, 32, 64). Tyto velikosti jsou optimální z hlediska výpočetní efektivity na GPU architekturách. Některé moderní GPU dokážou lépe využít paměť a paralelizaci právě při těchto hodnotách.

Zároveň byly do testování zařazeny i mezilehlé hodnoty jako 24, 48 nebo 56, které umožnily jemnější ladění optimální velikosti dávky.

Batch size	**acc@1** [%]
4	34.6
8	43.1
16	50.2
24	53.4
32	53.9
48	55.1
56	**56.3**
64	55.7

Tabulka 6.3: Výsledná přesnost modelu při různých hodnotách batch size

Z výsledků vyplývá, že větší batch size vede obecně k vyšší přesnosti, přičemž nejlepšího výsledku bylo dosaženo při hodnotě 56, která dosáhla přesnosti 56,3%. Hodnoty vyšší (např. 64) už nepřinášely významné zlepšení a mohly by navíc vyžadovat větší paměť GPU. Proto byla tato velikost zvolena jako optimální pro následující trénování modelu.

6.3 Datasety

V rámci doladění jazykového modelu bylo dalším klíčovým krokem zajistit kvalitní data, která odpovídají cílové úloze vyhledávání relevantních dokumentů v českém jazyce. Jelikož neexistuje nějaký jeden vhodný centralizovaný dataset, bylo nutné vytvořit vlastní kombinaci z aktuálně dostupných datasetů a případně je náležitě upravit do správné formy.

Primárním zaměřením při výběru datasetů byla jazyková relevance - zaměření na český jazyk. Následně bylo potřeba provést transformaci původních dat do formátu, který je vhodný pro doladění modelu v kontextu úlohy podobnosti mezi dotazem a dokumentem.

Experimentální fáze se soustředila na kombinaci různých datasetů s cílem maximalizovat přesnost modelu. Během tohoto procesu byly testovány různé poměry dat

a jejich vliv na výkonnost modelu při validaci. Ukázalo se, že použití více různých datových zdrojů přispívá k lepším výsledkům doladěného modelu.

Vzhledem k omezením dostupných datasetů a malého počtu trénovacích dvojic se přistoupilo k tvorbě vlastního datasetu (iDnes), který výrazně vylepšil výslednou přesnost modelu.

6.3.1 SQuAD

Prvním datasetem využitým v rámci experimentů byl český překlad datových sad SQuAD [31], který je široce používán pro trénování modelů typu question answering (QA). Cílem těchto datasetů je trénovat modely tak, aby dokázaly z textového kontextu odpovědět na položenou otázku.

