



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KOMUNIKAČNÍ AGENT PRO INFORMACE O BRNĚ

BRNO COMMUNICATION AGENT

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. KATEŘINA NEPRAŠOVÁ

VEDOUCÍ PRÁCE

SUPERVISOR

doc. RNDr. PAVEL SMRŽ, Ph.D.

BRNO 2024

Zadání diplomové práce



156444

Ústav: Ústav počítačové grafiky a multimédií (UPGM)
Studentka: **Neprašová Kateřina, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Počítačové vidění
Název: **Komunikační agent pro informace o Brně**
Kategorie: Zpracování řeči a přirozeného jazyka
Akademický rok: 2023/24

Zadání:

1. Seznamte se s existujícími řešeními komunikačních agentů a nástroji pro jejich vytváření.
2. Shromážděte data znalostní báze, ze které bude agent čerpat.
3. Navrhněte a implementujte systém pro komunikaci s uživatelem v oblasti informací o Brně, aktuálních událostech a dalších informacích.
4. Vyhodnoťte vytvořený systém v testu s prvotními uživateli.
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- dle doporučení vedoucího

Při obhajobě semestrální části projektu je požadováno:

- funkční prototyp řešení

Podrobné závazné pokyny pro vypracování práce viz <https://www.fit.vut.cz/study/theses/>

Vedoucí práce: **Smrž Pavel, doc. RNDr., Ph.D.**
Vedoucí ústavu: Černocký Jan, prof. Dr. Ing.
Datum zadání: 1.11.2023
Termín pro odevzdání: 17.5.2024
Datum schválení: 21.12.2023

Abstrakt

Tato diplomová práce se zaměřuje na doménově specifické komunikační agenty, cílem je vytvořit funkčního komunikačního agenta pro turisty i místní obyvatele Brna, poskytujícího relevantní a aktuální informace podle individuálních potřeb uživatelů. Popisuje velké jazykové modely, analyzuje existující technologie pro doménově specifické komunikační agenty a jejich tvorbu. Soustředí se na vytváření znalostní báze a implementaci efektivního dialogového rozhraní s využitím generování s rozšířeným vyhledáváním (RAG), přičemž srovnává vybrané jazykové modely na různých úlohách.

Abstract

This thesis focuses on domain-specific communication agents, the aim is to create a functional communication agent for both tourists and locals in Brno, providing relevant and up-to-date information according to individual user needs. It describes large language models, analyses existing technologies for domain-specific communication agents and their construction. It focuses on the creation of a knowledge base and the implementation of an efficient dialogue interface using Retrieval-Augmented Generation (RAG), while comparing selected language models on different tasks.

Klíčová slova

Zpracování přirozeného jazyka, Velké jazykové modely, Komunikační agent, Město Brno, Inteligentní asistent, Generování s rozšířeným vyhledáváním

Keywords

Natural Language Processing, Large Language Models, Communication Agent, City of Brno, Intelligent Assistant, Retrieval-Augmented Generation

Citace

NEPRAŠOVÁ, Kateřina. *Komunikační agent pro informace o Brně*. Brno, 2024. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. RNDr. Pavel Smrž, Ph.D.

Komunikační agent pro informace o Brně

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně pod vedením pana doc. RNDr. Pavla Šmrže, Ph.D. Uvedla jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpala.

.....
Kateřina Neprašová
16. května 2024

Poděkování

Děkuji všem, kteří mi při tvorbě této práce pomáhali, ať už jakýmkoliv způsobem. Zejména mému vedoucímu, rodině, přátelům a kolegům. Mé poděkování patří i všem testovacím respondentům.

Obsah

| | | |
|----------|--|-----------|
| 1 | Úvod | 3 |
| 2 | Komunikační agenti | 5 |
| 2.1 | Velké jazykové modely | 5 |
| 2.2 | Doménově specifíční komunikační agenti | 13 |
| 3 | Generování s rozšířeným vyhledáváním | 17 |
| 3.1 | Metody | 18 |
| 3.2 | Pokročilé metody | 19 |
| 3.3 | Evaluace | 25 |
| 4 | Návrh a implementace | 26 |
| 4.1 | Získávání dat | 27 |
| 4.2 | SQL Databáze | 34 |
| 4.3 | Agent | 36 |
| 4.4 | Používané velké jazykové modely | 39 |
| 4.5 | Klasifikace | 40 |
| 4.6 | Zpracování dat | 42 |
| 4.7 | Vektorová reprezentace dat | 47 |
| 4.8 | Konstrukce odpovědí | 52 |
| 4.9 | Uživatelské rozhraní | 54 |
| 4.10 | Testování a evaluace | 56 |
| 5 | Závěr | 62 |
| | Literatura | 64 |
| A | Vzorové použití aplikace | 67 |
| B | Vzorové dotazy | 69 |
| C | Zdroje informací pro databázi | 70 |
| D | Manuál | 71 |
| E | Plakát | 73 |

Seznam obrázků

| | | |
|------|---|----|
| 2.1 | Všestrannost využití LLM. | 6 |
| 2.2 | Vztah vektorů v embedding prostoru. | 9 |
| 2.3 | Vrstvená architektura transformerů. | 11 |
| 2.4 | Transformer – kodér a dekodér. | 11 |
| 2.5 | Architektura transformeru podle [38]. | 12 |
| 2.6 | Schéma základního konceptu jazykových modelů. | 14 |
| 3.1 | Schéma základního konceptu RAG. | 17 |
| 3.2 | Původní přístup dle [20]. | 18 |
| 3.3 | Pokročilá architektura RAG podle [9]. | 20 |
| 4.1 | Zjednodušená architektura systému. | 26 |
| 4.2 | Detailní architektura systému. | 27 |
| 4.3 | Schéma procesu získávání dat. | 28 |
| 4.4 | Cílová část obsahu ze stránky z moravska-galerie.cz. | 32 |
| 4.5 | Box s popisky odkazujícími na jiné stránky z konce stránky z gotobrno.cz (nahore) a konce stránky z moravska-galerie.cz (dole). | 33 |
| 4.6 | Schéma SQL databáze. | 35 |
| 4.7 | Schéma generování odpovědi. | 53 |
| 4.8 | Testovací uživatelské rozhraní. | 55 |
| 4.9 | Produkční uživatelské rozhraní. | 55 |
| 4.10 | Hlavní komponenta formuláře pro hodnocení respondenty. | 59 |
| 4.11 | Graf hodnocení modelů respondenty při zadání dotazu v angličtině. | 61 |
| E.1 | Náhled plakátu prezentujícího cíle a výsledky této práce. | 73 |

Kapitola 1

Úvod

V poslední době se umělá inteligence stává ve společnosti velmi častým tématem, a to právě v důsledku vývoje a inovací v oblastech strojového učení. Se zveřejněním volně dostupného konverzačního agenta ChatGPT společností OpenAI získal pozornost i pokrok ve vývoji velkých jazykových modelů. Díky tomu si lidé začínají více uvědomovat, že se strojové učení stává součástí čím dál více systémů, na které se v každodenním životě spoléhají.

Technologické inovace a inteligentní systémy dnes usnadňují získávání informací a pochopení okolního světa. A právě v reakci na pokroky ve vývoji velkých jazykových modelů vzniká i tato diplomová práce, jejímž hlavním cílem je vytvořit komunikačního agenta pro poskytování informací o městě Brně s využitím těchto modelů. Zaměřuje na vytvoření inteligentního asistenta, který bude schopen odpovídat na otázky spojené s Brnem a poskytovat uživatelům relevantní a aktuální informace v souladu s jejich konkrétními potřebami.

Následující kapitola shrnuje nástroje a technologie pro vytváření komunikačních agentů, analyzuje současný stav těchto technologií a vyhodnocuje jejich schopnosti a omezení. V dialogových systémech se běžně využívají velké jazykové modely. Ty jsou však trénovány se zpožděním – o čemkoliv, co se stane později, než je v tréninkových datech, nemá model informace a může si vymýšlet, neboli *halucinovat*. Zároveň jim mohou scházet specializované informace. Řešením tohoto problému jsou doménově specifictí agenti. Jejich možností, jako je fine-tuning, adapter nebo přidání dokumentů, se tato kapitola také věnuje.

Tato práce se soustředí na řešení problému chybějících znalostí propojením jazykového modelu s externími daty, známé jako generování s rozšířeným vyhledáváním (RAG, Retrieval-Augmented Generation). V kapitole 3 jsou popsány základní i pokročilé metody při návrhu těchto systémů, možnosti jejich optimalizace a evaluaci.

Kapitola 4 se věnuje samotnému návrhu výsledné aplikace. Zde je popsáno zpracování dat a tvorba znalostní báze, kterou agent využívá k poskytování informací o Brně. Tato fáze obsahuje sběr dat z různých zdrojů a jejich strukturování za účelem efektivního využití agentem. Dále se tato kapitola zabývá návrhem a implementací systému pro komunikaci s uživatelem. Zde se soustředí na rozhraní mezi uživatelem a agentem, s důrazem na intuitivnost a efektivitu komunikace. Implementace agenta staví na již zmíněném RAG systému, což umožňuje kombinovat schopnosti vyhledávání a generování informací. Pro posouzení úspěšnosti systému jsou provedeny testy s prvotními uživateli, kteří hodnotí přesnost a relevanci odpovědí, uživatelskou přívětivost a celkový výkon agenta. Zároveň se zde nachází srovnání vybraných velkých jazykových modelů na různých úlohách.

Výsledkem práce je funkční komunikační agent, který může být využit jak turisty hledajícími informace o Brně, tak i místními obyvateli zajímajícími se o kulturní události nebo jiné informace týkající se jejich města. V závislosti na schopnostech jazykového modelu dokáže komunikovat v angličtině i češtině. Tento agent čerpá informace z vektorové databáze obsahující nejen informace o místech, administrativě, osobách a událostech spojených s městem Brnem, ale také informace o aktuálních kulturních akcích. Zároveň je schopen posoudit relevantnost získaných dat a vyhledávat na internetu, jsou-li potřeba další informace.

Kapitola 2

Komunikační agenti

V posledních letech jsou na internetu čím dál více používáni tzv. komunikační agenti, kteří usnadňují interakce, poskytují informace a podporují různé online aktivity. Dle *Hugeta* [12] se dá komunikační agent definovat jako softwarová entita, která komunikuje s jinými agenty nebo uživateli prostřednictvím komunikačního kanálu a vyměňuje si zprávy za účelem dosažení konkrétních cílů nebo úkolů. Tito agenti jsou navrženi tak, aby odesílali a přijímali informace, odpovídali na dotazy a zapojovali se do dialogů s cílem usnadnit efektivní komunikaci a spolupráci.

Tradičně se tyto systémy spoléhaly na předem definované skripty, systémy založené na pravidlech, nebo omezené modely strojového učení, aby porozuměly dotazům uživatelů a reagovaly na ně. S příchodem velkých jazykových modelů se však prostředí komunikačních agentů výrazně proměnilo – způsobily totiž revoluci ve schopnostech těchto agentů tím, že jim umožnily zpracovávat a vytvářet text s porozuměním kontextu a s plynulostí srovnatelnou s lidským projevem, čímž se zvýšila jejich efektivita při přesnějším a přirozenějším chápání a odpovídání na dotazy uživatelů, jak uvádí *Raroque* [31].

A protože jsou tyto modely klíčovou součástí tohoto projektu, věnuji se jim více v následující části této kapitoly. Následně se v této kapitole věnuji možnostem návrhů doménově specifických agentů, jejich vlastnostem a srovnání.

2.1 Velké jazykové modely

Velké jazykové modely (Large Language Models, LLM) představují průlomový krok v rozvoji schopnosti umělé inteligence porozumět lidskému jazyku a generovat jej. Základem těchto modelů jsou principy strojového učení, které analyzují a interpretují složitosti jazyka pomocí rozsáhlých hierarchicky uspořádaných neuronových sítí.

Pro tuto práci jsou velké jazykové modely důležité svou schopností komunikace, jsou nejen hlavní součástí systému komunikujícího s uživatelem, ale také se starají o sémanticky správné zpracování vstupních dat.

Velké jazykové modely mají různé využití v různých oblastech a odvětvích díky své schopnosti porozumět a generovat text až nerozeznatelně podobný textům vytvořeným člověkem. Mezi klíčové aplikace velkých jazykových modelů patří úlohy zpracování přirozeného

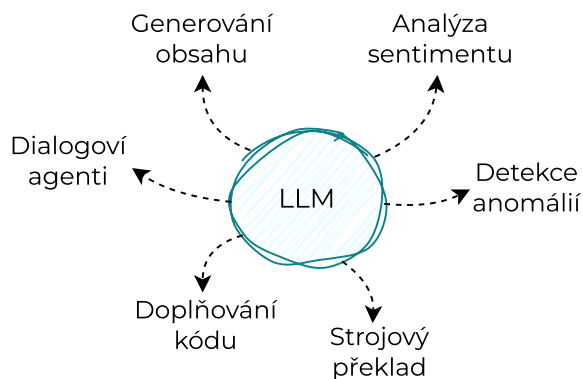
jazyka (Natural Language Processing, NLP), jako je klasifikace textu, rozpoznávání pojmenovaných entit, analýza sentimentu a generování jazyka. Jak zmiňuje *Pathak* [29], dokážou porozumět lidskému jazyku, interpretovat jej a manipulovat s ním, což usnadňuje efektivní komunikaci mezi lidmi a stroji. Všestranné využití reprezentuje diagram na obrázku 2.1.

Všestranným případem použití jsou dialogoví agenti, chatboti nebo virtuální asistenti, což je právě i případem této práce. Jazykové modely těmto systémům umožňují porozumět dotazům uživatelů a reagovat podobně jako člověk. Mohou generovat odpovědi, které jsou nejen kontextově relevantní, ale také zachovávají koherenci v průběhu celého dialogu. Jak uvádí *Raroque* [31], dialogoví agenti nacházejí uplatnění v zákaznickém servisu, zdravotnictví, vzdělávání a různých dalších oblastech, kde je výhodná interakce podobná interakci s lidmi.

Velké jazykové modely hrají podle *Pathaka* [29] také zásadní roli v systémech strojového překladu, protože zvyšují přesnost a plynulost překládaných textů. Tradiční přístup ke strojovému překladu zahrnoval systémy založené na pravidlech, které vyžadovaly mnoho lidských vstupů k vytvoření složitých souborů jazykových pravidel. Vývoj jazykových překladových modelů schopných zachytit jemné nuance a kontext vede k přesnějším a kontextuálně vhodnějším překladům.

Jak již bylo zmíněno, velké jazykové modely se osvědčily při generování různých forem textů, využívají se i pro generování obsahu a automatizaci v činnostech jako je psaní článků, tvůrčí psaní a syntéza dat. Mohou vytvářet souvislý a kontextově relevantní obsah a automatizovat generování velkých objemů textu pro různé případy použití.

Používají se rovněž pro analýzu sentimentu, což je technika pro identifikaci a extrakci subjektivních informací z textu, jako jsou emoce, názory a postoje. *Keary* [19] zmiňuje, že nejčastěji se uplatňuje při analýze zpětné vazby od zákazníků, při managementu obchodních značek a monitorování sociálních médií a při průzkumu trhu, což organizacím umožňuje snadněji získat přehled o veřejném mínění a sentimentu spotřebitelů.



Obrázek 2.1: Všestrannost využití LLM.

Další využití je podle *Kearyho* [19] také automatické dokončování kódu v integrovaných vývojových prostředích (IDE). Lze je použít i jako alternativní vyhledávací nástroj, či je využít pro analýzu malware nebo detekci a prevenci kybernetických útoků. Rozpoznávají totiž vzory, které naznačují škodlivý kybernetický útok, a mohou generovat upozornění. Neposlední uplatnění najdou v systémech pro odhalování podvodů, které identifikují vzory svědčící o podvodných aktivitách.

2.1.1 Typy jazykových modelů

Existuje více druhů LLM a celá řada způsobů, jak je dělit. Tato podkapitola se krátce věnuje dělení jazykových modelů podle jejich architektury, jak je popisuje *Mostafa* [13]. Každý s jedinečnými vlastnostmi a aplikacemi, od modelů založených na auto-enkodéru, přes sequence-to-sequence modely, modely založené na transformerech, modely založené na rekurzivních neuronových sítích po hierarchické modely.

Modely založené na auto-enkodéru

Velké jazykové modely založené na architektuře auto-enkodéru se běžně skládají z kodéru a dekodéru a jejich funkcionalita je následující:

1. Kodér přijímá vstupní data, například textové sekvence, a komprimuje je do méně rozměrné reprezentace.
2. Dekodér tuto komprimovanou reprezentaci pak použije k rekonstrukci původních vstupních dat.

V kontextu rozsáhlých jazykových modelů se tato architektura používá k efektivnímu zachycení a reprezentaci složitých jazykových vzorů a struktur – auto-enkodéry jsou schopné zachytit bohatství výrazů přirozeného jazyka, kontext a sémantický význam v rámci kódovaných reprezentací. Jejich nejpoužívanější aplikací je in-context auto-enkodér.

Komprimací vstupních dat do méně rozměrné formy a jejich následnou rekonstrukcí se tyto modely dokáží naučit a reprezentovat složité jazykové rysy a vztahy. Kvůli komprimaci však mohou mít problémy se zachováním jemných detailů původního textu, což může ovlivnit přesnost generovaného obsahu.

Sequence-to-sequence modely

Velké jazykové modely založené na Sequence-to-sequence principu se vyznačují schopností překládat nebo generovat sekvence tokenů proměnlivé délky. Též se skládají z kodéru a dekodéru:

1. Kodér přijímá vstupní sekvenci a zpracovává ji do kontextového vektoru pevné velikosti, který zachycuje podstatné informace ze vstupní sekvence.
2. Dekodér použije tento vektor k předpovědi dalšího tokenu ve výstupní sekvenci a tento proces se opakuje, dokud není vygenerována celá výstupní sekvence.

Jsou efektivní v úlohách jako je strojový překlad, sumarizace textu a v poslední době i generování odpovědí v přirozeném jazyce. Mohou se však setkat s problémy při práci se složitějšími závislostmi v textu, což může ovlivnit souvislost překladů a shrnutí.

Transformer modely výrazně zvýšily jejich efektivitu tím, že umožnily paralelní zpracování vstupních sekvencí a zavedly mechanismy pozornosti, které umožňují modelu zaměřit se při generování výstupu na různé části vstupní sekvence.

Modely založené na transformerech

Asi nejznámější architekturou jsou modely založené na transformerech. Transformer model, paradigma neuronových sítí, které způsobilo revoluci ve zpracování přirozeného jazyka, byl představený v roce 2017 *Vaswanim a spol.* v článku *Attention is All You Need* [38]. Využívá mechanismus známý jako *self-attention* nebo také *intra-attention*, který zpracovává celou sekvenci slov současně. To modelu umožňuje zvážit důležitost každého slova ve větě a zachytit nuance, které se odvíjejí v různých kontextech.

Ukázaly se jako vysoce efektivní v široké škále úloh, jako je strojový překlad, shrnutí textu, modelování jazyka a analýza sentimentu. Kromě toho byly přizpůsobeny pro různé multimodální úlohy, které kombinují vstupy z různých modalit, jako je text a obrázky.

Transformery umožňují efektivní trénování na velkých souborech dat a rychlejší inferenci, což je praktické pro aplikace zpracování jazyka v reálném čase a ve velkém měřítku. I když jsou vysoce efektivní, modely založené na transformerech vyžadují značné výpočetní zdroje a mohou mít problémy se zpracováním extrémně velkých souborů dat.

Modely založené na rekurzivních neuronových sítích

Velké jazykové modely založené na rekurentních neuronových sítích (Recurrent neural network, RNN) zpracovávají datové struktury hierarchickým a rekurzivním způsobem, díky čemuž se dobře hodí k pochopení složité a vnořené struktury jazyka.

Vynikají ve zpracování strukturovaných dat, jako jsou parsovací stromy, jenž představují syntaktickou strukturu věty a také v úlohách, které zahrnují porozumění struktuře vět, jako je analýza sentimentu a sémantický rozbor. Jejich výkonnost může být negativně ovlivněna při práci s velmi složitými jazykovými vzory či při zpracování nestrukturovaného textu.