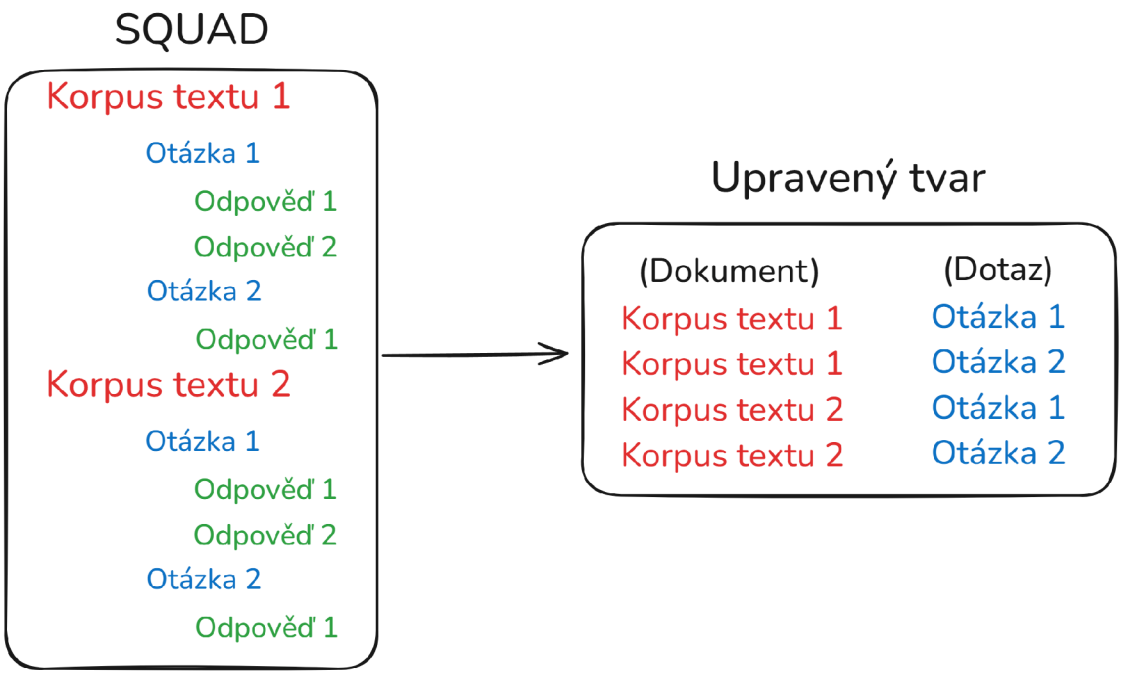
Datasety byly původně vytvořeny pro anglický jazyk ve dvou verzích – SQuAD v1.1 a SQuAD v2.0 [21]. Česká verze vznikla automatickým překladem těchto dat do češtiny v rámci snahy o tvorbu českých QA modelů a doladění modelů typu BERT a RoBERTa.

Standardní struktura SQuAD datasetu obsahuje následující komponenty:

- textový kontext – obvykle jeden nebo více odstavců textu
- otázky – jedna nebo více otázek vytvořených na základě daného kontextu
- odpovědi – správné odpovědi obsažené přímo v textu

Jelikož se však tato práce nezaměřuje na úlohu odpovídání na otázky, ale na vyhledávání relevantních dokumentů pro zadaný dotaz, bylo nezbytné dataset strukturovaný pro QA přizpůsobit. Proto byla provedena transformace původních dat do podoby dvojic dotaz – dokument, které odpovídají požadavkům modelu pro hodnocení relevance.

Transformace spočívala ve spojení každé otázky s odpovídajícím textovým kontextem, z něhož byla otázka odvozena. Výsledkem byly páry, kde otázka funguje jako dotaz a kontext jako dokument. Tento přístup umožňuje využít SQuAD dataset jako trénovací materiál pro model zaměřený na porozumění vztahu mezi dotazem a dokumentem.



Obrázek 6.1: Transformace datasetu SQuAD

Pro účely této práce byly zpracovány čtyři varianty dat: dev v1.1, dev v2.0, test v1.1, test v2.0. Sady označené jako dev byly použity pro doladění modelu. Sada test v1.1 byla využita jako testovací a sada test v2.0 sloužila jako validační. Po transformaci obsahovaly přibližně 37 tisíc dotaz – dokument párů. Testovací sady (test) pak sloužily k vyhodnocení výkonnosti modelu. Tento přístup umožnil využít i QA dataset pro trénování modelu pro hodnocení relevance.

6.3.2 DaReCzech

Jedním z významných datasetů použitých v této práci je DaReCzech [30], určený pro hodnocení relevance textů v českém jazyce. DaReCzech patří mezi nejrozsáhlejší dostupné české datasety pro úlohu hodnocení relevance, jelikož obsahuje více než 1,6 milionu anotovaných dvojic dotaz-dokument.

Dataset je rozdělen do čtyř částí podle účelu jejich použití:

- Train-big - obsahuje více než 1,4 milionu záznamů a slouží pro trénování modelů odhadujících relevanci textu.
- Train-small - zahrnuje přibližně 97 tisíc záznamů, původně určených pro trénování GBRT modelu, ale v praxi dobře posloužily i při doladění neuronových modelů.
- Dev - vývojová sada s 41 tisíci záznamy.
- Test - testovací sada s 64 tisíci záznamy.