Hierarchické modely

Díky svému potenciálu pracovat s různými úrovněmi vstupních dat, jako jsou slova, věty a odstavce, což odpovídá struktuře přirozeného jazyka, kde jsou informace uspořádány hierarchicky, si i hierarchické velké jazykové modely získaly značnou pozornost. Tyto modely pracují na různých úrovních granularity, a tak mají potenciál zachytit komplexní vztahy a informace v rámci textových dat.

Jsou úspěšné v úlohách jako je klasifikace dokumentů. Jednou z výzev je ale zvýšená složitost modelování a trénování, což může vyžadovat pokročilé výpočetní zdroje a infrastrukturu. Zároveň tyto modely mohou narážet na problémy při zpracování velmi nejednoznačných nebo kontextově proměnlivých dat.

2.1.2 Architektura

Obecná architektura velkých jazykových modelů obvykle zahrnuje více vrstev neuronových sítí, jako jsou rekurentní vrstvy, dopředné (feedforward) vrstvy, embedding a attention vrstvy. Tyto vrstvy pracují v tandemu při zpracování vstupního textu a generování výstupního obsahu.

Architektury se samozřejmě liší už podle typu modelu, dnes se běžně kombinuje několik funkcionalit dohromady. Nejznámější LLM architekturou je architektura založená na transformerech. Mezi velké jazykové modely postavené na transformerové architektuře patří známé modely, jako jsou GPT-2¹, GPT-3² a GPT-4³, a také některé Open Source modely jako LLaMA⁴, BERT⁵, Mistral-7B⁶ nebo Falcon LLM⁷. *Jurafsky* a *Martin* [17] uvádějí, že takovýto jazykový model se běžně skládá ze čtyř hlavních kroků:

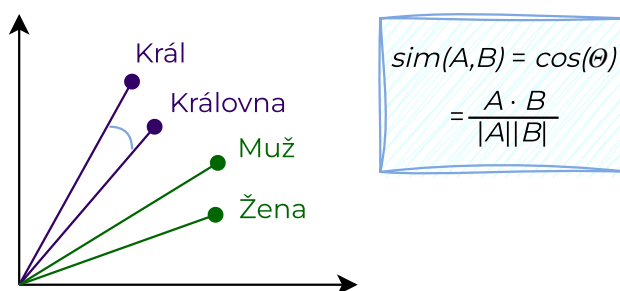
1. Embedding slov,
2. Poziční kódování,
3. Vrstvy s transformery,
4. Generování textu.

Pochopení fungování jazykových modelů je pro práci s nimi klíčové. Proto se v následujících podkapitolách práce věnuji detailnějšímu popisu každého z těchto kroků.

Embedding slov

Lidé vyjadřují slova pomocí posloupnosti písmen, například *P-E-S* jako *pes*. Pro zpracování slov pomocí jazykových modelů je podle *Jurafskyho* a *Martina* [17] klíčová reprezentace tokenů (tj. slov nebo podslov) jako vektorů ve vysokodimenzionálním prostoru, kde jsou podobná slova seskupena. To pomáhá modelu porozumět významu slov a na základě tohoto porozumění vytvářet předpovědi.

Princip lze ilustrovat na slovech *král* a *královna*. Tato dvě slova k sobě mají obvykle blíže než jiná dvojice slov, například *královna* a *traktor*. Podobnost mezi těmito slovy spočívá v jejich společných vlastnostech, protože obě slova představují královské tituly, často spojené s monarchií a autoritou. V kontextu embeddingu slov by tato slova byla zobrazena jako vektory umístěné ve vektorovém prostoru v těsné blízkosti.



Obrázek 2.2: Vztah vektorů v embedding prostoru.

¹<https://huggingface.co/openai-community/gpt2>

²<https://platform.openai.com/docs/models/gpt-3-5-turbo>

³<https://openai.com/research/gpt-4>

⁴<https://huggingface.co/meta-llama/Llama-2-7b>

⁵https://huggingface.co/docs/transformers/model_doc/bert

⁶<https://huggingface.co/mistralai/Mistral-7B-v0.1>

⁷<https://huggingface.co/tiiuae/falcon-7b>

Z matematického hlediska by měl být kosinus úhlu mezi těmito vektory blízký 1, tj. úhel blízký 0. Rozdíly vektorů mezi dvojicí slov lze přičíst k jinému vektoru slov a najít analogické slovo. Například *muž - žena + královna ≈ král*. Tento vztah popisuje i *Ibrahim* [13] a je demonstrován na obrázku 2.2.

Dříve byly pro reprezentaci slov používány techniky jako Bag of Words a TF-IDF (Term Frequency-Inverse Document Frequency). Kromě toho existují techniky, jako je jednofázové kódování, nebo lze použít jedinečná čísla pro reprezentaci slov ve slovníku.

Dnes snad nejpoužívanější metoda **Word2Vec** byla představena kolektivem výzkumníků společnosti Google vedeném *Mikolovem* [27]. Při tréninku využívá techniku CBOW (Continuous Bag of Words, distribuované reprezentace kontextu se kombinují za účelem předpovědi slova uprostřed) a/nebo Skip-Gram (distribuovaná reprezentace vstupního slova se používá k předpovědi kontextu). Word2vec je podle *Chaubarda a spol.* [4] metoda pro efektivní vytváření embeddingů slov pomocí dvouvrstvé neuronové sítě, nejedná se sice o hlubokou neuronovou síť, ale převádí text do číselné podoby, které hluboké neuronové sítě rozumí. Vstupem metody word2vec je textový korpus a jejím výstupem je sada vektorů známých jako feature vectors, které reprezentují slova v tomto korpusu.

Poziční kódování

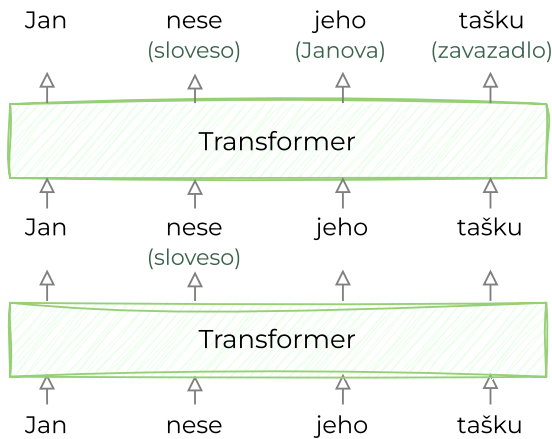
Vzhledem k tomu, že transformer ze své podstaty nerozumí pořadí tokenů v sekvenci, přidávají se ke vstupním embeddingům poziční kódování, která tuto informaci o pozici poskytují, jak uvádějí *Jurafsky a Martin* [17] v kapitole *The input: embeddings for token and position*.

Transformery využívají pokročilou metodu pozičního kódování, při níž je každá pozice nebo index spojen s vektorovou reprezentací. V důsledku toho je podle *Saeeda* [36] výstupem vrstvy pozičního kódování matice, jejíž každý řádek označuje zakódovaný prvek sekvence v kombinaci s jeho polohovými daty. Tato technika zajišťuje, že model dokáže účinně zachytit jedinečné atributy polohy každého prvku ve vstupní sekvenci, což zvyšuje jeho schopnost vhodně zpracovat sekvenční informace.

Vrstvy s transformery

Dá se říci, že limitace rekurentních neuronových sítí a LSTM vedly k vývoji transformer modelu. Autoři *Jurafsky a Martin* [17] v kapitole *The Transformer: A Self-Attention Network* zmiňují, že transformery jsou, stejně jako LSTM, schopné zpracovávat vzdálené informace, ale na rozdíl od nich nejsou založeny na rekurentních spojeních, které lze obtížně paralelizovat. Transformer modely jsou zaměřeny na zvýšení rychlosti a efektivity zpracování tím, že využívají přístup paralelního zpracování sekvencí.

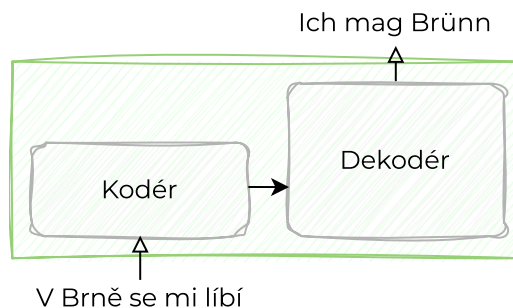
V praxi je v LLM často použito více vrstev transformerů, které postupně předávají embeddingy, dokud se konečný výstup smysluplných výsledků nepoužije k předpovědi dalšího slova v sekvenci, což popisuje i *Mulkar* v [28]. Jinak řečeno, v rámci této vrstevnaté architektury slouží výstupy jednoho transformeru jako vstupy pro další, čímž se postupně hromadí kontext. Tento proces je zjednodušeně znázorněn na schématu na obrázku 2.3. Toto schéma vyobrazuje architekturu se dvěma vrstvami transformerů, skutečné jazykové modely se ale obvykle skládají z více než jen dvou vrstev.



Obrázek 2.3: Vrstvená architektura transformerů.

Vstupní data modelu na obrázku, zobrazená ve spodní části, se skládají z věty *Jan nese jeho tašku*. Slova, reprezentovaná jako vektory stylu word2vec, jsou vložena do počátečního transformeru. Ten rozpozná, že *nese* je přísudek, tedy nosné sloveso. V praxi model tento dodatečný kontext zahrnuje tak, že pozmění vektory slov na skryté stavové vektory. Tyto upravené vektory se přenášejí do následujícího transformeru.

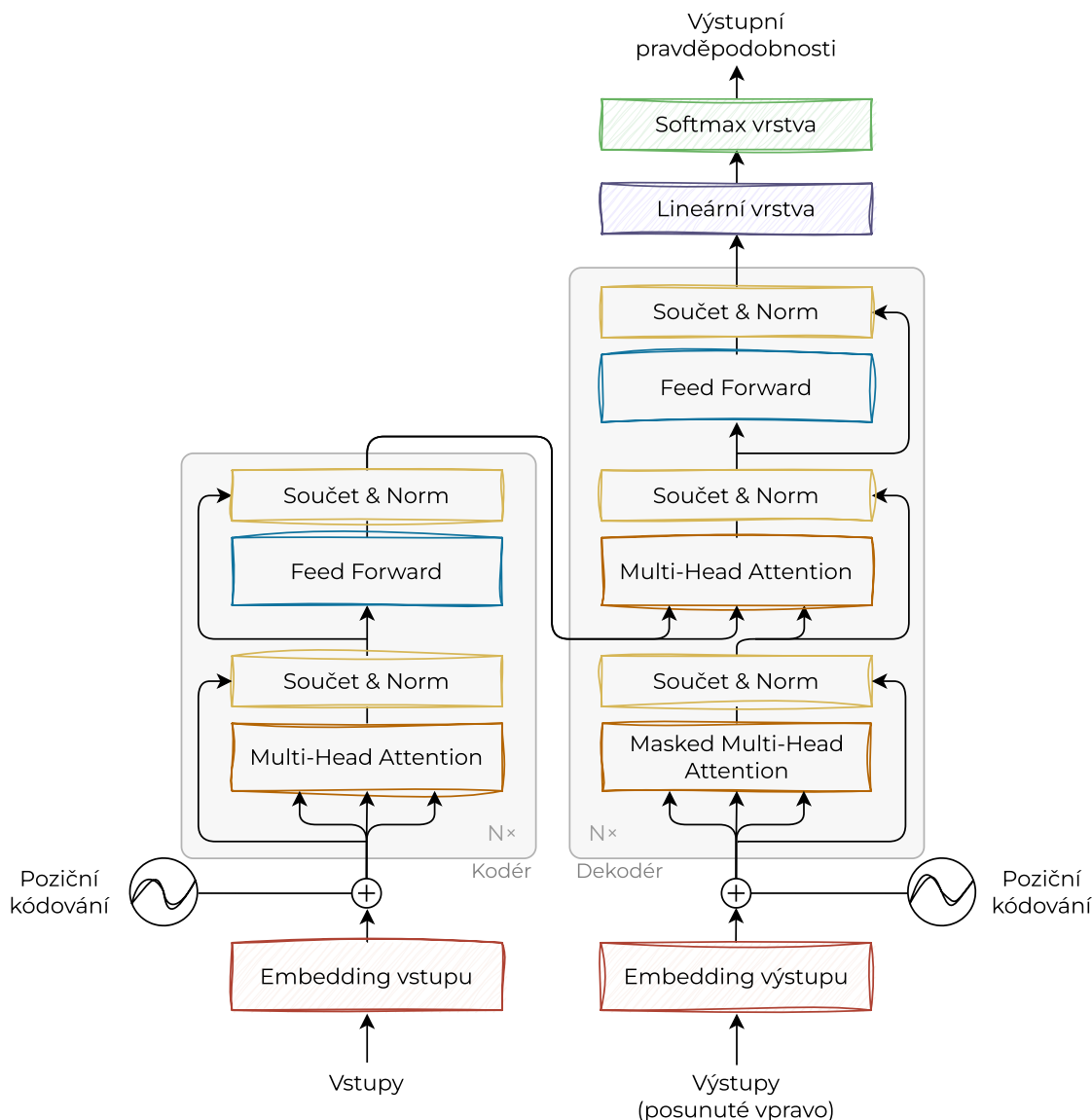
Druhý transformer poskytuje další kontext tím, že rozpozná slovo *tašku* jako zavazadlo a nikoli střešní tašku, a identifikuje *jeho* jako zájmeno odkazující na Jana. Výstupem je vytvořená další řada skrytých stavových vektorů, které zahrnují znalosti, jež model v této fázi nashromáždil.



Obrázek 2.4: Transformer – kodér a dekodér.

Samotnou vnitřní architekturu transformeru, tak jak byla představena *Vaswanim a spol.* [38], vyobrazuje schéma na obrázku 2.5. Architektura se dá rozdělit na dvě základní části, a to kodér a dekodér, jak prezentuje obrázek 2.4. Reálně je část kodéru spíše sestava kodérů – v architektuře ze zmíněného článku je jich šest nad sebou, kdy každá vrstva postupně vytváří dokonalejší reprezentaci vstupní sekvence, podobně pak s částí dekodéru.

Všechny kodéry mají stejnou konstrukci (nemají však společné váhy). Architektura kodéru podle *Vaswanio a spol.* [38] se skládá z řady identických vrstev, z nichž každá obsahuje dvě základní podvrstvy: multi-head self-attention mechanismus a pozičně plně propojenou feed-forward síť. Výstup poslední vrstvy kodéru je nakonec přenášen do dekodéru.



Obrázek 2.5: Architektura transformeru podle [38].

Multi-head attention zahrnuje rozdělení vstupu do více hlav, přičemž self-attention mechanismus je aplikován nezávisle na každou hlavu. Self-attention je mechanismus, který umožňuje každému tokenu ve vstupní sekvenci soustředit se na různé segmenty sekvence a využít různá skóre pozornosti k výpočtu komplexní reprezentace sekvence. Tato funkce se ukazuje jako výhodná pro lepší uchopení kontextu a vztahů v sekvenci bez ohledu na prostorové oddělení tokenů. Samotný proces je implementován v rámci každé vrstvy kodéru, což usnadňuje schopnost modelu zvažovat význam ostatních slov ve vstupní sekvenci pro každé jednotlivé slovo.

Výsledné výstupy z hlav se pak spojí a podrobí lineární transformaci. *Mostafa* [13] uvádí jako výhodu tohoto přístupu to, že umožňuje modelu soustředit se na různé pozice, což usnadňuje zachycení různorodých aspektů informací přítomných ve vstupní sekvenci. V každé vrstvě kodéru se výstup attention vrstvy zpracovává pomocí Feed-Forward neu-

ronových sítí. Tento krok slouží k vylepšení reprezentací tím, že do naučených rysů vnáší složitost a hloubku.

Dekodér používá naučenou meziproductovou reprezentaci rysů z kodéru k vytvoření výstupní sekvence vždy jeden token za druhým. Jako součást vstupu využívá také své vlastní předchozí výstupy, což je nezbytné pro generování dalšího tokenu v sekvenci na základě předchozích tokenů. Dekodér se od kodéru liší tím, že vkládá druhou multi-head cross-attention podvrstvu, která aplikuje attention mechanismus jak nad výstupem kodéru, tak nad nově vygenerovanými výstupními tokeny.

Autoři *Jurafsky* a *Martin* [17] popisují, že při přechodu do lineární vrstvy a vrstvy s funkcí softmax prochází výstup řadou transformací. Lineární vrstva je zodpovědná za převod výstupu dekodéru na logity velikosti slovníku. Tyto logity představují nenormalizované predikce pro každý token ve slovníku. Poté se na tyto logity aplikuje funkce softmax, která je změněna na pravděpodobnosti. Token s nejvyšší pravděpodobností je poté vybrán jako výstupní token a podílí se na generování konečného výstupu.

Zde je potřeba zmínit, že architektury jazykových modelů nejsou všechny stejné, a to i mezi těmi založenými na transformerech. Některé modely jsou například založeny pouze na kodéru transformeru, sem spadá například BERT. Modely GPT, včetně GPT-2 a GPT-3, a třeba také model Mistral, jsou naopak postaveny na architektuře transformer dekodéru. Celou architekturu kodér-dekodér využívají modely LLaMA a T5⁸.

Generování textu

Při generování textu model vypočítá pravděpodobnostní rozdělení nad slovníkem pro další slovo nebo token v sekvenci, což *Jurafsky* a *Martin* [17] uvádějí v kapitole *Large Language Models: Generation by Sampling*. Toto rozdělení je založeno na naučených vzorcích a vztazích v rámci trénovacích dat a kontextu poskytnutém vstupem. K vygenerování dalšího slova nebo tokenu může model použít vzorkovací techniky, jako je hladové (greedy) dekodování, náhodné vzorkování či vzorkování top K , což pomáhá určit nejvhodnější token, který má být vygenerován jako další, s ohledem na dříve vygenerované tokeny a pravděpodobnostní rozdělení slovní zásoby.

Na základě vybraného slova nebo tokenu model vygeneruje další část sekvence. *Reznikov* [35] uvádí, že se tento proces iterativně opakuje, dokud nevznikne požadované množství textu nebo dokud není splněno kritérium zastavení.

2.2 Doménově specifictí komunikační agenti

Velké jazykové modely dosáhly pozoruhodného úspěchu při zpracování přirozeného jazyka a prokázaly vynikající výkon na mnoha benchmarcích (jako je např. Trustbit LLM Benchmarks⁹ či HuggingFace Open LLM Leaderboard¹⁰) pro různé úlohy.

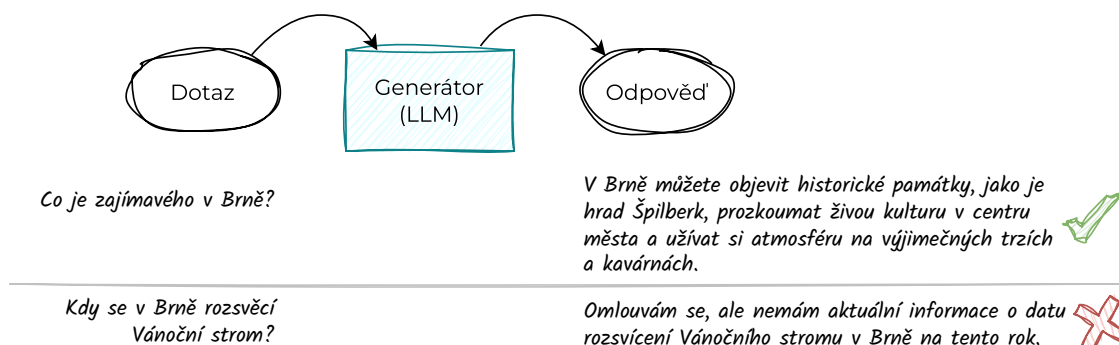
I přes tyto pokroky vykazují LLM značná omezení, zejména při zpracování doménově specifických nebo vysoce specializovaných dotazů. Jsou totiž trénovány na velkém množství

⁸<https://huggingface.co/google-t5/t5-base>

⁹<https://www.trustbit.tech/en/llm-benchmarks>

¹⁰https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

dat, aby bylo dosaženo širokého spektra obecných znalostí uložených ve vahách neuronové sítě (parametrické paměti), jak popisuje i *Gao a spol.* v článku *Retrieval-Augmented Generation for Large Language Models: A Survey* [9].