Každá část datasetu je distribuována ve formátu TSV a obsahuje šest sloupců:

- ID - unikátní identifikátor
- query - uživatelův dotaz
- url - URL anotovaného dokumentu
- doc - reprezentace dokumentu, která obsahuje název, URL a extrahovaný hlavní obsah pomocí interního nástroje vyhledávače.
- title - název dokumentu
- label - míra relevance dokumentu k danému dotazu. Hodnota se pohybuje na škále od 0 (zcela irelevantní) do 1 (zcela relevantní), přičemž existuje pět diskrétních hodnot.

Pro potřeby této práce byl dataset předzpracován – z každého záznamu byly extrahovány pouze dva sloupce: query a doc, které slouží jako vstupní páry.

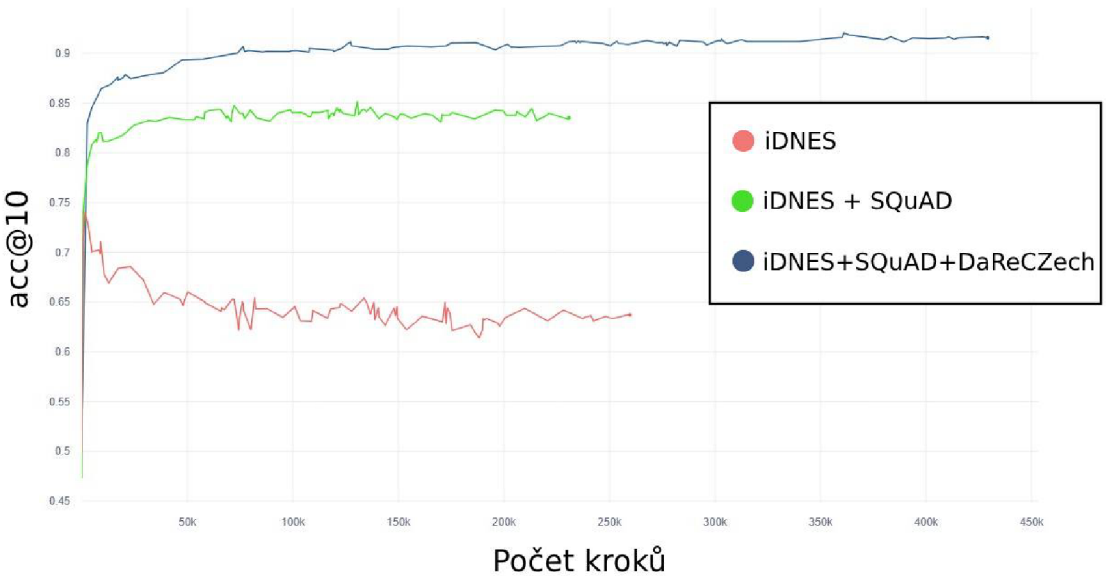
V počátečních experimentech byla využita především rozsáhlá část Train-big. Nicméně výsledky ukázaly, že použití této části pro dotrénování modelu vedlo k méně uspokojivým výsledkům, pravděpodobně z důvodu větší míry šumu nebo rozmanitosti dat. Naopak menší trénovací sada Train-small vykázala lepší vlastnosti a vedla k vyšší přesnosti modelu při následném vyhodnocení.

Dále se testovalo, zda by filtrování záznamů na základě vyšších hodnot relevance (label) mohlo přispět ke zlepšení přesnosti modelu. Byly tedy vybrány pouze dokumenty označené jako převážně relevantní. Nicméně tato strategie neprokázala výrazný přínos – výsledná přesnost modelu zůstala víceméně nezměněná.

6.3.3 iDNES

Při práci s původními dvěma datasety (SQuAD a DaReCZech) se ukázalo, že množství dostupných trénovacích párů nebylo dostatečné pro kvalitní doladění jazykového modelu. Z toho důvodu bylo potřeba přistoupit k vytvoření vlastního datasetu, který by poskytl větší variabilitu a množství trénovacích vzorků. Jako vhodný zdroj pro získání dat se ukázal novinkový portál iDnes.cz, z něhož byly extrahovány články společně s jejich nadpisy a shrnutími. Detailní popis způsobu extrakce a předzpracování dat je uveden v následující kapitole.

Výsledkem této činnosti byl vlastní dataset složený z 500 000 párů ve formátu nadpis – anotace (shrnutí článku). Tento dataset následně posloužil jako základ pro další sérii experimentů zaměřených na doladění modelu. V rámci těchto experimentů byly zkoumány různé kombinace datasetů a jejich vliv na výslednou přesnost modelu.



Obrázek 6.2: Doladění pomocí různých kombinací datasetů - acc@10

Na obrázku výše jsou znázorněny průběhy přesnosti metriky acc@10 v závislosti na počtu trénovacích kroků. Graf obsahuje tři křivky, z nichž každá reprezentuje jiný experiment s odlišnou kombinací trénovacích dat:

- Červená křivka - znázorňuje výsledky modelu trénovaného pouze na datasetu iDnes. Zpočátku dochází k rychlému nárůstu přesnosti, model se velmi rychle učí základní vztahy mezi nadpisem a shrnutím. Následně ale dochází ke stagnaci a mírnému poklesu výkonu, přičemž křivka začíná oscilovat kolem určité hodnoty bez dalšího zlepšování. Tento jev může být způsoben omezenou jazykovou rozmanitostí nebo doménovou specifikací článků portálu iDnes.

- Zelená křivka - ukazuje experiment, ve kterém byl dataset iDnes zkombinován s datasetem SQuAD. Tato kombinace přinesla znatelné zlepšení – model se učil stabilněji a přesnost dosahovala vyšších hodnot než v předchozím případě. Oscilace výsledků byly menší a model si dokázal udržet stabilní výkon v delším trénovacím horizontu.

- Modrá křivka - představuje finální a nejúspěšnější experiment, kdy byl ke kombinaci iDnes + SQuAD přidán ještě dataset DaReCZech. Tento krok vedl k nejlepším výsledkům – přesnost acc@10 se postupně vyšplhala až k hodnotě 92 %, přičemž trend křivky nadále mírně stoupal. Výsledný model tak překonal i vyhodnocený opensource model multilingual-e5-base.

Na základě těchto výsledků je patrné, že přidání doménově různorodých dat má pozitivní vliv na výkonnost jazykového modelu. Vytvořený dataset z iDnes poskytl silný základ, ale teprve v kombinaci s dalšími datasety se podařilo dosáhnout výrazného zlepšení modelu.

Tvorba datasetu

Pro účely vytvoření vlastního datasetu zpravodajských článků z portálu iDnes.cz byl implementován webový scraper v jazyce Python, který využíval dvě klíčové knihovny: `requests` a `BeautifulSoup4` (pro HTML parsování).

Prvním problémem, který bylo nutné vyřešit, byla samotná dostupnost obsahu článků. Při standardním odesílání HTTP požadavků pomocí knihovny `requests` server vracel pouze omezený obsah stránky – konkrétně okno s výzvou k přijetí cookies. Aby bylo možné získat kompletní HTML kód s obsahem článku, bylo nezbytné manuálně nastavit hlavičky požadavku tak, aby simulovaly běžného uživatele prohlížeče, který již cookies přijal. Teprve poté bylo možné HTML úspěšně získat a dále zpracovávat.

Samotné stahování článků probíhalo napříč několika tematickými kategoriemi portálu iDnes (např. Domácí, Zahraničí, Kultura atd.). Každá kategorie má svou vlastní stránku s výpisem článků, která je dále stránkovaná pomocí číselného parametru v URL. Skript tak opakovaně generoval URL adresy s postupně rostoucím číslem stránky, dokud byly k dispozici nové články. Po dosažení konce jedné kategorie skript automaticky přešel na další.