Obrázek 2.6: Schéma základního konceptu jazykových modelů.

Častým problémem je proto generování nesprávných informací (zpravidla označováno jako halucinace), zejména vyžaduje-li dotaz znalosti, jež nebyly zahrnuty v tréninkových datech modelu a vyžadují aktuální informace, což demonstruje i diagram na obrázku 2.6. Tradiční jazykové modely, ačkoli jsou silné v generování textu, často nemají schopnost ověřit přesnost informací, které produkují.

Samotné jazykové modely mají sice široké uplatnění, avšak nejsou vždy dostačující pro konkrétní požadavky. To tedy platí zejména v případech, kdy aplikace vyžaduje alespoň jednu z následujících vlastností:

- a) Přizpůsobený styl psaní,
- b) Specifický kontext,
- c) Specializovaná slovní zásoba.

Metody, které se tento problém snaží řešit se liší potřebnou odborností, náklady a vhodností pro různé scénáře. Patří sem:

1. Plný fine-tuning – úprava všech parametrů pomocí specifických dat,
2. PEFT – úprava vybraných parametrů modelu,
3. Adapter – přidání parametrů LLM specifických pro danou úlohu,
4. Prompt inženýrství – řízení výstupu modelu specifickými pokyny,
5. RAG – kombinace prompt inženýrství s přístupem k datům v externí databázi.

V následujících sekcích se práce bude stručně zabývat každou z těchto metod, neboť každá je odlišná v zásadním ohledu a vhodná pro různé situace použití. Při implementaci komunikačního agenta pro informace o Brně je rozhodující vybrat tu nejvhodnější metodu.

2.2.1 Plný fine-tuning

Plný fine-tuning podle *Jurafskyho a Martina* [17] zahrnuje trénování předem natrénovaného LLM menším, specifickým souborem dat, aby se přizpůsobil pro konkrétní úlohu nebo aby se zlepšil jeho výkon. Upravují se váhy modelu na základě doménově specifických dat, čímž se model více přizpůsobuje jedinečným potřebám dané aplikace, přičemž předtrénovaný model zajišťuje schopnosti porozumění jazyku.

Pro fine-tuning je potřeba rozsáhlý soubor anotovaných dat z cílové domény. Jak uvádí i *Hotz*[11], výsledný model obsahuje pevnou reprezentaci těchto dat, což je nepraktické pro případy, kdy je potřeba častá aktualizace informací, které je model schopen poskytnout. Pro udržení relevance LLM v takovém dynamickém prostředí je nutné pravidelné přeučování, které si může vyžádat značné časové nároky a zdroje. Každé opakování navíc vyžaduje pečlivé sledování, aby bylo zaručeno, že revidovaný model bude i nadále efektivně fungovat v různých scénářích, aniž by vnášel nová zkrslení nebo mezery ve znalostech.

Plný fine-tuning je skvělou volbou, jsou-li k dispozici anotovaná data a je cílem model lépe přizpůsobit konkrétním potřebám, přičemž není potřeba aktualizace informací.

2.2.2 PEFT

Parametrově efektivní fine-tuning (PEFT, Parameter-Efficient Fine-Tuning) je velmi podobný předchozí metodě, avšak zahrnuje fine-tuning jen malé sady doménově specifických parametrů, přičemž většina parametrů předtrénovaného LLM zůstává pevně daná.

Autoři *Mangrulkar a Paul* v článku *PEFT: Parameter-Efficient Fine-Tuning of Billion-Scale Models on Low-Resource Hardware* [25] zmiňují, že díky zachování většiny parametrů předtrénovaných LLM se nejen výrazně sníží výpočetní a paměťové nároky, ale zároveň to i řeší problémy s katastrofickým zapomínáním a nadměrným přizpůsobením (overfitting) což je chování pozorované při úplném fine-tuningu LLM.

Stejně jako plný fine-tuning, i PEFT není ideálním přístupem pro systémy s potřebou často se aktualizujících informací. Tato metoda snižuje spotřebu zdrojů a může nabídnout srovnatelný výkon jako úplný fine-tuning s minimálními aktualizacemi parametrů, ale může omezit přizpůsobivost modelu tím, že aktualizuje pouze malou podmnožinu parametrů.

2.2.3 Adapter

Tato metoda zvyšuje adaptabilitu modelu tím, že vedle předtrénovaných jazykových modelů zavádí parametry specifické pro danou úlohu. Jak popisují i autoři v článku *Adapters: A Unified Library for Parameter-Efficient and Modular Transfer Learning* [30], ve kterém byla metoda poprvé představena, tento přístup umožňuje efektivní přizpůsobení novým úlohám bez rozsáhlých změn původní architektury modelu.

Jedná se o integraci adaptačních vrstev do architektury předtrénovaných LLM, což umožňuje doménově specifické nastavení modelu s minimem dodatečných trénovatelných parametrů.

Ačkoli metoda nabízí výhody v oblasti efektivního fine-tuningu parametrů a přizpůsobování jazykových modelů konkrétním úlohám, může zaostávat v oblastech vyžadujících roz-

sáhlou integraci externích znalostí a kontextovou relevanci. V porovnání s metodou PEFT a úplným fine-tuningem nabízí adapter zjednodušený přístup s nenáročnými parametry specifickými pro danou úlohu, což kontrastuje s rozsáhlejšími aktualizacemi parametrů.

2.2.4 Prompt inženýrství

Prompt inženýrství je kritickým aspektem práce s LLM a zahrnuje sestavování a optimalizaci vstupních pokynů (*prompt*) takovým způsobem, aby model generoval specifické a žádané výstupy aniž by bylo nutné měnit základní parametry modelu [26].

Prompt inženýrství má tu výhodu, že nezasahuje do architektury modelu a umožňuje přizpůsobit LLM specifickým úlohám, aniž by se přímo měnily předtrénované parametry, jak popisuje i *Shin a spol.* v [37]. Navzdory jeho účinnosti je jedním z omezení potenciální složitost při navrhování optimálních pokynů k dosažení požadovaného chování modelu, což mnohdy vyžaduje časově náročné experimentování.

I přesto, že se jedná o účinnou metodu pro vývoj doménově specifických komunikačních agentů, nemusí být prompt inženýrství vhodné pro úlohy vyžadující hluboké porozumění specializovaným doménám nebo schopnost vytvářet vysoce kontextově citlivé výstupy. Jedná se spíše o optimalizaci stylu výstupu z modelu dle speciálních požadavků, jelikož narušil od předchozích metod model nezískává žádné nové znalosti úpravami parametrů.

2.2.5 RAG

Generování s rozšířeným vyhledáváním (RAG, Retrieval-Augmented Generation), představené týmem *Patricka Lewise* v průlomovém článku *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks* [20], kombinuje vyhledávací systém, který získává relevantní fragmenty dokumentů z externího zdroje, a LLM, který vytváří odpovědi pomocí informací z těchto fragmentů.

Tato metoda zahrnuje bezproblémovou integraci externích informací pro generování přesnějších a informovanějších odpovědí, což umožňuje překonat omezení statických tréninkových dat a řešit nedostatky ve znalostech i potřebu stále nových informací, jak popisuje *Gao a spol.* [9]. Nevýhodou je důležitost dobře uspořádaných dat a možnost zkresleného vyhledávání informací.

Ve srovnání s ostatními metodami se RAG liší od metod fine-tuning, PEFT, adapter a prompt inženýrství svým specifickým přístupem k vylepšování LLM přidáním externích dat. V některých případech může být optimální hybridní přístup využívající jak RAG, tak fine-tuning či některou z dalších metod.

Při vývoji komunikačního agenta s informacemi o Brně je pro jeho účinnost a relevanci nejdůležitější zajistit přístup k externím aktuálním informacím. Za tímto účelem byla jako základní rámec architektury v této práci zvolena právě RAG architektura, které se bude věnovat i celá následující kapitola.

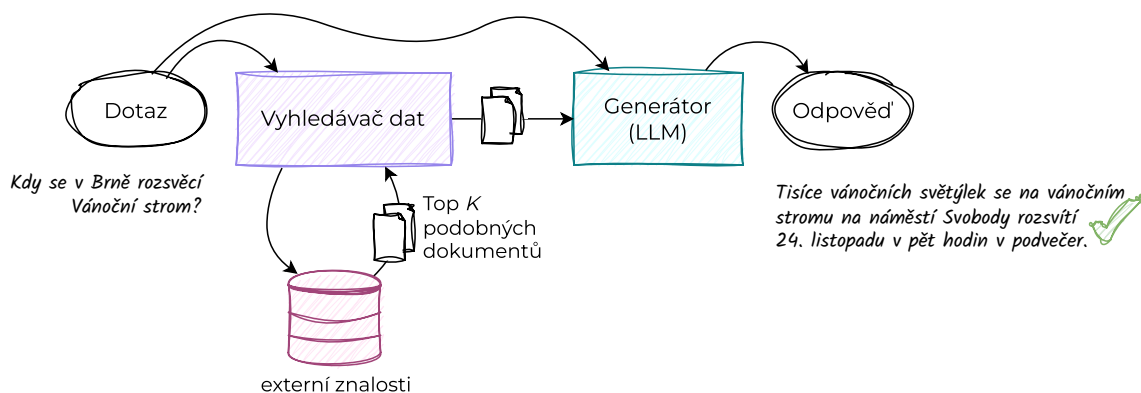
Kapitola 3

Generování s rozšířeným vyhledáváním

Jak již bylo zmíněno, generování s rozšířeným vyhledáváním se objevilo jako odpověď na potřebu, aby měly jazykové modely přístup ke spolehlivým faktickým informacím, které mohou zakotvit jejich generování jazyka ve znalostech reálného světa. Jedná se o slibné řešení začleněním znalosti z externích databází, čímž se zlepšuje přesnost, spolehlivost i důvěryhodnost výstupu. V této kapitole se věnuji základním i pokročilým metodám při návrhu RAG systémů, jejich optimalizacím a evaluaci.

V základu RAG zahrnuje počáteční krok získávání informací, kdy jazykový model dotazuje externí zdroj dat, aby získal relevantní informace, než přejde k odpovídání na otázky nebo generování textu, jak popisuje *Gao a spol.* [9]. Tento proces dodává informace pro následující fázi generování výstupu a zajistí, že jsou odpovědi podloženy získanými důkazy, což významně zvyšuje přesnost a relevanci výsledků. Dynamické získávání informací ze znalostních databází během fáze inferenčního zpracování umožňuje systému lépe řešit i problém halucinací. Základní koncept RAG popisuje schéma na obrázku 3.1.

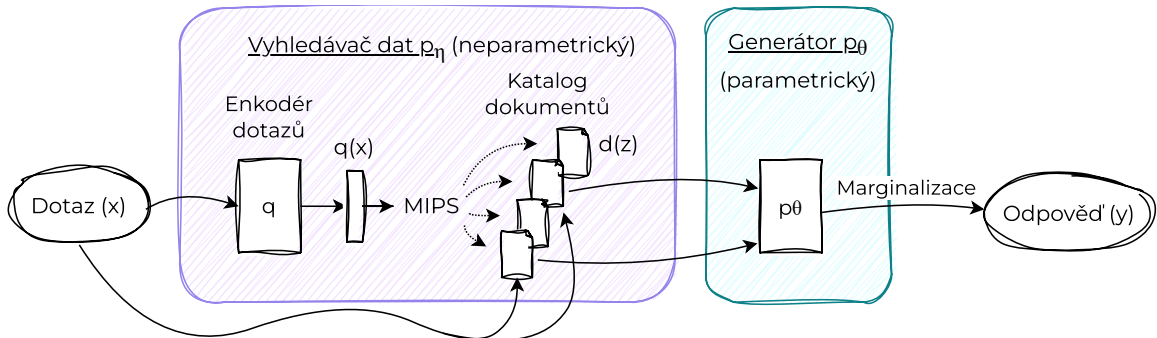
RAG má tedy oproti klasickým LLM mnoho výhod. Jeho znalosti lze snadno aktualizovat nahrazením nebo přidáním dalších dokumentů/informací do neparametrické paměti, nevyžaduje tedy přeučení modelu. přičemž umožňuje uživateli zkontrolovat, které dokumenty byly načteny, a poskytnout tak kontext, což u tradičních LLM nelze.



Obrázek 3.1: Schéma základního konceptu RAG.

3.1 Metody

Základní RAG se řídí tradičním postupem, který zahrnuje indexování, vyhledávání a generování, jak uvádí *Lewis a spol.* [20], odkud jsou čerpány informace v této podkapitole. Model využívá vstupní sekvenci x k získání textových dokumentů z , které jsou použity jako další kontext při generování cílové rovnice (přístup popisuje obrázek 3.2). Základními kameny jsou vyhledávač $p_\eta(z|x)$ s parametry η , který vrací top K podobných dokumentů vzhledem k dotazu x , a generativní model ($p_\theta(y_i | x, z, y_{1:i-1})$).



Obrázek 3.2: Původní přístup dle [20].

Předtrénovaný vyhledávač (enkodér a katalog dokumentů) je kombinován s předtrénovaným sequence-to-sequence (seq2seq) modelem (generátor), následuje ladění. Pro dotaz x se používá vyhledávání maximálního vnitřního produktu (MIPS, Maximum Inner Product Search) k nalezení top K dokumentů z_i . Pro koncovou predikci y je z marginalizováno přes seq2seq předpovědi dané různými dokumenty. Rozlišujeme zde tedy dva typy znalostí:

- Parametrické znalosti – naučené během tréninku,
- Neparametrické znalosti – získané z externího zdroje znalostí.

Parametrické znalosti jsou implicitně uloženy ve váhách neuronové sítě, zatímco neparametrické znalosti jsou běžně ukládány ve vektorové databázi.

3.1.1 Ukládání a vyhledávání dat

Obvykle se používají techniky, jako je husté vyhledávání pasáží (DPR, Dense Passage Retrieval). Jeho cílem je indexovat miliony článků v nízkorozměrném a spojitém prostoru a efektivně vyhledat K nejlepších článků, které jsou pro danou otázku nejrelevantnější. DPR používá dva enkodéry – enkodér úryvků a enkodér otázek.

Enkodér úryvků převádí každý článek na d -rozměrný vektor a indexuje jej, většinou pomocí knihoven pro vyhledávání podobnosti vektorů, jako je např. FAISS¹. Ve fázi indexování se dokumenty zpracují, segmentují a transformují na embeddingy (vektorové reprezentace), které se uloží do vektorové databáze. Kvalita sestavení indexu určuje, nakolik

¹<https://github.com/facebookresearch/faiss>

lze ve fázi vyhledávání získat odpovídající kontext – vektorová reprezentace umožňuje matematicky zachytit kontext znalostí, což umožňuje odhalit informace, které nejsou explicitně uvedené, ale jsou zřejmé z kontextu.

Enkodér otázek převede vstupní otázku na d -rozměrný vektor a poté (též s dopomocí k tomu určených knihoven) vyhledá K úryvků, které mají nejbližší vektor k vektoru otázky. Podobnost mezi vektory (zde u a v) lze dle *Jurafskyho* a *Martina* [17] vypočítat pomocí jejich bodového součinu: $u \cdot v = |u||v| \cos(\theta)$.

Architektura kodéru používaného v DPR v původním článku [20] je již dříve zmíněná síť BERT, která převádí vstup na vysokorozměrný vektor. Lze však použít jakoukoli funkční architekturu, kdy se dokumenty zakódují do vysokorozměrných embeddingů a poté se k vyhledání nejrelevantnějších dokumentů použijí účinné metody jako je vyhledávání podle nejbližšího souseda.

Vyhledávací komponenta tedy vyžaduje znalostní databázi, která používá nějaký druh kódování k zachycení podobnosti, často pomocí vektorového embedding modelu. Tato vektorová databáze se stává zdrojem, který se používá k rozšíření znalostí LLM.

3.1.2 Generování odpovědi

Poté, co vyhledávací model získá příslušné informace, přichází na řadu generativní model, který slouží jako poslední krok poskytující textový výstup. Generativní model má za úkol syntetizovat vyhledané informace do souvislého a kontextově relevantního textu. Tyto modely jsou obvykle postavené na rozsáhlých jazykových modelech a stejně jako LLM mají schopnost vytvářet text, který je gramaticky správný, sémanticky smysluplný a v souladu s původním dotazem.

3.2 Pokročilé metody

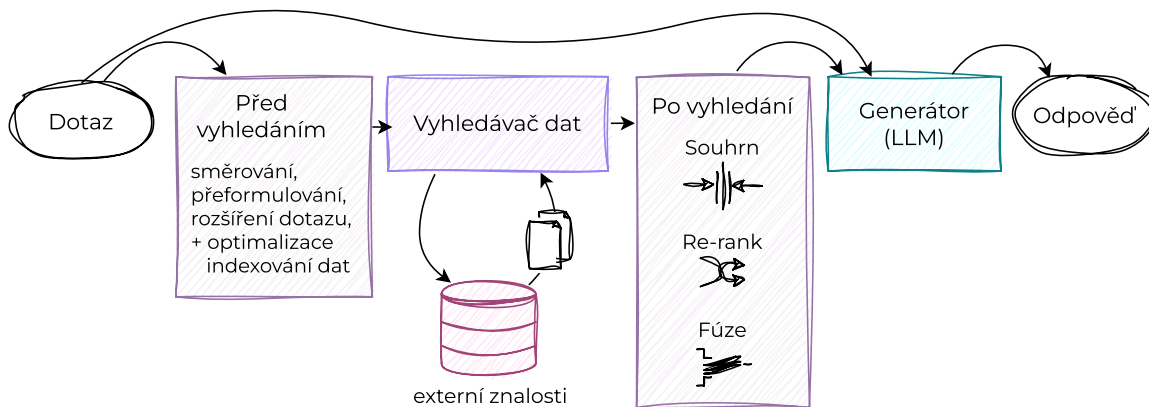
S rozšiřujícím se spektrem aplikací těchto systémů jsou požadavky kladené na RAG stále složitější a rozmanitější. Základní architektura RAG je sice robustní, avšak může být nedostatečná pro adresování široké škály potřeb různých odvětví a dynamicky se vyvíjejících scénářů použití. Zde přicházejí ke slovu pokročilé metody optimalizující RAG, které jsou přizpůsobeny specifickým výzvám a nabízejí větší přesnost, přizpůsobivost a efektivitu při zpracování informací.

Autoři *Gao a spol.* [9] zmiňují hned několik nedostatků základního konceptu RAG. Při vyhledávání dat může podle nich způsobovat problémy kvalita načítání a nízká přesnost, což vede k nesprávnému zarovnání načtených úryvků a potenciálním problémům, jako jsou halucinace. Také se nemusí podařit načíst všechny relevantní úryvky, čímž se ztěžuje schopnost LLM vytvářet komplexní odpovědi kvůli nekompletním informacím pro generování výstupu.

Problémem může být i nadbytečnost a opakování, zejména pokud více načtených úryvků obsahuje podobné informace. To jak uvádí *Ilin* [14] může vést k potencionálnímu opakování obsahu v generované odpovědi. Další výzvou je podle *Besbese* [1] rozpoznání důležitosti a relevance pro generování výstupu více znovu vyhledaných pasáží, což vyžaduje správné

posouzení hodnoty každé pasáže. Kromě toho může být pro kvalitu odpovědi důležité sladit rozdíly ve stylu a tónu psaní, aby byla zajištěna konzistence výstupu.

Jak ukazuje schéma na obrázku 3.3, pokročilá RAG architektura podle *Gao a spol.* [9] implementuje optimalizační strategie před i po procesu vyhledávání informací z příslušné vektorové databáze. Zároveň jsou aplikovány i metody optimalizace indexování dat, např. jemné segmentování nebo práce s metadaty.



Obrázek 3.3: Pokročilá architektura RAG podle [9].

3.2.1 Optimalizace před vyhledáním dat

Optimalizace v procesu před vyhledáváním informací hrají zásadní roli při zvyšování účinnosti a efektivity vyhledávání informací. Jak popisuje *Lin* [14], tato fáze se zaměřuje na přípravu systému na vyhledávání relevantních a přesných informací pomocí doladění počátečního dotazu a optimalizace základních datových struktur. Proces před vyhledáváním zahrnuje řadu strategických optimalizací, včetně směrování dotazů, přepisování dotazů, rozšiřování dotazů a optimalizace indexování dat.

Směrování dotazů

Jednou z optimalizačních technik je směrování (routing) dotazů s cílem využít více indexovacích strategií, jak zmiňuje *Ma* v [23]. Jediný vektorový index má omezení při zpracování různých typů dotazů. Například jeden index může předávat sumarizační dotazy, jiný zpracovává bodové dotazy a další dobře funguje pro dotazy citlivé na datum. Vytvořením více indexů - například vektorového, souhrnného, seznamu a grafové znalostní báze - můžeme každý z nich optimalizovat pro různé charakteristiky dotazu.

Dotazy jsou pak při příchodu směrovány na příslušný index. To umožňuje sdružovat různé indexy dohromady a zároveň maximalizovat jejich individuální silné stránky. Dle *Gao a spol.* [9], může logika směrování při určování optimálního indexu zkoumat faktory, jako je sémantika dotazu, přítomnost klíčových slov nebo dat a podrobnost odpovědi. Tuto techniku podporují různé frameworky, např. *LlamaIndex*², či *LangChain*³.

²https://docs.llamaindex.ai/en/stable/examples/query_engine/RetrieverRouterQueryEngine/

³https://python.langchain.com/docs/use_cases/query_analysis/techniques/routing/

Přeformulování dotazů

Přeformulováním dotazu je proces, který zahrnuje přeformulování dotazu uživatele s cílem zlepšit jeho srozumitelnost, přesnost a relevanci pro systém vyhledávání informací, což popisují i *Ma a spol.* ve článku *Query Rewriting for Retrieval-Augmented Large Language Models* [24]. Dvě otázky, které se lidem zdají stejné, nevypadají totiž v embedding prostoru vždy tak podobně. Přeformulování má za cíl zvýšit účinnost následných kroků vyhledávání tím, že zpřesní jazyk nebo strukturu dotazu tak, aby lépe odpovídal dostupným zdrojům dat. S přeformulováním dotazů souvisí optimalizace zarovnání probraná v podkapitole 3.2.1. I tato metoda je podporována různými frameworky jako například LlamaIndex⁴.

Zároveň s touto metodou souvisí i dekompozice dotazů – rozdělení složitého uživatelského dotazu na menší, cíleně zaměřené dílčí dotazy, což umožňuje přesnější a cílenější vyhledávání informací, jak je popsáno i v dokumentaci této funkcionality v LangChain⁵ knihovně. Například zeptá-li se uživatel „*Jaký je rozdíl mezi RAG a PEFT?*“ a máme k dispozici oddělené dokumenty, jeden vysvětlující RAG a druhý vysvětlující PEFT, ale žádný dokument, který by je přímo porovnával, je pravděpodobné, že získáme lepší výsledky tím, že vyhledáme informace o každém zvlášť a poté je kombinujeme, než když bychom hledali přímo odpověď na otázku uživatele.

Rozšiřování dotazů

Rozšíření dotazů je technika používaná k obohacení původního dotazu uživatele přidáním souvisejících termínů nebo pojmů, aby bylo možné získat komplexnější sadu relevantních výsledků. Cílem tohoto procesu před vyhledáváním je odstranit případná omezení původního dotazu a zlepšit schopnost procesu vyhledávání identifikovat relevantní informace.

Je více způsobů, jak toho docílit, např. použít synonyma a další odvozené sémanticky příbuzné termíny, což může pomoci rozšířit rozsah vyhledávání a zachytit širší rozsah relevantních informací. *Troynikov* v kursu *Advanced Retrieval for AI with Chroma*⁶ zmiňuje také rozšíření dotazu o vygenerovanou odpověď nebo rozšíření dotazu pomocí několika souvisejících otázek.

Tradičním přístupem k rozšiřování dotazů je podle autorů *Jagermana a spol.* článku *Query Expansion by Prompting Large Language Models* [15] pseudo-relevantní zpětná vazba (PRF, Pseudo-Relevance Feedback). Ta se při rozšiřování dotazů spoléhá na získání dobré sady pseudo-relevantních dokumentů, jejichž obsah používá k extrakci nových termínů dotazu. V praxi však načtené dokumenty nemusí být s původním dotazem dokonale v souladu, zejména je-li dotaz krátký nebo nejednoznačný.

Autoři článku prezentují zajímavější metodu využívající generativní a kreativní schopnosti velkých jazykových modelů – LLM lze využít pro vygenerování rozšířeného dotazu, který obsahuje další termíny dotazu. Uvažovací techniky jako *Chain-of-Thought* jsou zde též užitečné, protože tyto podněty dávají modelu pokyn k postupnému rozčlenění dotazů a mohou poskytnout velké množství termínů souvisejících s původním dotazem. Prezentované

⁴https://docs.llamaindex.ai/en/stable/examples/query_transformations/query_transform_cookbook/#query-rewriting-using-querytransform

⁵https://python.langchain.com/docs/use_cases/query_analysis/techniques/decomposition/

⁶<https://www.deeplearning.ai/short-courses/advanced-retrieval-for-ai/>

experimentální výsledky na systémech MS-MARCO a BEIR ukazují, že rozšíření dotazů generovaná pomocí LLM jsou výkonnější než tradiční metody rozšiřování dotazů.

Optimalizace indexování dat

Jednou z dalších možných optimalizací je zlepšit indexování dat a zvýšit tak kvalitu indexovaného obsahu. To podle *Gaa a spol.* [9] zahrnuje pět základních strategií:

- zvýšení granularity dat,
- optimalizaci indexových struktur,
- přidání metadat,
- optimalizaci zarovnání.

Zvýšení granularity dat, jinak řečeno zlepšení kvality indexovaných dat, má za cíl zvýšit jednotnost textu, konzistenci, faktickou přesnost a bohatý kontext s cílem zvýšit efektivitu systému RAG. Toto zahrnuje eliminaci nepodstatných informací, odstranění nejednoznačností v pojmech a entitách, ověření faktické správnosti, udržení kontextu a celkovou eliminaci informačního šumu, což popisuje i *Besbes* [1]. Záleží-li zároveň na udržitelnosti RAG, jsou zapotřebí mechanismy pro obnovu zastaralých dokumentů.

I velikost úryvků dokumentů, které načítáme z vektorového úložiště, je zde klíčovým parametrem, proto je důležitá i **optimalizace indexových struktur**. Malá velikost úryvků může dle *Ilina* [14] vést k tomu, že v dokumentech budou chybět některé důležité informace, zatímco velká velikost může vnášet irelevantní šum. Hledání optimální velikosti zahrnuje nalezení vyváženého přístupu, který zachytí relevantní kontext, umožní efektivní dotazování a integruje informace z grafické struktury pro komplexní pochopení kontextu.

Rozšíření informací o **metadata** zahrnuje integrování relevantních metadat (jako například datum a typ) do indexovaných vektorů úryvků dokumentů do databáze, aby se umožnilo efektivnější filtrování, jak uvádí i *Gao a spol.* [9]. Užitečnost metadat spočívá v přidání další vrstvy strukturovaného vyhledávání, čímž lze výrazně zlepšit celkovou efektivitu vyhledávání.

Optimalizace zarovnání se věnuje řešení problémů spojených se správným zarovnáním a rozdíly mezi dokumenty. Jedním z přístupů k tomuto problému je zavádění hypotetických otázek do dokumentů – již se neindexuje úryvek pomocí jeho embeddingu, ale pomocí embeddingů otázek, na které odpovídá. S tímto přístupem přišel *Li a spol.* v článku [21]. Tyto otázky mají za cíl vyřešit potenciální nesrovnalosti v zarovnání a přispět k lepšímu porozumění obsahu mezi dokumenty. Tímto způsobem je možné dosáhnout vyšší kvality zarovnání a zvýšit srozumitelnost textových dat.

Podobným přístupem je indexování na základě souhrnu obsahu nebo vyhledávání vět na základě abstraktních popisů jejich obsahu, což prezentuje *Ravfogel a spol.* v článku *Retrieving Texts based on Abstract Descriptions* [33]. Použití stručných souhrnů zefektivní proces indexování, což může vést k rychlejšímu a přesnějšímu vyhledávání.

S generováním hypotetických otázek do dokumentů souvisí optimalizační metoda HyDE (Hypothetical Document Embedding), představená *Gaem a spol.* v článku *Precise Zero-Shot*

Dense Retrieval without Relevance Labels [8]. Spočívá v použití zero-shot učení. Po obdržení dotazu je LLM instruován, aby vygeneroval hypotetickou odpověď. Tato odpověď je sice k dotazu vždy relevantní, ale není reálná a může obsahovat nepřesnosti (často v případech, kdy je model dotázán na téma, které nebylo zahrnuto v jeho učící množině témat). Následně kontrastně naučený kodér (např. Contriever) transformuje odpověď na embedding vektor, který je pak spolu s vektorem dotazu použit ke zvýšení kvality vyhledávání. Výzkumy prezentované v článku naznačují, že HyDE prokazuje robustní výkon srovnatelný s optimalizovanými vyhledávači v různých úkolech, jako je vyhledávání na internetu, kontrola kvality a ověřování faktů, a to v různých jazycích.

3.2.2 Optimalizace vyhledávání dat

Během vyhledávací fáze je hlavním cílem identifikace vhodného kontextu výpočtem podobnosti mezi dotazem a uloženými částmi textu. V tomto procesu hraje klíčovou roli embedding model (který vytváří vektorové reprezentace), a právě zde leží potenciál k optimalizaci této fáze RAG.

Jak uvádí *Karpukhin* [18], **fine-tuning** embedding modelů výrazně ovlivňuje relevantnost získaného obsahu v systémech RAG. Tento proces zahrnuje adaptaci embedding modelů tak, aby zejména v profesních oblastech, které pracují s neobvyklými nebo se rozvíjejícími termíny, lépe odpovídaly hledaným informacím v kontextu specifických domén.

V závislosti na situaci bývá nezbytné dosáhnout přesné shody. Ačkoliv vektorové vyhledávání pomáhá nalézt sémanticky relevantní části pro daný dotaz, občas mu schází přesnost při hledání konkrétních klíčových slov. A právě tento problém podle *Broggse* [2] řeší **smíšené vyhledávání** (hybrid search). Tato strategie využívá silné stránky různých technologií vyhledávání, jako je vektorové vyhledávání a vyhledávání podle klíčových slov, a inteligentně je kombinuje.

3.2.3 Optimalizace po vyhledání dat

Po získání důležitého kontextu z databáze je klíčové integrovat ho s dotazem a vytvořit tak vstup pro LLM. Přímé vložení všech relevantních dokumentů do LLM najednou může překročit limit kontextového okna, přivést nežádoucí šum a zabránit efektivnímu zaměření na klíčové informace. Řešením těchto problémů je dle *Gaa a spol.* [9] provedení dodatečného zpracování získaného obsahu, jako je například re-ranking, fúze, či komprese promptů.

Re-ranking

Při zadávání dotazu do vektorové databáze nemusí být K nejlepších výsledků seřazeno nejrelevantnějším způsobem. Re-ranking (přehodnocení) tento problém řeší opětovným řazením nalezených informací s cílem identifikovat nejrelevantnější obsah podle aktuálních požadavků, což popisuje *Wang a spol.* [39]. Re-ranking může být vlastně považován za součást samotného procesu vyhledávání, jelikož je jeho úkolem zpřesnit a zlepšit původní pořadí vyhledaných dokumentů posouzením jejich relevance k dotazu, zejména v profesních oblastech, které pracují s neobvyklými nebo se rozvíjejícími termíny. Výhody re-rankingu

jsou však doprovázeny kompromisy, včetně zvýšené latence a výpočetních nákladů, což je činí méně vhodnými pro aplikace s vysokým provozem.

Opakované řazení nalezených informací s cílem znovu identifikovat nejrelevantnější obsah podle aktuálních požadavků je přímočarý koncept, který byl integrován mnoha frameworky, jako jsou LlamaIndex, LangChain a HayStack.

Asi není velkým překvapením, že při dnešním vývoji v této oblasti existuje více re-ranking modelů. Za nejznámější považuji Cohere⁷, hojně používaným je ale i Voyage AI Re-ranker⁸ nebo open-source Transformers Re-Ranker⁹.

Fúze

Aplikování fúze představil *Adrian Raudaschl* v článku *Forget RAG, the Future is RAG-Fusion* [32]. Tato metoda přináší optimální řešení pro eliminaci omezení RAG systémů, jako je neefektivita manuálního vyhledávání a přílišné zjednodušení vyhledávacích procesů, které často vedou k méně relevantním výsledkům.

Princip spočívá v duplikaci dotazů prostřednictvím převodu dotazů uživatele na podobné, ale odlišné dotazy pomocí LLM. Následně zahrnuje paralelní vektorové vyhledání jak původních, tak rozšířených dotazů a využívá pokročilých strategií, jako je Reciprocal Rank Fusion (RRF), což je technika, která kombinuje výsledky z různých dotazů pomocí opětovného řazení dat.

Generováním více uživatelských dotazů a implementací řazení výsledků model úspěšně zmenšuje rozdíl mezi uživatelskými dotazy a jejich zamýšleným významem. Konečným cílem je uspořádat výsledky vyhledávání do ucelené struktury, čímž se výrazně zvýší přesnost relevantních informací.

Souhrn dat a komprese promptů

Shromáždění mnoha relevantních dokumentů a jejich spojení do dlouhého dotazu není tak výhodné, jak by se mohlo zdát, jelikož příliš mnoho kontextu může ve skutečnosti přidat více irelevantních informací, což ztěžuje jazykovému modelu zachytit klíčové detaily a celkově negativně ovlivňuje výkonnost systému. Proto přichází souhrn dat a komprese promptů.

Přístupy jako *Selective Context* [22] a *LLMLingua* [16] využívají malé jazykové modely k výpočtu informační hodnoty nebo perplexity vstupních dat, což pomáhá odhadnout důležitost jednotlivých prvků. Naopak metody jako *Long Context* [40] a technika popsaná *Chenem a spol.* v článku *Walking Down the Memory Maze* [5] navrhuji sumarizační techniky, které zlepšují schopnost jazykových modelů vnímat klíčové informace, zejména při zpracování rozsáhlých kontextů.

⁷<https://cohere.com/blog/rerank>

⁸<https://docs.voyageai.com/docs/reranker>

⁹https://www.sbert.net/examples/applications/retrieve_rerank/README.html

3.3 Evaluace

I když je relevance získané odpovědi k uživatelské dotazu asi nejdůležitějším aspektem, hodnocení výkonu RAG vyžaduje zvažování různých klíčových kroků a metrik. Dvěma základními kroky systému je získávání informací z databáze a generování výsledku. Při evaluaci je třeba vyhodnotit obě složky zvlášť i dohromady, aby šlo posoudit celkovou efektivitu systému.

Základní metodou je samozřejmě ruční vyhodnocení získaných odpovědí souborem testovacích uživatelů. Nejběžnější pokročilou technikou, jež se nemusí spoléhat na lidský faktor, je Zero-Shot LLM evaluace, na které je postavený i framework RAGAs (Retrieval-Augmented Generation Assessment)¹⁰. Právě RAGAs se dá pokládat za základ evaluace RAG systémů, hodnotí na základě těchto metrik:

- věrohodnost,
- relevantnost odpovědi,
- vyvolání kontextu,
- přesnost kontextu,
- relevance kontextu,
- kontextové vybavování entit.

Pro vyhodnocení je potřeba mít sadu testovacích dat obsahující otázky, kontexty pro odpověď a kontrolní očekávané odpovědi (ground-truth), k čemuž lze využít i samotný RAGAs, jelikož tento framework aplikuje moderní metody jako využití LLM ke generování syntetických dat. Na základě těchto dat se pak hodnotí, zda systém vyhledává relevantní úryvky a odpovídá, jak se od něj očekává.

RAGAs rozumně interpretuje výsledky různých přístupů používaných v RAG systémech, což umožňuje průběžné zlepšování jejich výkonnosti. Využívá vizuálního zobrazení a komplexní analýzy k efektivnímu porovnání a pochopení různých konfigurací výkonnosti.

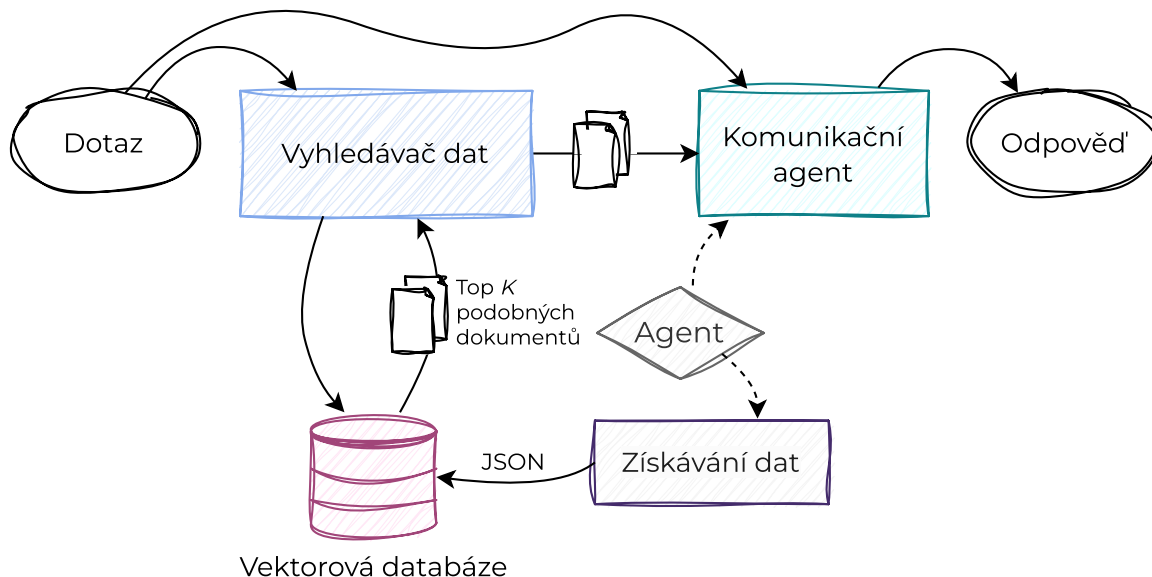
¹⁰<https://docs.ragas.io/en/latest/index.html>