Z každého článku byla extrahována dvojice informací – nadpis článku a anotace, tedy krátké shrnutí uvedené zpravidla pod nadpisem. Extrakci zajišťovala funkce, která po načtení HTML ověřila přítomnost obou prvků. Pokud byly dostupné, daný pár byl uložen jako jedna trénovací instance. Články, u nichž chyběl buď nadpis, nebo anotace, byly přeskočeny.

Celý proces stahování dat trval přibližně osm hodin a vygeneroval sadu o velikosti přes 500 000 párů nadpis–shrnutí. Po dokončení sběru následovala fáze analýzy a čištění dat. I přes předběžnou kontrolu se v datasetu nacházely duplicitní záznamy a záznamy s prázdnými políčky. Po jejich odstranění bylo z datasetu vyřazeno přibližně 15 000 párů.

7 Tvorba aplikace

V rámci této práce byla vytvořena aplikace, která implementuje techniku RAG. Hlavním cílem aplikace bylo umožnit efektivní práci s textovými dokumenty ve formátu PDF, provést jejich předzpracování, převést jejich obsah do vektorové reprezentace a umožnit uživateli zadávat dotazy, na které odpovídá jazykový model na základě relevantního kontextu.

7.1 Předzpracování dokumentů

Prvním krokem celého procesu je extrakce čistého textu z PDF dokumentů. K tomuto účelu byla využita knihovna pdfplumber, která umožňuje přesnou extrakci textu z jednotlivých stránek. Získaný text je následně rozdělen do menších částí o velikosti 800 znaků s překrytím 200 znaků, a to za pomoci nástroje `RecursiveCharacterTextSplitter` z knihovny `Langchain`. Tento přístup zajišťuje, že nedojde ke ztrátě důležitého kontextu na hranicích mezi jednotlivými částmi.

7.2 Vektorizace a ukládání do databáze

Rozdělené části textu jsou dále převedeny do vektorové podoby pomocí vlastního jazykového modelu. Pro tuto vektorizaci byla zvolena knihovna `SentenceTransformers`, která umožňuje přímé a flexibilní použití předtrénovaných i vlastních modelů.

Namísto použití nástrojů z knihovny `Langchain` pro zjednodušení implementace, jako je např. wrapper `HuggingFaceEmbeddings` nebo integrace `ChromaDB` pomocí `langchain vectorstores`, byl celý postup implementována manuálně. To zahrnuje:

- Vytváření nové kolekce ve vektorové databázi `ChromaDB`
- Převod jednotlivých chunků na vektory pomocí modelu ze `SentenceTransformers`
- Ruční ukládání vektorů spolu s metadaty (např. název souboru, cesta) do databáze.

Testování ukázalo, že implementace využívající nástroje knihovny `Langchain`, jako je `HuggingFaceEmbeddings` a využití ChromaDB pomocí `langchain`

`vectorstores`, dosahovala horších výsledků při vyhledávání relevantních částí dokumentů než ručně implementované řešení. Vlastní přístup umožnil přesnější kontrolu nad jednotlivými kroky a výrazně zlepšil kvalitu vyhledaného kontextu.

7.3 Vyhledávání a generování odpovědí

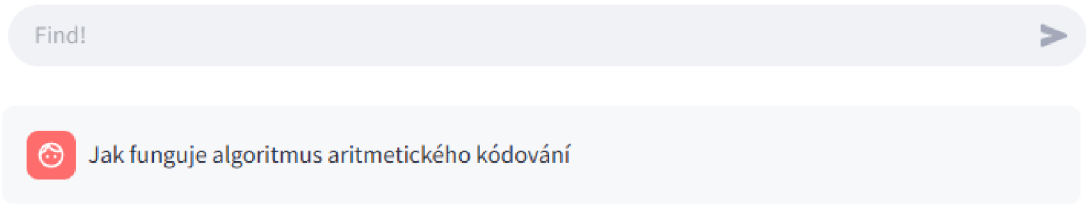
Po zadání dotazu uživatelem je dotaz převeden na vektor, který je následně porovnán s uloženými vektory ve `ChromaDB`. Na základě kosinové podobnosti jsou vybrány nejbližší části dokumentů, které slouží jako kontext pro generaci odpovědi.