Kapitola 4

Návrh a implementace

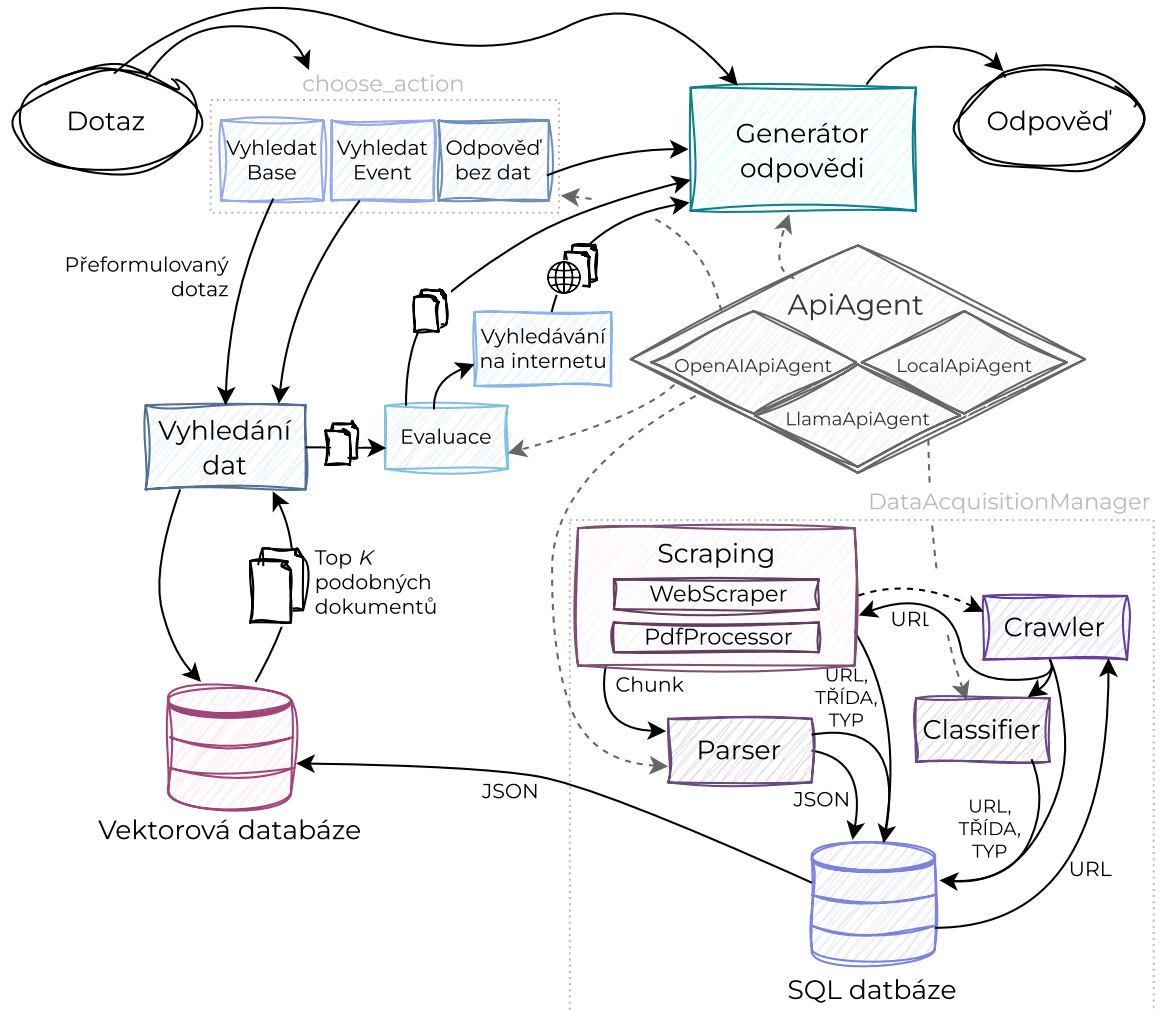
Cílem této práce je vytvořit inteligentního asistenta, který dokáže odpovědět na otázky v souvislosti s městem Brnem a poskytovat relevantní informace s ohledem na konkrétní potřeby uživatelů. Agent není určen pouze turistům, ale měl by být schopen odpovídat i na dotazy ohledně administrativy a sloužit i běžným obyvatelům města Brna. Bez ohledu na to, zda se jedná o návštěvníka hledajícího tipy na zajímavá místa, nebo místního občana zajímavajícího se o aktuality či kulturní události, agent by měl disponovat informacemi z každého odvětví. Příklady použití vzorovými uživateli jsou k nahlédnutí v příloze A. Příklady dotazů, na které by měl být agent schopen odpovědět, lze nalézt v příloze B.

Aby bylo zajištěno, že má systém komunikačního agenta k dispozici potřebné informace, je postaven na bázi RAG, tedy spojení jazykového modelu a vektorové databáze. Navíc obsahuje komponenty pro získávání dat. Zjednodušený návrh systému je vyobrazen ve schématu na obrázku 4.1. Každá komponenta se však doopravdy skládá z dalších komponent, čímž vzniká komplexní celek vyobrazený na schéma v obrázku 4.2.



Obrázek 4.1: Zjednodušená architektura systému.

Získávání dat sestává z rozšiřování url adres, stahování obsahu webových stránek a PDF souborů, klasifikace a parsování, přičemž se získaná data spolu s dalšími parametry ukládají do SQL databáze. Data jsou následně zpracována a uložena i do vektorové databáze. Agent pro komunikaci s LLM je pro změnu abstraktní třídou nad třídami podporujícími různé služby s velkými jazykovými modely. K vyhledávání dat zase náleží volba postupu pro generování odpovědi, evaluace a vyhledávání na internetu. Každé z těchto částí se podrobně věnuji v následujících podkapitolách. Nakonec srovnávám výkonnost implementovaného systému na různých jazykových modelech.



Obrázek 4.2: Detailní architektura systému.

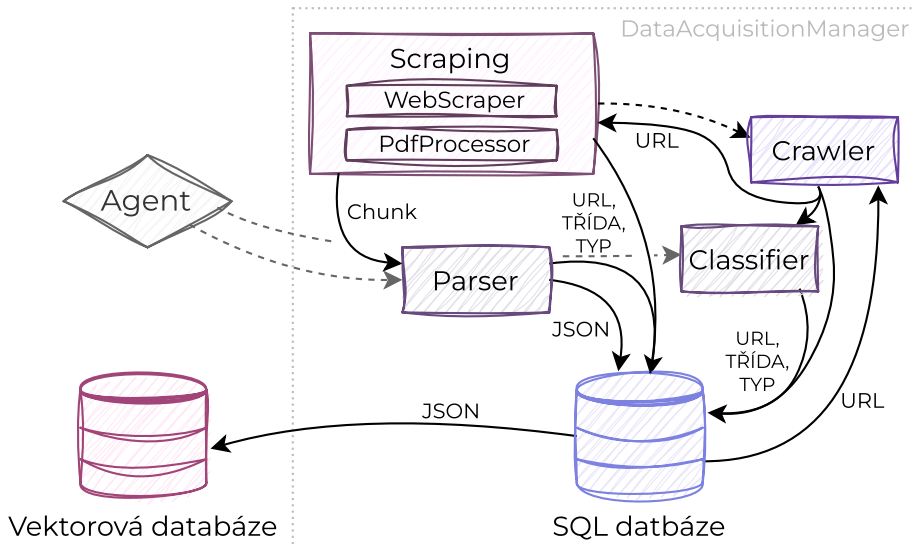
4.1 Získávání dat

Získávání a zpracování dat v této práci zapouzdřuje modul `data_acquisition`. Celý proces je řízen třídou `DataAcquisitionManager`. Zde jsou implementovány metody řídicí získávání a aktualizování obsahu, tak aby byla data pro komunikačního agenta vždy co nejaktuálnější.

Pro získání potřebných dat o městě Brně, místech, akcích i administrativě týkající se Brna byl sestaven seznam relevantních url adres, tyto url adresy jsou k vidění v příloze C a slouží jako informační základ aplikace. Jsou uloženy spolu s dalšími podstatnými parametry v SQL databázi, což je popsáno v podkapitole 4.2.

Proces získávání dat je vyobrazen na schéma v obrázku 4.3 a jeho popis je následující:

1. Třída `WebCrawler` získává z url nové rozšiřující url, a určuje, zda se jedná o stránku s užitečným obsahem, stránku pro sběr rozšiřujících adres, či o irelevantní stránku.
2. Získané url klasifikované jako adresy pouze pro získání dalších url (`crawl_only`) jsou podle obsahu klasifikovány pomocí modulu `content_classification` a spolu s dalšími parametry uloženy do SQL databáze.
3. Stránky které nebyly označeny jako irelevantní či `crawl_only` jsou dále zpracovány:
 - (a) Jedná-li se o relevantní webový obsah, který je nový, nebo pozměněný (to se detekuje diferencí kódovaného obsahu), `WebScraper` zpracovává HTML obsah na souvislé kusy textů.
 - (b) V případě PDF souboru dochází ke zpracování modulem `pdf_processor`, jehož výstupem jsou též souvislé kusy textů.
4. Tyto texty následně `DataAcquisitionManager` předává modulu `content_parsing`, který vrací text strukturovaný do JSON podoby. Takto jsou texty uloženy do tabulky `parsed_sources` SQL databáze.



Obrázek 4.3: Schéma procesu získávání dat.

Systém se soustředí na sběr dat zejména v anglickém jazyce, ale počítá i s daty v češtině. Protože pokusy s vícejazyčným embedding modelem ukázaly podstatně horší výsledky, než u modelu čistě pro angličtinu, jsou české texty v kroku zpracování dat přeloženy do anglického jazyka. Kvůli šetření zdrojů k tomu není využíván speciální model pro překlad, ale přímo LLM, které obsah zpracovává, jak popisuje podkapitola 4.6.1 o parsování obsahu.

Pro zhodnocení metod zpracování dat jsem vytvořila evaluační datovou sadu sestávající ze 100 úryvků textů. Každý úryvek je doplněn o URL adresu jeho zdroje a třídu, do které by měl být klasifikován. V celé sadě jsou třídy zastoupeny rovnoměrně, ke každé náleží přesně 25 úryvků.

4.1.1 Interval obnovy

Bez pravidelných aktualizací může systém poskytovat zastaralé nebo nepřesné informace, což vede ke snížení spokojenosti uživatelů a důvěry ve schopnosti systému. Proto je pro udržení účinnosti a relevance RAG nezbytné průběžně obnovovat data, které systém používá. Zejména v případě kulturních akcí je mimo aktualizaci obsahu zároveň potřeba neustále získávat informace o nových událostech.

| Třída | Interval | CRON příkaz |
|----------------|----------|---|
| event | denně | 0 0 * * * bash \$FILES_DIR/cron_event.sh |
| administration | týdně | 0 0 * * 1 bash \$FILES_DIR/cron_administration.sh |
| place | měsíčně | 0 0 1 * * bash \$FILES_DIR/cron_place.sh |
| static | půlročně | 0 0 1 1,7 * bash \$FILES_DIR/cron_static.sh |
| pdf | ročně | 0 0 1 11 * bash \$FILES_DIR/cron_pdf.sh |

Tabulka 4.1: Interval obnovy obsahu podle třídy a příslušný CRON příkaz

K řízení aktualizaci dat v předem definovaných intervalech jsem se rozhodla použít CRON úlohy, interval je definovaný na základě třídy dat, jak popisuje tabulka 4.1. Každá třída má jiné potřeby, CRON úlohy automatizují provádění skriptů v určených časech a zajišťují, že jsou data aktualizována.

Obsah je dělen do tříd (proč tomu tak je popisují podkapitoly 4.5 a 4.6) a pro každou třídu existuje zvlášť skript, který obsahuje příkaz pro spuštění aktualizace/sběru dat. Tento příkaz obsahuje i potřebné lokální proměnné, které je pro správné fungování nutné přizpůsobit dle zařízení.

Po spuštění příkazu jsou procházeny stránky, na kterých byla zaznamenána příslušná třída obsahu, a je detekováno, zda na nich od posledního průchodu došlo ke změně. To je implementováno jako porovnávání *sha256* kódovaného obsahu s hexadecimálním kódem, který je již uložen v SQL databázi. V případě, že ke změně došlo, či se jedná o nový obsah, dochází k následovnému zpracování obsahu, tak jak popisují v podkapitole 4.1.3 o získávání webového obsahu. Získaný obsah je následně přidán do vektorové databáze. Pokud se ve vektorové databázi již nachází záznam s danou url, je původní obsah nahrazen novým.

4.1.2 Procházení webového obsahu

Jelikož je pro automatizovaný sběr dat potřeba rozšiřovat a aktualizovat zdrojové url, bylo nutné implementovat i *WebCrawler*. Díky němu je SQL databáze adres rozšiřována o relevantní url adresy nalezené na již známých stránkách. *WebCrawler* používá pro získání pseudo-vyčištěného zdrojového HTML (bez skriptů, stylů, obrázků apod.) pro sběr url adres *WebScaper* popsany v podkapitole 4.1.3.

Implementace univerzálního webového prohlédače je vzhledem k rozsáhlé a neustále se měnící povaze internetu náročným úkolem. Naštěstí se tato práce soustředí jen na určitý typ webového obsahu, a to stránky obsahující textové informace relevantní k městu Brnu.

`WebCrawler` následuje jen ty adresy, které vedou na další HTML obsah, vyhýbá se médiím a sociálním sítím. Protože vím, jaká data mají být sbírána, dochází k filtrování obsahu – aby byla stránka zahrnuta, musí na ní být zmínka o Brně. Obsah je tříděn i na základě samotných url adres, pokud adresa obsahuje zakázaný podřetězec (zakázané podřetězce jsou definovány v konfiguračním souboru a obsahují vysledované nechtěné části adres, jako *jobs*, *newsletter*, *shop* apod.).

Výsledkem zpracování stránky tímto modulem je seznam rozšiřujících url adres, které odpovídají požadavkům tohoto systému. Dalším krokem je zde pokročilé filtrování obsahu, k čemuž lze využít LLM, aby podle předem definovaných parametrů určil, zda je obsah dostatečně relevantní pro cílovou aplikaci.

4.1.3 Získávání webového obsahu

Modul `web_scraper` se zase soustředí na url adresy, které nejsou určeny pouze pro crawling. Pro získávání obsahu se nabízí LlamaIndex `SimpleWebPageReader`¹, který poskytuje funkce pro načítání dat z webových stránek a převod obsahu HTML na prostý text, pomocí knihovny `html2text`. Od prvotního plánu použít tuto knihovnu jsem však velmi rychle upustila, a to hned z několika důvodů:

1. Nástroj vrací celý obsah stránky jako jeden nestrukturovaný dokument – dochází ke ztrátě hierarchických informací
2. V textu zůstává spousta neužitečného balastu (jako názvy souborů obrázků, které se na stránce vyskytovaly, cookies štítky nebo odkazy na videa), který jen tvoří šum v informacích
3. Funkcionalita přidání vlastních metadat nefunguje správně a způsobuje tak problémy

Nástroj sice umožňuje i jeho použití pro získání čistého zdrojového HTML kódu stránky, ale pak není žádná výhoda jeho použití, jelikož jej k tomu není potřeba. Aktuálně (leden 2024) se mi nepodařilo nalézt žádný nástroj, ke kterému by nebyla potřeba placeného API klíče, jenž by vyhovoval požadavkům projektu. Proto jsou potřebné nástroje v této práci implementovány přímo.

Získání obsahu

K získání zdrojového HTML kódu stránek je použita knihovna `requests`. Pro zabránění detekci a blokování webových stránek je klíčová rotace hlaviček, k tomu se zase využívá `UserAgent` knihovny `fake_useragent`. Generováním náhodných `User-Agent` řetězců při každém požadavku napodobuje scraper různé prohlížeče a zařízení, což webovým stránkám ztěžuje rozpoznání a blokování automatizované činnosti scraperu. To zvyšuje schopnost scraperu přistupovat k datům z různých zdrojů a získávat je z nich, aniž by byl označen jako podezřelý nebo porušoval podmínky služby webových stránek.

¹https://docs.llamaindex.ai/en/v0.9.48/api/llama_index.readers.SimpleWebPageReader.html

Po získání HTML následuje krok čištění obsahu, jehož snaha je zbavit výsledný text nepotřebného šumu. `WebScrap`er disponuje metodami které s použitím knihovny `BeautifulSoup` krok po kroku odstraňují nepotřebné elementy. Jelikož vím, jak by měl výsledný text vypadat (znázorněno na obrázku 4.4), můžu rovnou z HTML odstranit styly, a skripty, média (obrázky a videa), navigační prvky, citace, neviditelné prvky, prvky spojené s cookies a sociálními sítěmi, správou uživatelů apod. Pro různé typy obsahu existují v HTML různé značky (tag), nebo je jejich určení pojmenováno v názvu třídy prvku. Pro detekci a odstranění/rozbalení nechtěných prvků se používá právě kombinace extrakce podle typu značky i vzorů v názvech třídy (k tomu se využívá regex).

Zároveň jsou v třídě `WebScrap`er implementovány metody, které detekují nepotřebné hierarchické celky, a pro získání obsahu (ne získání url adres), je z obsahu stránky odstraňují. Vhodným příkladem je třeba box s url adresami a popisky, jenž demonstruje obrázek 4.5, které často fungují jako rozcestník pro další obsah stránek, jejich textový obsah je však pro získávání čistého obsahu stránky irelevantní.

Za zmínku stojí i kódovací metoda `_hash_text`, která hashuje text pomocí algoritmu SHA-256 a vrací jeho hexadecimální reprezentaci. To je velmi užitečné pro detekci, zda v obsahu došlo od ke změně vůči již zpracované verzi a je potřeba ho zpracovávat znovu. V databázi není potřeba držet celý text, což výrazně pomůže nárokům na paměť – např. článek s Wikipedie o městě Brně je už po vyčištění delší než 45 000 znaků, jeho zakódovaná reprezentace je však pouze 64 znaků dlouhá. Pro detekci změn pak jen stačí porovnat kód aktuální stránky s kódem z databáze.

Rozdělení na části (chunking/splitting)

Chunking označuje proces rozdělení rozsáhlých textových dokumentů na lépe zpracovatelné části neboli *chunky*. To je v RAG systémech důležité pro efektivní vyhledávání – menší, dobře definované, části textu lze indexovat a vyhledávat efektivněji než velké dokumenty, jak bylo zmíněno i v podkapitole 3.2.1. Díky rozdělení dokumentů na části se systém může při generování odpovědi zaměřit na nejrelevantnější části textu. Nedochozí tak k rozmělnění relevantních informací, k čemuž by mohlo dojít, bral-li by se v úvahu celý dokument, zejména u dokumentů, kde jsou pro daný dotaz relevantní pouze části.

Bohužel neexistuje univerzální nejlepší velikost indexovaných částí. Optimální velikost totiž závisí na povaze dat (u textů, které jsou bohaté na informace je vhodnější použít menší části, aby se neztratily nuance a detaily, zatímco pro jednodušší texty jsou vhodnější hrubší části, aby by se neztratil kontext) i účelu RAG systému (systémy určené pro podrobné dotazy mohou potřebovat menší části, aby zachytily konkrétní informace, zatímco pro systémy, které odpovídají na obecnější dotazy, jsou vhodnější větší indexované části). Velikost indexovaných částí samozřejmě závisí i na maximální délce tokenů pro vstupní sekvence LLM.

Velmi oblíbeným přístupem je teď indexování po větách s překrytím, jak popisuje *Chen a spol.* [6]. To je však podle mě vhodné pro texty, ve kterých tolik nezáleží na kontextu. V tomto projektu je kontext klíčový, jelikož se na každé stránce vyskytují trochu jiné informace a např. v případě kulturních akcí, je nutné vědět jaké akce se informace o místě a času konání týká. Proto jsem se rozhodla využít k rozdělení na části hierarchické informace HTML a dělit text na ucelené části.



Koncert vstupenky

Zážitek ve virtuální realitě

● 30/04/2022-31/03/2025

Muzeum Josefa Hoffmanna

Umění - Výstavy

Virtuální rekonstrukce první prodejny uměleckořemeslných děl Wiener Werkstätte, která vznikla ve spolupráci Rakouského muzea užitého umění MAK a firmy FILMBÄCKEREI FASOLT-BAKER KG a měla svou premiéru ve vídeňském muzeu MAK, umožňuje návštěvníkům imerzivní cestu časem do interiéru z raného období vídeňské moderny. V precizně rekonstruované virtuální prodejně si lze zblízka prohlédnout předměty vyrobené podle návrhů Josefa Hoffmanna a Wiener Werkstätte.

Cílová oblast



MOHLO BY SE VÁM TAKÉ LÍBIT



David Böhm - Jiří Franta SIDE-EFFECT

● 05/10/2022

Tisková zpráva

Tisková zpráva ke stažení zde.
Fotografie ke stažení zde.
Fotografie určené ke stažení jsou k dispozici pouze pro novináře/ novinářky a jejich použití je povoleno jen v souvislosti...



Dušan Jurkovič. Architekt a jeho dům

Jurkovičova vila

Stálá expozice

Vlastní vila Dušana Jurkoviče z roku 1906 patří k vrcholným příkladům modernistické architektury inspirované britskou a vídeňskou tvorbou s prvky lidového umění nejen v Brně. Výstava...



ART DESIGN FASHION

Uměleckoprůmyslové muzeum

Stálá expozice

Budova Uměleckoprůmyslového muzea Moravské galerie v roce 2021 získala nový koncept: ART DESIGN FASHION. Především jim, že muzeum se věnuje nejen samotnému designu, ale...

Obrázek 4.4: Cílová část obsahu ze stránky z moravska-galerie.cz.

Podobná místa v okolí

| | | |
|---|--|---|
|  |  |  |
| STARÁ RADNICE → | LABYRINT POD ZELNÝM TRHEM → | KOSTNICE U SV. JAKUBA → |
| Jedna z nejpozoruhodnějších historických budov v centru. | Tajemné podzemní bludiště nejstarších sklepních prostor v Brně | Počtem nalezených ostatků druhá největší v Evropě, hned po té pařížské |

MOHLO BY SE VÁM TAKÉ LÍBIT

| | | |
|---|---|---|
|  |  |  |
| David Böhm – Jiří Franta SIDE-EFFECT | Dušan Jurkovič. Architekt a jeho dům | ART DESIGN FASHION |
| ● 05/10/2022 | | |
| Tisková zpráva | Jurkovičova vila | Stálé expozice |
| Tisková zpráva ke stažení zde. Fotografie ke stažení zde. Fotografie určené ke stažení jsou k dispozici pouze pro novináře/ novinářky a jejich použití je povoleno jen v souvislosti... | Vlastní vila Dušana Jurkoviče z roku 1906 patří k vrcholným příkladům modernistické architektury inspirované britskou a vídeňskou tvorbou s prvky lidového umění nejen v Brně. Výstava... | Budova Uměleckoprůmyslového muzea Moravské galerie v roce 2021 získala nový koncept: ART DESIGN FASHION. Předesílá jím, že muzeum se věnuje nejen samotnému designu, ale... |

Obrázek 4.5: Box s popisky odkazujícími na jiné stránky z konce stránky z gotobrnno.cz (nahore) a konce stránky z moravska-galerie.cz (dole).

Implementované metody třídy `WebScrapper` dělí text na sekce a podsekce podle nadpisů, přičemž se dbá na maximální počet znaků jedné části, kterou lze zadat a zajistit tak dodržení maximální délky tokenů pro vstup LLM.

4.1.4 Zpracování PDF

System je schopen zpracovávat i PDF, a to jak soubory uložené lokálně, tak ty, které jsou součástí crawlingu – ty se získávají s pomocí knihovny `requests`. Celé zpracování PDF souborů zapouzdřuje modul `pdf_processor`.

Získání obsahu z PDF souborů

Zpočátku byl k extrakci textu z PDF souborů používán nástroj Apache Tika². Převažujícím zdrojem obsahu jsou však webové stránky, nikoliv soubory PDF, takže používání nástroje Tika nebylo tak výhodné (použití vyžaduje samostatný provoz Tika serveru, což přidává další vrstvu složitosti do architektury systému). Proto jsem se rozhodla pro hledání alternativního řešení.

Implementace přešla na používání Llamaindex parseru PDF – LlamaParse³, který byl představen 20. února 2024⁴. LlamaParse je součástí služby LlamaCloud, a umožňuje analyzovat soubory PDF na strukturovaná data. Je k dispozici jako samostatné rozhraní REST API, Python balíček a webové uživatelské rozhraní. V současné době (květen 2024) je ve veřejné beta verzi.

V projektu se LlamaParse používá prostřednictvím Python balíčku, který se připojuje k LlamaCloud API. Pro jeho fungování je nutné mít osobní API klíč (zadaný v prostředí projektu), v neplacené verzi je zdarma zpracování 1000 stran denně (což pro tuto práci bohatě stačí), po vyčerpání 1000 stránek zdarma denně je následná cena 0,003 \$ za každou stránku (v přepočtu na CZK se jedná o cca 0.07 Kč za stránku).

LlamaParse umožňuje jako výstup extrakce PDF buďto prostý text, nebo strukturovaný text ve formátu markdown. Protože převodem na prostý text dojde ke ztrátě hierarchických informací, zvolila jsem jako výstup z LlamaParse markdown.

Rozdělení na části (chunking/splitting)

I text PDF je samozřejmě nutné rozdělit na malé části, které se dále budou zpracovávat pro vstup do vektorové databáze. Díky tomu, že je text strukturovaný, lze ho podobně jako u HTML obsahu webových stránek 4.1.3 rozdělit podle nadpisů a podnadpisů apod. na sounáležitě kousky až do maxima zadaného počtu znaků.

Jako nejjednodušší nástroj pro rozdělení textu na části mě prvně zaujal LangChain MarkdownTextSplitter. Při jeho použití v této práci však docházelo k odstraňování formátování a zalomení řádků, což značně komplikovalo další zpracování a formátování textu. Nepodařilo se mi najít žádnou rozumnou alternativu, a tak jsem nakonec implementovala vlastní chunking metody, které text dělí přesně podle potřeb projektu podle nadpisů na části maximálně zadaného počtu znaků. Spadají pod třídu PdfProcessor a je možné jak jednotlivé zpracování, tak asynchronní zpracování dávky souborů.

4.2 SQL Databáze

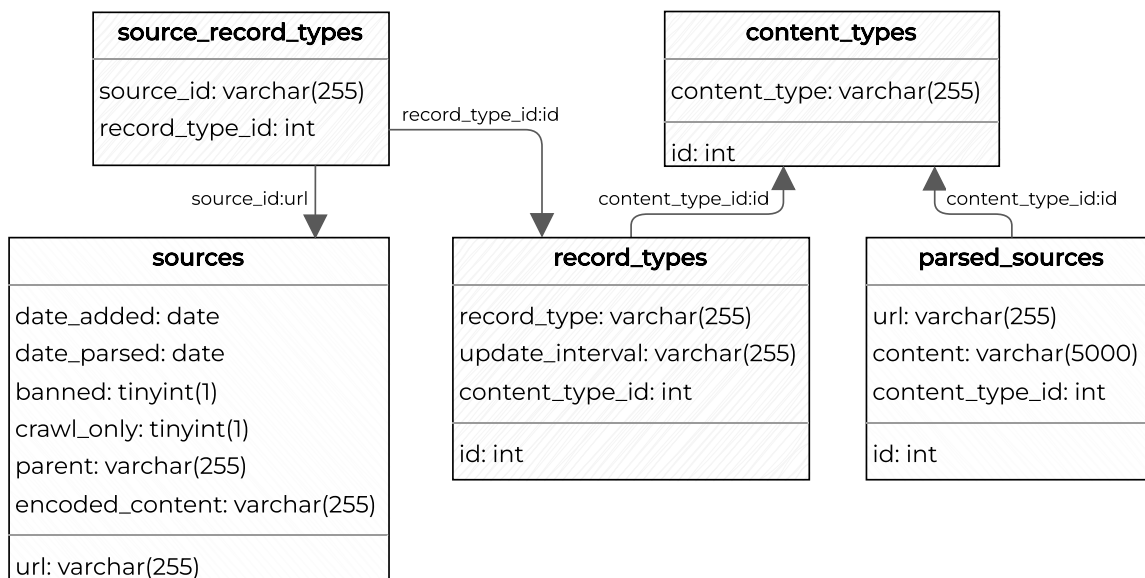
Při získávání obsahu je potřeba perzistentně ukládat mnohé parametry, k tomu účelu slouží SQL databáze `agent_sources`. Pro fungování je potřeba lokálně běžící MySQL⁵ databáze. Celkem se databáze skládá z pěti tabulek, jejich vztah je znázorněn na obrázku 4.6 a popsán v následujícím textu.

²<https://tika.apache.org/>

³<https://cloud.llamaindex.ai/>

⁴<https://www.llamaindex.ai/blog/introducing-llamacloud-and-llamaparse-af8cedf9006b>

⁵<https://dev.mysql.com/doc/mysql-getting-started/en/>



Obrázek 4.6: Schéma SQL databáze.

Přístup k databázi a práci s ní zprostředkovává implementovaná třída `SourcesDB` využívající přístup objektově relačního mapování (ORM, Object-Relational Mapping). Tato třída poskytuje metody pro vkládání a aktualizaci záznamů a typů, i vkládání zdrojů ze souborů CSV (protože je proces sběru dat časově/finančně náročný, je v projektu uchovávan základní rámec zdrojů pro vytvoření SQL databáze v CSV souborech).

Základem je tabulka `Sources`, do které jsou ukládány url adresy, na které crawler narazí, spolu se všemi potřebnými parametry:

- `url` – primární klíč, url adresa stránky,
- `date_added` – datum přidání/aktualizace url adresy,
- `date_parsed` – datum parsování,
- `banned` – ukazatel, zda-li není stránka nevyžádaná,
- `crawl_only` – ukazatel, zda-li se jedná jen o stránku pro získání dalších url adres,
- `parent` – adresa rodiče určená pro detekci potřeby získání adres i z navigace stránky,
- `encoded_content` – kódovaný obsah stránky pro pozdější detekci změn obsahu.

Z každého zdroje (url adresa) je získáno až několik ucelených textů, jak bylo popsáno v [4.1.3](#), přičemž každý z těchto kusů může být jiné třídy (většinou jsou na stránce texty stejné třídy, ale v několika případech jsou i třídy dvě, a to je nutné brát v potaz). Vztah mezi zdroji a třídami reprezentuje tabulka `source_record_types`. Jak tomu samo napovídá, tabulka `record_types` zase uchovává třídy.

Rozlišovaných tříd obsahu pro crawling/scraping je 5, ale pro další zpracování stačí rozlišovat jen 1) kulturní akce a 2) vše ostatní, protože vektorová databáze má pouze tato dvě schémata. Tento vztah je reprezentován hodnotou `content_type_id` v tabulce

`record_types`, a odkazuje na hodnoty tabulky `content_types`, jenž uchovává právě typy korelující se schémata ve vektorové databázi.

Posledním prvkem SQL databáze je tabulka `parsed_sources`, jejím úkolem je přímo držení dat pro vektorovou databázi – `content` slouží přímo pro parsovaný JSON.

Zde se nabízí migrovat celou databázi na NoSQL, jako je například MongoDB⁶, aby bylo možné v JSON datech vyhledávat a filtrovat, což je užitečnou optimalizací, avšak aktuálně to není potřeba. SQL databáze je důležitým prvkem celého systému, jen díky ní je možné získávání dat oddělit od ostatních komponent komunikačního agenta.

4.3 Agent

Alfou i omegou tohoto projektu je tzv. agent, jehož úkolem je komunikace s LLM. Knihovny jako již zmíněný LangChain či LlamaIndex nabízejí takovéto agenty, kteří přinášejí usnadnění a zrychlení vývoje. Začala jsem pracovat na projektovém agentovi založeném na funkcích knihovny LangChain. Avšak kvůli potřebě abstrakce jakéhokoliv modelu s možností spolehlivého volání funkcí jsem se použití agenta z knihovny musela vzdát. Knihovní agenti nabízejí pochopitelně jen pevně danou sadu funkcí, což omezuje přizpůsobivost a rozšiřitelnost, což je v tomto projektu klíčové.

Aktuálně (leden 2024) bohužel neexistuje celistvý způsob použití LLM z různých API s podporou volání funkcí identickým způsobem, ani s knihovnou, ani bez ní. Ale existuje alespoň způsob komunikace s LLM, který je podporován napříč většinou služeb, a to je OpenAI API request⁷, jehož základní podobu při volání službou CURL lze vidět ve výpise 4.1. OpenAI nabízí klienta, který zprostředkovává připojení k LLM serveru a tento klient je kompatibilní nejen s OpenAI API, ale i Llama API⁸ a službami pro lokální spouštění open-source LLM jako jsou Ollama⁹ či LM Studio¹⁰.

```
curl https://api.url
-H "Content-Type: application/json"
-H "Authorization: Bearer $API_KEY"
-d '{
  "model": "model_name",
  "messages": [{"role": "user", "content": "user_message"}],
  "tools": [{"type": "function", "function": {function_name_and_properties}},
  "tool_choice": "auto"
}'
```

Výpis 4.1: Příklad CURL dotazu v OpenAI API formátu.

V Pythonu lze pro komunikaci s API použít klienta prostřednictvím knihovny `openai`. Klientovi jsou pak předány potřebné parametry, tak jako je tomu u CURL, on se spojí s API a vytvoří odpověď ve formátu `chat completion object`, jehož příklad s voláním funkce je k vidění ve výpise 4.2. Parametry dotazů i odpovědí popisuje již zmíněná oficiální dokumentace klienta.

⁶<https://www.mongodb.com/>

⁷<https://platform.openai.com/docs/api-reference/introduction>

⁸<https://www.llama-api.com/>

⁹<https://ollama.com/>

¹⁰<https://lmstudio.ai/>


```

{
  "id": "chatcmpl-abc123",
  "object": "chat.completion",
  "created": 1699896916,
  "model": "model_name",
  "choices": [
    {"index": 0,
     "message": {
       "role": "assistant",
       "content": null,
       "tool_calls": [{
         "id": "call_abc123",
         "type": "function",
         "function": {"name": "chosen_function_name",
                       "arguments": "{function_arguments}"}}]},
     "finish_reason": "tool_calls"}],
  "usage": {usage_information}
}

```

Výpis 4.2: Příklad odpovědi s voláním funkcí v OpenAI API formátu.

Základem flexibilní architektury implementace agenta pro interakci s různými rozhraními API je abstraktní třída `ApiAgent`, která právě klienta `openai` využívá. Díky zapouzdření všech funkcí souvisejících s rozhraním API do této abstraktní třídy z ní může dědit jakákoli konkrétní implementace rozhraní API a přitom zachovat konzistenci metod pro komunikaci s LLM a tudíž i volání funkcí. Tento návrh tedy umožňuje používat funkce pod stejnou abstraktní třídou a zároveň splňovat jedinečné požadavky pro komunikaci s jednotlivými API.

Při komunikaci s velkým jazykovým modelem je důležité si uvědomit, že model nechová historii konverzace, ale reaguje na základě poslední přijaté zprávy. To znamená, že každá zpráva slouží jako nezávislý vstup a model nemá paměť ani na předchozí interakce v rámci konverzace. Z tohoto důvodu se modelu předává list zpráv, přičemž ve zprávách se mění role mezi:

- a) user – příkazy zadané uživatelem, na které má model reagovat,
- b) assistant – odpovědi nebo další obsah poskytnutý modelem,
- c) system – instrukce, které jsou určeny k řízení chování a výkonu LLM.

Podle toho, kdo zprávu poslal. Součástí implementace je i abstraktní třída `Message` s podtřídami `UserMessage`, `AssistantMessage` a `SystemMessage`, které se starají o správný formát jednotlivých zpráv. `ApiAgent` implementuje i jednoduchou historii konverzací prostřednictvím listu provedených zpráv, ke kterému se zprávy přidávají. Tento list je automaticky zkracován při přesažení zadané velikosti kontextového okna modelu, do budoucna by však bylo vhodné přidat do správy historie systém sumarizace zpráv.

Ve třídě `ApiAgent` je definováno několik metod pro interakci s rozhraním API, jehož základem jsou funkce ve výpise 4.3. Metoda `get_base_response` slouží pro běžnou otázka-odpověď komunikaci, metoda `get_base_call_response` přidává do požadavku schéma funkcí ze kterých má model na výběr pro tzv. *function calling*.

Zde je nutno podotknout, že *function calling* není úplně trefné pojmenování, protože k žádnému volání funkcí reálně nedochází, ale LLM jen vrací odpověď obsahující název vybrané funkce a schéma s parametry. Metoda `get_forced_call_response` pak nutí model odpovědět ve formě schéma předané funkce, jelikož standardně model nemusí volání funkcí použít a může odpovědět, jak uzná za vhodné.

```
def get_base_response(self, messages):
    response = self.client.chat.completions.create(
        model=self.model_name,
        messages=messages,
        stream=False
    )
    return response

def get_base_call_response(self, messages, functions_schemas):
    response = self.client.chat.completions.create(
        model=self.model_name,
        messages=messages,
        functions=functions_schemas,
        stream=False
    )
    return response

def get_forced_call_response(self, messages, schema, function_call):
    response = self.client.chat.completions.create(
        model=self.model_name,
        messages=messages,
        functions=schema,
        stream=False,
        function_call=function_call,
    )
    return response
```

Výpis 4.3: Základní funkce agenta pro komunikaci s LLM.

Není nutné vše implementovat na míru, existují-li nástroje, které s tím pomohou. Callable funkci je nutné převést na kompatibilní schéma, které je modelu předáno, a k tomu se používá `convert_to_openai_tool`¹¹ z knihovny LangChain.

Hlavní metody jako `get_function_call` či `get_forced_function_call` zmíněné metody využívají a zakomponovávají pravidla pro zpracování chyb, opakování neúspěšných volání, formátování odpovědí a hlavně převod odpovědi na reálné volání zvolené funkce. Volání metod ostatním modulům zprostředkovávají tyto podtřídy:

- `OpenAIApiAgent` – určená pro komunikaci s modely společnosti OpenAI,
- `LlamaApiAgent` – vymezená pro komunikaci s LlamaAPI,
- `LocalApiAgent` – sloužící pro komunikaci s jakoukoliv lokální službou pro spouštění LLM, která podporuje OpenAI klienta.

¹¹https://python.langchain.com/v0.1/docs/modules/model_io/chat/function_calling/

Tyto podtřídy pokrývají celý základ možností komunikace s LLM, každá implementuje úpravy hlavních funkcí tak, jak je to pro dané API potřeba. `LocalApiAgent` mezi nimi vyniká celkovou dodatečnou implementací metod přizpůsobených pro volání funkcí lokálními modely, odlišných od volání prostřednictvím externích rozhraní API, jelikož lokální modely aktuálně volání funkcí vlastně nepodporují.

Zde se o výstup pro volání funkce stará metoda `get_json_format_response`, která je přizpůsobena k získání odpovědi lokálního modelu v požadovaném formátu JSON. Využívá knihovny `instructor` a `Pydantic`, aby donutila model odpovědět parametry pro zavolání funkce přesně v požadovaném formátu, kterým je zde model `Pydantic BaseModel`. Tuto metodu pak výše zmíněné metody používají místo `get_base_call_response`. Samozřejmě se zde nachází i přizpůsobené metody pro porovnávání parametrů, zpracování chyb, rekurzivní volání a převod `Callable` funkcí na `Pydantic BaseModel`.

Agent se opírá o téměř neměnný standard `OpenAI API`, což snižuje potřebu neustálého přizpůsobování se rychlým posunům v oblasti, není potřeba reagovat na změny v knihovnách, které připojení k LLM zprostředkovávají. Navíc agent disponuje na míru přizpůsobenými metodami, jako je metoda pro volání funkcí s přidaným vlastním popisem, což je užitečné v pozdější fázi projektu při experimentech s proměnnými v popisu funkcí.

4.4 Používané velké jazykové modely

Díky všestrannosti agenta lze používat široký výběr velkých jazykových modelů. Jako vedlejší cíl práce jsem si stanovila srovnání LLM na různých úlohách. Open-source modely jsou spouštěny na KNOTIS serveru `Athena20`, jehož vybrané specifikace ukazuje tabulka 4.2, pomocí služby `Ollama`. K serveru se připojuji pomocí `SSH tunelingu`, což je metoda směřování provozu přes zabezpečené šifrované připojení `SSH`, které vytváří bezpečnou cestu pro odesílání a přijímání dat mezi dvěma koncovými body. Tímto způsobem to v funguje stejně, jako by server běžel lokálně na mém počítači.

| RAM [GiB] | CPU model | název GPU | paměť GPU [MiB] |
|-----------|---|------------------|-----------------|
| 187.6 | Intel(R) Xeon(R) Silver 4214R CPU @ 2.40GHz, 24 cores | NVIDIA RTX A5000 | 24564, 24564 |

Tabulka 4.2: Vybraná specifikace KNOTIS serveru `Athena20`.

Původně byl Agent vyvíjen a testován na modelech `gpt-3.5-turbo-1106`, `llama2-70b` a `mixtral8x7b`. Model `mixtral8x22b` se mi na školním severu bohužel nepodařilo spustit, jinak by byl do testování též zařazen.