K vytvoření odpovědi je využíván lokálně hostovaný jazykový model `llama3.1:8b`, spuštěný pomocí nástroje `Ollama`. Přestože existují výkonnější modely, které jsou veřejně dostupné zdarma, byl záměrně zvolen menší model, kvůli možnosti nasazení modelu na vlastní GPU a možnosti sledovat, jak velký vliv má kvalitní kontext a správně strukturovaný dotaz na kvalitu výstupu. V rámci experimentů se testovali i větší modely jako llama3.3:70B.

7.4 Uživatelské rozhraní

Pro realizaci jednoduchého, ale funkčního uživatelského rozhraní byla zvolena knihovna `Streamlit`. Ta umožnila rychlé sestavení webové aplikace, ve které může uživatel:

- Položit dotaz



Obrázek 7.1: Uživatelské rozhraní - položení dotazu

- Zobrazit navrácené části dokumentů (kontext použitý pro odpověď)
- Získat výstupní odpověď generovanou modelem

Algoritmus aritmetického kódování je technika komprese dat, která umožňuje efektivně zkomprimovat datové soubory. Funkční princip aritmetického kódování spočívá v tom, že se při zpracování vstupních dat generuje libovolné reálné číslo z intervalu <0,1), které bude použito jako kód pro další znak vstupních dat.

Fáze algoritmu aritmetického kódování:

1. **Pravděpodobnosti výskytu jednotlivých znaků**: Předpokládejme, že máme datové soubory obsahující znaky A, B, C a D. Pro každý znak je určena pravděpodobnost jeho výskytu (například P(A) = 0,4, P(B) = 0,2, P(C) = 0,3, P(D) = 0,1).
2. **Rozdělení intervalu <0,1)**: Interval <0,1) je rozdělen do podintervalů tak, aby odpovídal kumulativním pravděpodobnostem jednotlivých znaků (například <0, Q1), <Q1, Q2), <Q2, Q3), <Q3, Q4), kde Q0 = 0 a QN = 1).
3. **Generování kódu**: Při zpracování vstupních dat se pro každý znak generuje libovolné reálné číslo z příslušného podintervalu. Toto číslo bude použito jako kód pro další znak vstupních dat.
4. **Speciální znak nebo délka původní posloupnosti znaků**: Na konec kódované zprávy dáme speciální znak, jinak při dekódování není možné určit konec datového toku, nebo uložíme délku původní posloupnosti znaků.

Algoritmus aritmetického kódování je efektivní technika komprese dat, která umožňuje efektivně zkomprimovat datové soubory.

Obrázek 7.2: Uživatelské rozhraní - odpověď LLM

Závěr

Tato bakalářská práce se zaměřila na tvorbu vlastního jazykového modelu pro vektorizaci českého textu. Na základě úvodní rešerše existujících architektur modelů a následného experimentálního vyhodnocení byla jako nejvhodnější zvolena architektura xlm-RoBERTa-base. Tato architektura byla následně využita jako výchozí pro vlastní dotrénování a přizpůsobení modelu specifickým potřebám českého jazyka.

Pro dosažení co nejlepších výsledků byl sestaven trénovací dataset kombinující více zdrojů. Využity byly jak existující datasety – přeložený SQuAD a český DaReCzech – tak i nový dataset iDnes, který byl vytvořen metodou web scrapingu článků z internetového portálu iDnes. Výsledkem bylo rozšíření trénovacích dat o více než 500 000 dvojic, čímž byl významně zvýšen objem dostupných dat vhodných pro dotrénování modelu.