`OpenAI` však vydalo model `gpt-3.5-turbo-0125` s vyšší přesností při odpovídání v požadovaných formátech a s opravou chyby, která způsobovala problém s kódováním textu u volání funkcí v jiných než anglických jazycích (což bylo v práci problémem jehož řešení bylo v agentovi implementováno). Zároveň došlo k představení modelů `llama3` společností `Meta`.

Výběr modelů jsem přizpůsobila těmto změnám a zvolenými modely pro testování a srovnání jsou `mixtral-8x7B-instruct-v0.1`, `lama3-8b`, `llama3-70b`, `llama2-70b`, `gpt-3.5-turbo-0125`

| Model | Typ | Počet parametrů | Velikost |
|------------|-----|-----------------|----------------------|
| mixtral | 7B | 46.70 B | 24.62 GiB (4.53 BPW) |
| llama3-8b | 7B | 8.03 B | 4.33 GiB (4.64 BPW) |
| llama3-70b | 70B | 70.55 B | 37.22 GiB (4.53 BPW) |
| llama2-70b | 70B | 68.98 B | 36.20 GiB (4.51 BPW) |

Tabulka 4.3: Srovnání parametrů open-source modelů.

a gpt-4-turbo. První 4 zmíněné modely jsou open-source, spouštím je s 4bitovou kvantizací pomocí služby Ollama. Používané llama modely jsou instruct verze laděné pro dialog.

Parametry těchto open-source modelů srovnává tabulka 4.3, pro gpt modely se mi tyto informace najít nepodařilo. Kdykoliv se v následujícím textu odkazují na nějaký z těchto modelů, je myšlena právě zmíněná verze.

4.5 Klasifikace

Rozdělení shromážděného obsahu do tříd je zásadní vzhledem k přirozené rozmanitosti obsahu a potřebě nastavit odlišné intervaly aktualizace podle povahy každé třídy – například obsah PDF souborů, na rozdíl od informací o kulturních událostech, opravdu není potřeba aktualizovat denně. Třídy, do kterých je obsah dělen, a intervaly, které k nim náleží, již byly zmíněny a popsány v podkapitole 4.1.1 o intervalech obnovy.

Tento systém klasifikace usnadňuje potřebné vyčlenění obsahu souvisejícího s událostmi a zajišťuje včasné rozšiřování a aktualizaci časově citlivých informací, přičemž nezatěžuje systém nepotřebnými aktualizacemi tolik se neměnicího obsahu, čímž zvyšuje užitečnost platformy. Klasifikace obsahu do různých kategorií navíc přináší významné výhody pro indexaci vektorů a následné filtrování při vyhledávání dat z vektorové databáze, což otevírá i možnosti pro experimenty s re-rankingem vyhledaných dat podle příslušné třídy.

Klasifikací se v projektu zabývá modul `content_classification`. První vyzkoušený přístup ke klasifikaci, implementovaný ve funkci `get_content_type_simple`, využívá k zero-shot klasifikaci transformers model `facebook/bart-large-mnli`¹². Tato metoda přímo přiřazuje vstupnímu textu jednu z předem definovaných tříd, umožňuje kategorizovat text do předem definovaných tříd bez nutnosti specifických trénovacích dat, což poskytuje univerzální přístup ke klasifikaci různorodého textového obsahu. Využívá předem natrénovaný model schopný porozumět úlohám odvozování přirozeného jazyka, takže je vhodný pro různé klasifikační úlohy.

| Třída | Správně klasifikováno | |
|----------------|-----------------------|-------|
| place | 25/25 | 100 % |
| static | 0/25 | 0 % |
| administration | 1/25 | 4 % |
| event | 20/25 | 80 % |
| celkem | 46/100 | 46 % |

Tabulka 4.4: Zero-shot výsledky.

¹²<https://huggingface.co/facebook/bart-large-mnli>

Při přizpůsobování modelu vlastním potřebám jsem zjistila několik nevýhod. Výkonnost modelu výrazně trpí při použití na domény, které se liší od jeho tréninkových dat – není vůbec adaptivní. A v tomto projektu se bohužel ukázalo, že potřeby klasifikace přesahují schopnosti tohoto modelu, jak lze vidět i v tabulce 4.4. Výsledky bohužel nebyly lepší ani při pokusech s alternativními názvy tříd.

```
def get_content_type_by_function_call(agent: ApiAgent, content: str) -> str:

    def assign_place() -> str:
        """Call this function if you encounter entity that describes
        places, tours or destinations in or near Brno city, such as
        restaurant, cafe, bar, bakery, museum, greenery, church, castle,
        university, kino, theatre or similar."""
        return PLACE

    def assign_event() -> str:
        """Call this function if you encounter entity that describes
        events, such as concert, exhibition, celebration, festival,
        sports match, theatrical performance or similar."""
        return EVENT

    def assign_administration() -> str:
        """Call this function if you encounter entity that contains
        administrative information such as Municipal office, business,
        authorities, insurance, social Care, vehicle registration, taxes,
        fees, information for expats, school system, residence,
        ID cards or similar."""
        return ADMINISTRATION

    def assign_static() -> str:
        """Call this function if you encounter entity that contains
        various articles, blog posts, or an article from wikipedia or
        information about well-known personality connected with Brno
        that is not likely to change in next 5 years. This entity does
        not contain any information about places in Brno, events
        or administrative."""
        return STATIC

    f = [assign_place, assign_event, assign_administration, assign_static]

    messages = [
        SystemMessage(f"""You're a function picker based on given web-scraped
        text. Follow these instructions:
        1. Take the given text as a one whole entity.
        2. Call one of the given functions, choose one with most fitting
        description. If the function accepts no parameters, return all
        parameters but the function name empty.
        3. Stop when you find the fitting function, you must call only one
        function."""),
        UserMessage(f"""Here is the text to process '{content}'""")]

    return agent.get_function_call(locals(), f, messages=messages)
```

Výpis 4.4: Funkce pro klasifikaci *voláním funkce* jazykovým modelem.

Jako lepší řešení se nabízí využít implementovaného agenta s voláním funkcí, použití LLM umožňuje širokou škálu přizpůsobitelných dotazů a cílených odpovědí. Je tedy možné přidat příklady, jaký obsah náleží ke které třídě a vytvořit tak mnohem přesnější few-shot klasifikátor. Tento přístup umožňuje klasifikaci podle požadavků projektu bez potřeby velké sady trénovacích dat, jediné, na co spoléhá je schopnost LLM určit správnou třídu a vrátit správný výstup.

Tento přístup je implementovaný ve funkci `get_content_type_by_function_call`, kde se nacházejí funkce pro každou třídu, jejichž popis obsahuje příklady, jenž se k pojmenované třídě vážou, jak je možné vidět ve výpise 4.4. Funkce byla vyzkoušena na vybraných modelech popsanych v podkapitole 4.4, výsledky shrnuje tabulka 4.5. Již na první pohled lze vidět, že je výstup na většině modelech nesrovnatelně přesnější než výstup zero-shot modelu, přičemž nejlépe si zde vede model mixtral a nejhůře llama2-70b.

| | mixtral | llama3-8b | llama3-70b | llama2-70b | gpt-3.5 | gpt-4 |
|----------------|---------|-----------|------------|------------|---------|---------|
| Správná třída | 89 % | 87 % | 85 % | 0 % | 84 % | 87 % |
| Špatný formát | 1 % | 3 % | 0 % | 99 % | 7 % | 0 % |
| place | 100 % | 100 % | 100 % | 0 % | 96 % | 100 % |
| static | 76 % | 68 % | 44 % | 0 % | 60 % | 52 % |
| administration | 96 % | 92 % | 100 % | 0 % | 100 % | 100 % |
| event | 84 % | 88 % | 96 % | 0 % | 80 % | 96 % |
| Čas | 3m 17s | 12m 10s | 11m 33s | 9m 36s | 1m 44s | 3m 28s |
| Cena | - | - | - | - | 0.04 \$ | 0.70 \$ |

Tabulka 4.5: Srovnání modelů na úloze jednoduché klasifikace voláním funkce agentem.

Kromě toho kód obsahuje i metodu pro klasifikaci obsahu používající funkci agenta pro návrat obsahu v JSON formátu, vracející název třídy entity podle poskytnutého JSON schéma. Tato metoda abstrahuje proces klasifikace a vyžaduje, aby klasifikátor při určování typu obsahu postupoval podle explicitních instrukcí, čímž je zajištěna konzistence klasifikace napříč různými vstupy. Spolehlivost klasifikace je srovnatelná s předchozí metodou, pro použití přímo názvu třídy je však nutné jej z tohoto formátu převést.

Později byla do projektu přidána metoda, která se před zavoláním agenta pokusí klasifikovat obsah na základě vzorů nalezených v url adrese, ne vždy se pak agent volá. To je možné díky tomu, že se ke každé třídě váže několik klíčových slov, které se často vyskytují v url stránek s tímto obsahem. Například u třídy event se velmi často v url vyskytuje *program* či *event* apod. V tabulce 4.6 lze vidět, že pokus o klasifikaci před zavoláním agenta přinesl další zlepšení výsledků, zároveň dochází k výrazné optimalizaci využití zdrojů.

4.6 Zpracování dat

Využití strukturovaného textového formátu (jako je JSON), který se řídí vlastním definovaným schématem ve vektorové databázi, nabízí několik výhod oproti pouhému použití textových kusů jako vstupních dat. Dodržováním strukturovaného formátu je možné implementovat mechanismy filtrování jednotlivých položek v databázi, což usnadňuje efektivnější a cílenější dotazy. To je důležité zejména u záznamů kulturních akcí, které je potřeba fil-

| | mixtral | llama3-8b | llama3-70b | llama2-70b | gpt-3.5 | gpt-4 |
|----------------|---------|-----------|------------|------------|---------|---------|
| Správná třída | 99 % | 99 % | 100 % | 91 % | 100 % | 100 % |
| Špatný formát | 0 % | 0 % | 0 % | 13 % | 0 % | 0 % |
| place | 100 % | 96 % | 100 % | 75 % | 100 % | 100 % |
| static | 100 % | 100 % | 100 % | 100 % | 100 % | 100 % |
| administration | 100 % | 100 % | 100 % | 100 % | 100 % | 100 % |
| event | 96 % | 100 % | 100 % | 96 % | 100 % | 100 % |
| Čas | 1m 01s | 4m 51s | 4m 38s | 3m 53s | 1m 01s | 2m 12s |
| Cena | - | - | - | - | 0.02 \$ | 0.47 \$ |

Tabulka 4.6: Srovnání modelů na úloze jednoduché klasifikace voláním funkce agentem s použitím před-klasifikace na základě url adresy.

trovat dle data konání. Příklad zpracovaného textu ukazuje výpis 4.5, zde se jedná právě o kulturní akci s detekovaným datem.

Tento přístup navíc zvyšuje konzistenci integrity dat tím, že zajišťuje, aby každá položka dodržovala zadané schéma, čímž se snižuje riziko inkonzistence a chyb v databázi. Konzistentní formát navíc umožňuje snadnější škálovatelnost a údržbu architektury databáze, protože změny a aktualizace lze provádět systematicky ve všech položkách.

```
{
  "header": "Vox Imaginis: Miluji...",
  "record_type": "event",
  "brief": "A string quartet concert dedicated to the 170th anniversary of Leos Janacek's birth and the 100th anniversary of Jiri Wolker's death, featuring Graffovo kvarteto and an exhibition of kovova zardiniera.",
  "text": "The Vox Imaginis: Miluji... event is dedicated to the 170th anniversary of composer Leos Janacek's birth and the 100th anniversary of poet Jiri Wolker's death. The concert will feature Graffovo kvarteto performing Smyccovy kvartet n.1 \"Miluji\" by Vladimir Ambrose and Smyccovy kvartet n.2 \"Listy duverne\" by Leos Janacek. The event also includes a selection of Jiri Wolker's love poetry and an exhibition of kovova zardiniera (1920-1930) by architect Emil Kralik from the collections of Moravska galerie in Brno.",
  "url": "https://moravska-galerie.cz/program/vox-imaginis-miluji/",
  "date_fetched": "2024-05-05",
  "address": "Umeleckoprumslove muzeum, Brno, Czech Republic",
  "dates": "[{\"start\": \"2024-05-14T19:00\"}]"}
}
```

Výpis 4.5: Příklad parsovaného JSON textu

Jak již bylo řečeno, využití vlastního schématu umožňuje integraci dalších dat, která mohou vylepšit proces vektorizace. Například začlenění souhrnu spolu s plným textem do indexu vektorizace může zvýšit kontextovost a relevanci generovaných vektorů a nabídnout bohatší a diferencovanější soubor dat, tento krok byl zahrnut po inspiraci již zmíněným článkem [33]. Souhrn obsahu nejen, že zvýrazňuje význam příslušného textu, ale potenciálně přidává i další slova, jichž se může chytit vyhledávání v databázi podle klíčových slov. Ve stejném duchu se nese přidání nadpisu.

Dělení obsahu do tříd je prospěšné nejen pro umožnění filtrování, jak zmiňuje podkapitola o klasifikaci 4.5, ale i třída se stává jedním z aspektů, na jejichž základě je obsah vektorizován. Díky tomu bude mezi vektory jednotlivých tříd menší vzdálenost, což je též optimalizací následného vyhledávání.

4.6.1 Parsování dat

Syntaktická analýza, též známá pod pojmem *parsing*, je proces, při kterém se analyzovaný text přetváří do strukturované podoby. V této práci si dovoluji pro tento proces používat počestěný pojem *parsování*.

Kousky textů je tedy potřeba zpracovat do strukturované JSON formy, to implementuje modul `content_parsing`. Ten, podobně jako modul pro klasifikaci 4.4, využívá volání funkcí s výstupy velkého jazykového modelu prostřednictvím třídy `ApiAgent`. Použití LLM pro analýzu textu zjednodušuje extrakci strukturovaných informací z nestrukturovaných textových dat, všechny parametry je možné získat naráz v jednom požadavku.

Rozlišuji zde dvě schémata, která korelují se schématem vektorové databáze, jak je popsáno v kapitole 4.7.2. Dodržení formátu je zaručeno tím, že volané funkce vrací objekt Python Dataclass s parametry, které jsou získány z LLM. Definice schémat je následující:

```
@dataclass
class BaseSchema:
    """ Base schema for all entities """
    header: str
    record_type: str
    brief: str
    text: str
    url: str
    date_fetched: str
    address: str = None

@dataclass
class EventSchema(BaseSchema):
    """ Event schema for events """
    dates: list = None
```

Položka `header` označuje nadpis úryvku, jedná se o trefné označení podtrhující hodnotu úryvku ve vektorové databázi, LLM tento nadpis vytvoří, není-li součástí předaného textu. Parametr `record_type` označuje třídu, do které byl úryvek klasifikován, zatímco `brief` je krátký souhrn daného textu založený na klíčových slovech.

Mimo přiřazeného `textu` a `url` se zde nachází i datum přidání `date_fetched` a adresa extrahovaná z textu, nacházela-li se v něm.

Třída pro kulturní akce tyto položky rozšiřuje o parametr `dates`, který je listem obsahujícím data dle formátu `dates=' [{"start": start_date, "end": end_date}] '`, přičemž parametr `"end"` je nepovinný a slouží pro data, která jsou časovým úsekem.

V rámci primární funkce `get_parsed_content_by_function_call` jsou podobně jako u kódu ve výpisu 4.4 definovány vnořené funkce `add_place`, `add_event`, `add_administration` a `add_static`, s tím rozdílem, že tyto funkce přijímají parametry pro schéma a konstruují schéma podle třídy entity, ke které náleží. V pokynech pro model se zde zároveň nachází instrukce pro doplnění parametrů:

```
SystemMessage(f"""\n\nYou are a smart processor of web-scraped text.\n\nFollow these instructions:\n\n1. Go through the text and extract information from the article,\n   translate to English if not in English. Use plain text.\n\n2. Use the function with the most fitting description, pass parameters as\n   described in the following steps:\n   Use provided or generate a header more fitting the found text.\n   The descriptive text of the entity (for example a plot of a theatrical\n   performance for an event, insurance application process for\n   administration, or a menu of a restaurant for a place) must be\n   assigned it as the text parameter. Do NOT SHORTEN it, do NOT OMIT\n   any important information.\n   Create a brief which is a keyword-extending sum up of the text no\n   longer than 2 sentences.\n   For non-static entities only: if you encounter address of a place\n   (such as address of a municipal office for administration or address\n   of concert-hall for an event), assign is as address. Fill in "Brno,\n   Czech Republic" if the specific address not found but required.\n   For an event entity: assign date(s) of the event as list of durations.\n   The duration format is a stringified JSON object {FORMAT_EXAMPLE}\n   with fields "start" and "end". Field "end" is optional, is used for\n   period of time (that are two dates from-to, like startdate-enddate,\n   for example 31 jan-14 feb 2024). Use the format YYYY-MM-DD for date,\n   and format HH:MM:SS for time. For example {DATES_EXAMPLE}.\n\n3. End with the function call response in valid JSON format, do NOT add\n   any additional text.""")
```

Výpis 4.6: Pokyny pro model při parsování textů.

Nutno podotknout, že bylo vyzkoušeno mnoho formátů tohoto promptu a je zajímavé, jaký vliv mají na výstup modelu. Například, když nebyly kroky v zadání očíslovány, výsledky byly přibližně o 28 % horší než v případě, který popisuje tabulka 4.8 níže.

Zde jsem se rozhodla hodnotit jednak určení správné třídy, jednak správný formát předání parametrů. O správném určení třídy v rámci parsování pojednává tabulka 4.7. Zde je nutné podotknout, že se pro porovnávací účely jedná o výsledky bez umožnění více pokusů při nesprávném výsledku. Výsledky ukazují, že si nejlépe vedl model gpt-4, přičemž je poněkud překvapivé, že model gpt-3.5 dosahuje značně horších výsledků, než instruct open-source modely.

Výsledky samotného obsahu parsování ukazuje tabulka 4.8. Zde lze vidět, že nejproblémovější třídou je pochopitelně *event*, a to právě kvůli komplexitě parametru *dates*. I přesto bych však výsledky označila jako dobré. Je těžké určit, který model si vedl nejlépe, jelikož je potřeba brát v úvahu nejen přesnost výsledků, ale i čas a cenu. Nejhůře si nepřekvapivě vedl model llama2-70b.

| | mixtral | llama3-8b | llama3-70b | llama2-70b | gpt-3.5 | gpt-4 |
|----------------|----------------|------------------|-------------------|-------------------|----------------|--------------|
| Správná třída | 82 % | 76 % | 80 % | 4 % | 42 % | 90 % |
| Špatný formát | 8 % | 12 % | 0 % | 96 % | 53 % | 1 % |
| place | 96 % | 92 % | 100 % | 0 % | 76 % | 100 % |
| static | 56 % | 88 % | 28 % | 4 % | 40 % | 64 % |
| administration | 88 % | 92 % | 100 % | 8 % | 48 % | 100 % |
| event | 88 % | 32 % | 92 % | 4 % | 4 % | 96 % |

Tabulka 4.7: Srovnání modelů na úloze klasifikace v rámci parsování voláním funkce agentem.

| | mixtral | llama3-8b | llama3-70b | llama2-70b | gpt-3.5 | gpt-4 |
|----------------|----------------|------------------|-------------------|-------------------|----------------|--------------|
| Správný formát | 92 % | 88 % | 100 % | 4 % | 47 % | 99 % |
| place | 96 % | 96 % | 100 % | 0 % | 76 % | 100 % |
| static | 96 % | 96 % | 100 % | 4 % | 60 % | 96 % |
| administration | 88 % | 96 % | 100 % | 8 % | 48 % | 100 % |
| event | 88 % | 64 % | 100 % | 4 % | 4 % | 100 % |
| Čas | 29m 05s | 105m 18s | 116m 32s | 45m 49s | 6m 02s | 11m 18s |
| Cena | - | - | - | - | 0.09 \$ | 1.99 \$ |

Tabulka 4.8: Srovnání modelů na úloze parsování voláním funkce agentem.

| | mixtral | llama3-8b | llama3-70b | llama2-70b | gpt-3.5 | gpt-4 |
|----------------|----------------|------------------|-------------------|-------------------|----------------|--------------|
| Správná třída | 99 % | 99 % | 100 % | 74 % | 100 % | 100 % |
| Špatný formát | 0 % | 0 % | 0 % | 26 % | 0 % | 0 % |
| place | 100 % | 96 % | 100 % | 0 % | 100 % | 100 % |
| static | 100 % | 100 % | 100 % | 100 % | 100 % | 100 % |
| administration | 100 % | 100 % | 100 % | 100 % | 100 % | 100 % |
| event | 96 % | 100 % | 100 % | 96 % | 100 % | 100 % |

Tabulka 4.9: Srovnání modelů na úloze klasifikace v rámci parsování děleným voláním funkce agentem s použitím před-klasifikace na základě url adresy.

| | mixtral | llama3-8b | llama3-70b | llama2-70b | gpt-3.5 | gpt-4 |
|----------------|----------------|------------------|-------------------|-------------------|----------------|--------------|
| Správný formát | 98 % | 85 % | 100 % | 0 % | 100 % | 100 % |
| place | 100 % | 100 % | 100 % | 0 % | 100 % | 100 % |
| static | 96 % | 100 % | 100 % | 0 % | 100 % | 100 % |
| administration | 96 % | 100 % | 100 % | 0 % | 100 % | 100 % |
| event | 100 % | 40 % | 100 % | 0 % | 100 % | 100 % |
| Čas | 23m 52s | 59m 18s | 102m 43s | 145m 54s | 5m 58s | 12m 8s |
| Cena | - | - | - | - | 0.12 \$ | 1.93 \$ |

Tabulka 4.10: Srovnání modelů na úloze parsování děleným voláním funkce agentem s použitím před-klasifikace na základě url adresy.