Nastavení optimálních hyperparametrů probíhalo pomocí experimentálního vyhodnocování. Bylo provedeno několik dávkových úloh s různými kombinacemi parametrů, na jejichž základě byly identifikovány hodnoty vedoucí k nejlepším výsledkům. Samotný proces dotrénování modelu, realizovaný za využití výkonných výpočetních prostředků Metracentra Cesnet, trval celkem 168 hodin.

Výsledný model byl validován na stejných datech jako byly vyhodnocovány aktuálně veřejně dostupné modely, přičemž největší důraz byl kladen na přesnost v rámci 10 vrácených dokumentů (acc@10). V této metrice model překonal všechny ostatní testované varianty.

Následně byla vytvořené praktická chatbotová aplikace[1] využívající techniku RAG pro demonstraci vytvořeného modelu při vektorizaci dokumentů, které si uživatel nahraje. Pro generování odpovědí se využívaly různé verze velkých jazykových modelů, konkrétně varianty modelů Llama a Groq. Tato aplikace ukazuje uživateli možnosti praktického použití vytvořeného modelu.

[1] https://owncloud.cesnet.cz/index.php/s/l4NmXosuAH3zNTH

Použitá literatura

[1] MIKOLOV, Tomas et al. *Efficient Estimation of Word Representations in Vector Space* [online]. 2013. [cit. 2013-09-07]. Dostupné z: https://arxiv.org/abs/1301.3781.

[2] MIKOLOV, Tomas et al. *Distributed Representations of Words and Phrases and their Compositionality*. 2013. Dostupné z arXiv: 1310.4546 [cs.CL].

[3] VASWANI, Ashish et al. *Attention Is All You Need* [online]. 2023. [cit. 2023-08-02]. Dostupné z: https://arxiv.org/abs/1706.03762.

[4] SCHMIDT, Robin M. *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview*. 2019. Dostupné z arXiv: 1912.05911 [cs.LG].

[5] DEVLIN, Jacob et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* [online]. 2019. [cit. 2019-05-24]. Dostupné z: https://arxiv.org/abs/1810.04805.

[6] BROWN, Tom B. et al. *Language Models are Few-Shot Learners*. 2020. Dostupné z arXiv: 2005.14165 [cs.CL].

[7] KALYAN, Katikapalli Subramanyam. *A Survey of GPT-3 Family Large Language Models Including ChatGPT and GPT-4*. 2023. Dostupné z arXiv: 2310.12321 [cs.CL].

[8] NIE, Zhijie et al. *When Text Embedding Meets Large Language Model: A Comprehensive Survey*. 2025. Dostupné z arXiv: 2412.09165 [cs.CL].

[9] ŠKORIĆ, Mihailo. *Text vectorization via transformer-based language models and n-gram perplexities*. 2023. Dostupné z arXiv: 2307.09255 [cs.CL].

[10] WANG, Liang et al. *Improving Text Embeddings with Large Language Models*. 2024. Dostupné z arXiv: 2401.00368 [cs.CL].

[11] BEDNÁŘ, Jiří et al. *Some Like It Small: Czech Semantic Embedding Models for Industry Applications* [online]. 2023. [cit. 2023-11-23]. Dostupné z: https://arxiv.org/abs/2311.13921.

[12] XIAO, Shitao et al. *RetroMAE: Pre-Training Retrieval-oriented Language Models Via Masked Auto-Encoder*. 2022. Dostupné z arXiv: 2205.12035 [cs.CL].

[13] GOOGLE. *The WordPiece Algorithm in Open Source BERT*. 2018. Dostupné také z: https://github.com/%20google-research/bert/blob/master/%20tokenization.py#L335-L358.

[14] LOSHCHILOV, Ilya a Frank HUTTER. *Decoupled Weight Decay Regularization*. 2019. Dostupné z arXiv: 1711.05101 [cs.LG].

[15] REIMERS, Nils a Iryna GUREVYCH. *Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation*. 2020. Dostupné z arXiv: 2004.09813 [cs.CL].

[16] WANG, Liang et al. *Multilingual E5 Text Embeddings: A Technical Report* [online]. 2024. [cit. 2024-02-08]. Dostupné z: https://arxiv.org/abs/2402.05672.

[17] WANG, Liang et al. [online]. 2024. [cit. 2024-02-22]. Dostupné z: https://arxiv.org/abs/2212.03533.

[18] LIU, Yinhan et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach* [online]. 2019. [cit. 2019-07-26]. Dostupné z: https://arxiv.org/abs/1907.11692.

[19] OORD, Aaron van den, Yazhe LI a Oriol VINYALS. *Representation Learning with Contrastive Predictive Coding*. 2019. Dostupné z arXiv: 1807.03748 [cs.LG].

[20] BAJAJ, Payal et al. *MS MARCO: A Human Generated MAchine Reading COmprehension Dataset*. 2018. Dostupné z arXiv: 1611.09268 [cs.CL].

[21] RAJPURKAR, Pranav et al. *SQuAD: 100,000+ Questions for Machine Comprehension of Text*. 2016. Dostupné z arXiv: 1606.05250 [cs.CL].

[22] ZHANG, Xinyu et al. *Towards Best Practices for Training Multilingual Dense Retrieval Models*. 2022. Dostupné z arXiv: 2204.02363 [cs.IR].

[23] FENG, Fangxiaoyu et al. *Language-agnostic BERT Sentence Embedding*. 2022. Dostupné z arXiv: 2007.01852 [cs.CL].

[24] PETERS, Matthew E. et al. *Deep contextualized word representations* [online]. 2018. [cit. 2018-03-22]. Dostupné z: https://arxiv.org/abs/1802.05365.

[25] SENNRICH, Rico, Barry HADDOW a Alexandra BIRCH. *Neural Machine Translation of Rare Words with Subword Units*. 2016. Dostupné z arXiv: 1508.07909 [cs.CL].

[26] KUDO, Taku a John RICHARDSON. *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*. 2018. Dostupné z arXiv: 1808.06226 [cs.CL].

[27] ALI, Mehdi et al. *Tokenizer Choice For LLM Training: Negligible or Crucial?* [online]. [cit. 2024-03-17]. Dostupné z: https://arxiv.org/abs/2310.08754.

[28] CORDONNIER, Jean-Baptiste, Andreas LOUKAS a Martin JAGGI. *Multi--Head Attention: Collaborate Instead of Concatenate*. 2021. Dostupné z arXiv: 2006.16362 [cs.LG].

[29] CONNEAU, Alexis et al. *Unsupervised Cross-lingual Representation Learning at Scale*. 2020. Dostupné z arXiv: 1911.02116 [cs.CL].

[30] KOCIÁN, Matěj et al. Siamese BERT-Based Model for Web Search Relevance Ranking Evaluated on a New Czech Dataset. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2022, roč. 36, č. 11, s. 12369–12377. Dostupné z DOI: 10.1609/aaai.v36i11.21502.

[31] MACKOVÁ, Kateřina a Milan STRAKA. *Reading Comprehension in Czech via Machine Translation and Cross-lingual Transfer*. 2020. Dostupné z arXiv: 2007.01667 [cs.CL].

[32] REIMERS, Nils a Iryna GUREVYCH. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2019. Dostupné také z: http://arxiv.org/abs/1908.10084.

[33] ANSEL, Jason et al. *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. ACM, 2024-04. Dostupné z DOI: 10.1145/3620665.3640366.

[34] CLARK, Kevin et al. *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*. 2020. Dostupné z arXiv: 2003.10555 [cs.CL].

[35] WOLF, Thomas et al. *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. 2020. Dostupné z arXiv: 1910.03771 [cs.CL].

[36] SIDO, Jakub et al. *Czert – Czech BERT-like Model for Language Representation*. 2021. Dostupné z arXiv: 2103.13031 [cs.CL].