Protože jsem stále nebyla s výsledky dostatečně spokojená, přišla jsem s již zmíněným přístupem klasifikace na základě url adresy před samotným voláním agenta. Zároveň jsem se rozhodla zkusit volání funkcí rozdělit na:

1. výběr funkce, která se bude volat,
2. dodání parametrů pro vybranou funkci.

A to s vidinou zlepšení výsledků v oblasti navracení správného formátu, jelikož model nebude mít k dispozici všechny možné formáty, ale dojde k vynucení pouze toho vybraného, pro což je v třídě `ApiAgent` přímo implementovaná funkce.

Výsledky klasifikace v rámci parsování tímto způsobem ukazuje tabulka 4.9 a lze si povšimnout, že podobně jako u klasifikace v 4.5, dochází k drastickému zlepšení výsledků. Parsování srovnává tabulka 4.10 – kromě modelu llama2-70b, který prostě není schopen na první pokus vrátit správný formát, mají všechny modely velmi pěkné výsledky. Jediný model llama3-8b vykazuje mírné zhoršení, a to proto, že si nevede dobře se třídou `event` kvůli parametru `dates`, v jiných třídách však podává 100% výsledky, stejně jako všechny ostatní kromě modelu mixtral.

Nakonec jsem se rozhodla vyzkoušet i přímé parsování (neděleným) voláním funkce agentem s použitím před-klasifikace na základě url adresy, a to hlavně z důvodu úspory času. Ukázalo se, že výsledky modelů placených API jsou srovnatelné s výsledky parsování děleným voláním funkce agentem s použitím před-klasifikace na základě url, přičemž došlo k 16% úspoře času. Pro úsporu času a prostředků je mimo lokální modely pro parsování dat nakonec používána tato metoda.

Co se týče výběru modelu, který se pro sběr dat používá, rozhodla jsem se zvažovat jen modely se stoprocentními výsledky, a to protože je při zpracování klíčová právě spolehlivost. Pak už jen záleží, zda je k dispozici více času, nebo financí. Já jsem se rozhodla pro llama3-70b pro prvotní sběr dat, na který není žádný časový nátlak, a pak model gpt-3.5 pro průběžné aktualizace.

4.7 Vektorová reprezentace dat

Ze sesbíraných dat je potřeba získat jejich vektorové reprezentace a uložit je do vektorové databáze, při jejímž výběru jsem zvažovala tyto populární vektorové databáze:

- ElasticSearch¹³ – primárně určeno pro vyhledávání a analýzu logů, skvělá schopnost zpracovávat polostrukturovaná nebo nestrukturovaná data,
- Weaviate¹⁴ – efektivní ukládání vektorů, vyhledávání a aktualizace v reálném čase,
- Pinecone¹⁵ – vysoce výkonné vyhledávání podobností vektorů,
- Chroma¹⁶ – podpora rozsáhlých vektorových dat s vyhledáváním v reálném čase.

¹³<https://www.elastic.co/elasticsearch>

¹⁴<https://weaviate.io/>

¹⁵<https://www.pinecone.io/>

¹⁶<https://docs.trychroma.com/>

| | Elasticsearch | Weaviate | Pinecone | Chroma |
|---------------------------------|---------------|----------|----------|--------|
| Open-Source | Ne | Ano | Ne | Ano |
| Self-host | Ano | Ano | Ne | Ano |
| Vytvořeno speciálně pro vektory | Ne | Ano | Ano | Ano |
| Hybridní vyhledávání | Ano | Ano | Ano | Ano |
| Dynamické umístění segmentů | Ne | Ne | Ano | Ano |
| Statické rozdělování dat | Ano | Ano | Ne | Ne |
| Bezplatné hostování v cloudu | Ne | Ano | Ano | Ne |

Tabulka 4.11: Srovnání vektorových databází dle ANN Benchmarks [7].

Jak i srovnání v tabulce 4.11 napovídá, vektorové databáze mají mezi sebou drobné rozdíly, ale jednoznačně určit tu nejlepší nelze. Z rozpočtových důvodů jsem se rozhodla vybírat z bezplatných služeb. Po pečlivém vyhodnocení jsem jako preferovanou vektorovou databázi pro tuto práci vybrala Weaviate díky jejímu open-source charakteru, robustní sadě funkcí a aktivní podpoře komunity – velká komunita, dostupné moduly a rozsáhlá dokumentace tohoto systému výběr ovlivnila nejvíce.

Weaviate je vektorová databáze, která podporuje rozsáhlé vyhledávání a filtrování. Modulární struktura umožňuje bezproblémovou integraci volitelných modulů pro vektorizaci, zodpovídání otázek, kontrolu pravopisu a generování textu, což zvyšuje jeho univerzálnost a přizpůsobivost v široké škále typů dat a případů použití. Navíc Weaviate nabízí jednoduchý Docker konfigurační¹⁷ pro lokální nasazení, který umožňuje vytvořit a nakonfigurovat potřebné nastavení Weaviate, včetně specifických Weaviate modulů, které běží lokálně.

4.7.1 Vektorizace a indexování

K převodu obsahu na vektory je používán právě vektorizační/vyhledávací modul (Vectorizer&Retriever Text Module) databáze, díky němuž není potřeba předávat databázi již vytvořené embeddingy, modul je totiž z textu vytvoří.

Byly zvažovány moduly založené na API, jako `text2vec-cohere` (jehož bezplatnou úroveň bohužel požadavky systému přesahují) nebo `text2vec-openai`. Weaviate nabízí i širokou škálu open-source modulů, např. `text2vec-contextionary` nebo `text2vec-gpt4all`. Nejvýhodnější se pro tuto práci ukázal být modul open-source `text2vec-transformers`.

Mezi transformers modely jsou samozřejmě na výběr i vícejazyčné modely, ale jak jsem už zmínila v kapitole 4.1, jejich výkonnost je pro český jazyk oproti čistě anglickým modelům značně nižší, a je tedy výhodné vektorizovat všechny texty už v jejich anglické podobě. Nakonec jsem zvolila `sentence-transformers/multi-qa-MiniLM-L6-cos-v1`¹⁸, který se podle *Reimense* [34] ukazuje být nejlepším kompromisem mezi přesností a rychlostí. Je speciálně trénovaný pro sémantické vyhledávání a jeho nižší dimenzionalita snižuje paměťové nároky větších datasetů.

Při vektorizaci textů a následném vyhledávání na základě podobnosti s vektorizovanými dotazy je dle *Jurafskyho* a *Martina* [17] pro zachování sémantické konzistence a zvýšení přesnosti vyhledávání zásadní používat v celém procesu stejný embedding model. To je použitím

¹⁷<https://weaviate.io/developers/weaviate/installation/docker-compose>

¹⁸<https://huggingface.co/sentence-transformers/multi-qa-MiniLM-L6-cos-v1>

Weaviate modulů zajištěno, jelikož automaticky vektorizuje dotazy na schéma tím modelem, který byl použit k indexaci dat. Konzistentní embeddingy zajišťují přesnou sémantickou shodu a minimalizují sémantický drift, čímž zlepšují relevanci a výsledky vyhledávání.

Po vektorizaci textů přichází na řadu krok indexování vektorů (proces uspořádání vektorů a přístupu k nim). Indexování je klíčovou součástí vektorových databází, umožňuje totiž efektivní vyhledávání a porovnávání vektorů na základě stanovených kritérií.

Weaviate podporuje dva typy indexování vektorů – prvním je plochý (flat) index, což je jednoduchý a úsporný index určený pro malé datové sady, druhým je HNSW index, který je škálovatelný a rychlý v době dotazování, ale nákladný při přidávání dat během vytváření indexu. Navíc je zde k dispozici funkce dynamického indexu umožňující automatické přepnutí z plochého indexu na index HNSW při zvýšení počtu objektů. Ta je v současné době (duben 2024) považována za experimentální novinku, i přesto jsem se ji rozhodla použít.

Jako metrika vzdálenosti (metoda používaná k měření podobnosti nebo nepodobnosti mezi dvěma vektory) byla zvolena kosinová vzdálenost, která měří úhlovou vzdálenost mezi dvěma vektory a je ideální pro textová data. Weaviate v době importu (nebo čtení v případě dotazu) všechny vektory normalizuje na délku 1, pro výpočet vzdálenosti používá pro zvýšení efektivity bodový součin. Rozsah hodnot je od 0 do 2, kde 0 označuje shodné vektory.

4.7.2 Schéma

Weaviate schéma¹⁹ (od přechodu na v4 nazváno collection) definuje ve formálním jazyce datovou strukturu jak mají být data organizována a uložena. Schéma definuje datové třídy (tj. kolekce objektů), vlastnosti a parametry v rámci třídy (název, typ, popis, nastavení), možné grafické vazby mezi datovými objekty (křížové odkazy) a také nastavení, jako je vektorizační modul a konfigurace indexů.

```
self.client.collections.create(
    name=EVENT_SCHEMA_NAME,
    description="Culture events, concerts, festivals etc.",
    vectorizer_config=Vectorizer.text2vec_transformers(),
    reranker_config=Reranker.cohere(),
    vector_index_config=VectorIndex.dynamic(VectorDistances.COSINE),
    properties=[
        Property(name="header", data_type=TEXT),
        Property(name="record_type", data_type=TEXT),
        Property(name="brief", data_type=TEXT),
        Property(name="text", data_type=TEXT),
        Property(name="url", data_type=TEXT),
        Property(name="date_fetched", data_type=DATE, skip_indexing=True),
        Property(name="address", data_type=TEXT),
        Property(name="dates", data_type=OBJECT_ARRAY, nested_properties=[
            Property(name="date", data_type=OBJECT, nested_properties=[
                Property(name="start", data_type=DATE),
                Property(name="end", data_type=DATE, optional=True)
            ])],
        Property(name="dates_str", data_type=TEXT, skip_indexing=True)])
```

Výpis 4.7: Definice schéma pro kulturní události.

¹⁹<https://weaviate.io/developers/weaviate/starter-guides/schema>

Pro práci s vektorovou databází v implementaci slouží modul `vector_storage` a zde je schéma také definováno. Databáze následuje formát parsovaných dat popsany v kapitole 4.6. Jsou zde používána dvě schémata, jedno pro kulturní akce a druhé pro všechny ostatní typy dat, které má definici ochuzenou o položky spojené s datem. Výpis 4.7 ukazuje jak nastavení vektorizace, tak parametrů. Zde si lze povšimnout odchylky od schéma z výpisu 4.6.1 v nadbytečné položce `dates_str`. Ta je přidána z toho důvodu, že filtrování nad typem `object` ještě (duben 2024) není ve Weaviate podporováno²⁰, takže zatím nelze stavět pokročilé filtry nad položkou `dates`, a je potřeba toto úskalí řešit jinak.

4.7.3 Vyhledávání informací

Pro získání dat z databáze na základě dotazu je k dispozici několik způsobů. Weaviate klient nabízí tyto základní možnosti vyhledávání:

- vyhledávání podle podobnosti vektorů,
- vyhledávání na základě klíčových slov,
- smíšené vyhledávání (hybrid search).

Pro vyhledávání na základě podobnosti vektorů lze ve Weaviate použít funkci `nearText` a provést textové vyhledávání zadaného dotazu, např. *How many citizens does live in Brno?*. Tato funkce zahrnuje porovnání podobnosti textu mezi dotazem a uloženými textovými daty v databázi. Alternativně lze využít možnosti vyhledávání pomocí funkce `nearVector` k získání dat na základě vektorové podobnosti. V práci se v metodách `vector_query_base` a `vector_query_event` používá vyhledávání `nearText`.

Protože je algoritmus k-nejbližších sousedů (kNN) velmi výpočetně nákladný, používá Weaviate pro vyhledávání podle podobnosti vektorů algoritmus Aproximovaně nejbližších sousedů (ANN) který na úkor trochy přesnosti nabízí obrovský nárůst rychlosti [10].

Vyhledávání podle klíčových slov (*BM25 – Best match 25*) vrací objekty, které mají nejvyšší skóre BM25F. BM25 vychází z již zmíněné metody hodnocení klíčových slov TF-IDF – z výpočtu přebírá model binární nezávislosti a přidává normalizační penalizaci, která zohledňuje délku dokumentu vzhledem k průměrné délce všech dokumentů v databázi.

$$\text{score}(D, Q) = \sum_{i=1}^n \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)} \quad (4.1)$$

Výsledné skóre popisuje rovnice 4.1: skóre dvojice dokument D , dotaz Q je určeno posouzením jedinečnosti každého klíčového slova v dotazu q vzhledem ke kolekci textů. BM25 obsahuje další statické parametry, k_1 a b , které pomáhají kalibrovat výkonnost pro konkrétní soubory dat [3]. V práci je vyhledávání pomocí této metody implementováno v metodách `keyword_query_base` a `keyword_query_event`.

S oběma těmito metodami jsem experimentovala na různých dotazech, přičemž výsledky pokusů jsou popsány v kapitole o vyhodnocení 4.7.4. U dotazů závislejících na přesných

²⁰<https://github.com/weaviate/weaviate/issues/3694>

pojmech má lepší výsledky vyhledávání podle klíčových slov, jindy si vedlo lépe vyhledávání na základě podobnosti vektorů, a nelze jednoznačně určit, který z přístupů je obecně lepší.

V práci je proto použito smíšené vyhledávání – metoda kombinující tyto dvě metody, zmíněná v kapitole o optimalizaci vyhledání dat 3.2.2. Hybridní vyhledávání je ideální pro systém, který chce využít sílu sémantického vyhledávání, ale stále se spoléhá na přesnou shodu klíčových slov, jako je ten náš. Například v případě otázky *What influence did Gregor Mendel have on the scientific community in Brno?* se vyhledávání na základě podobnosti vektorů zaměří na celkový význam věty, přičemž řídké vektorové vyhledávání na základě klíčových slov bude odpovídat pouze frázi *Gregor Mendel*.

Smíšené vyhledávání ve Weaviate kombinuje výsledky BM25 a vektorového vyhledávání do jediného hodnoceného seznamu pomocí RRF (Reciprocal Rank Fusion) skóre. To se vypočítá jako součet vzájemných ohodnocení každého seznamu: $\sum_{d \in D} \frac{1}{k+r(d)}$. Tím, že se do jmenovatele dává pořadí dokumentu, jsou penalizovány dokumenty, které mají v seznamu nižší pořadí.

Metody `hybrid_query_base` a `hybrid_query_event` slouží v implementaci pro smíšené vyhledávání. Výpis 4.8 ukazuje metodu pro smíšené vyhledávání kulturních akcí. Na vyhledávací proces je pro optimalizaci aplikován fúzovací algoritmus Relative Score Fusion. Jedná se o úpravu RRF používající normalizaci. Celkové skóre je tak škálovaný součet normalizované vektorové podobnosti a normalizovaného skóre BM25. To umožňuje zachovat relevanci danou původními algoritmy vyhledávání klíčových slov/vektorů a zároveň je možné výsledky porovnávat a slučovat.

```
def hybrid_query_event(self, query: str, dates: list):
    """
    Query the 'EventSchema' collection in Weaviate
    based on a question using hybrid search.
    :return: The query results
    """
    collection = self.client.collections.get(EVENT_SCHEMA_NAME)
    query = collection.query.hybrid(
        query=query,
        fusion_type=wcq.HybridFusion.RELATIVE_SCORE,
        auto_limit=2,
        limit=5,
        alpha=0.65,
        query_properties=["text", "brief", "header", "address"],
        return_properties=["text", "brief", "header", "address", "url"],
        filters=wcq.Filter.by_property("dates_str").contains_any(dates)
                                                if dates else None,

        rerank=wcq.Rerank(
            prop="text",
            query=query
        ),
        return_metadata=wcq.MetadataQuery(score=True, explain_score=True)
    )
    return query
```

Výpis 4.8: Smíšené vyhledávání ve schématu pro kulturní akce.

Hodnota alpha nastavuje relativní váhy těchto dvou vyhledávacích kritérií, přičemž hodnota alfa 1 znamená čistě vektorové vyhledávání a alpha 0 se rozumí vyhledávání pouze podle klíčových slov. Na základě experimentů jsem zde zvolila hodnotu 0.65.

Zároveň jsem se rozhodla použít tzv. *autocut* funkci, která omezuje výsledky tím, že detekuje nespojitosti, například náhlé změny v metrikách, jako je vektorová vzdálenost nebo skóre hledání v souboru výsledků. Spoléhá na to, že v datech existují přirozené shluky objektů s blízkým skóre a nezahrnuje data, jejichž skóre se významně liší od předchozích. Celkový maximální počet výsledků je jinak limitován na 5.

Po vyhledání je jako další optimalizace použit re-ranking popsany v kapitole o optimalizaci vyhledaných výsledků 3.2.3. Pořadí výsledků vrácených vyhledávačem se změní jiným modelem, konkrétně *cohere-reranking*. Protože re-ranker pracuje s menší podmnožinou dat, lze tento omezeně dostupný model využít k vylepšení relevance vyhledávání, jelikož je jeho bezplatná úroveň, na rozdíl od úlohy vektorizace, pro tuto úlohu zatím dostačující.

Rozdílem mezi vyhledáváním v *EventSchema* a *BaseSchema* je potřeba filtrování podle data. K tomu má sloužit parametr *dates*, ale protože se jedná o objekt a filtrování objektů zatím není ve Weaviate podporováno, je filtrování prováděno podle položky *dates_str*. Zde je uložen textový řetězec obsahující všechna data, která se kulturní akce týkají, a při filtrování je porovnáván se seznamem dat získaných z uživatelova dotazu.

Výsledkem vyhledávání je *QueryReturn* objekt obsahující list získaných úryvků. Mimo položek, které jsou stanoveny v *return_properties* se zde nachází i skóre a další metadata.

4.7.4 Evaluace vyhledávání informací

Pro evaluaci vyhledávání informací jsem, podobně jako u vyhodnocování v kapitole 4.6.1 o zpracování dat, vytvořila datovou sadu 100 parsovaných úryvků, které se (spolu s ostatními daty) nacházejí ve vektorové databázi. K těmto úryvkům jsem pomocí agenta synteticky vygenerovala dotazy, se kterými jsem následně dotazovala databázi každým typem dotazu, zatímco počet vrácených položek dotazu byl limitován na 3.

| | vector | keyword | hybrid |
|--------------------|---------------|----------------|---------------|
| Celkem | 82 % | 87 % | 95 % |
| <i>BaseSchema</i> | 78 % | 93 % | 97 % |
| <i>EventSchema</i> | 92 % | 64 % | 84 % |

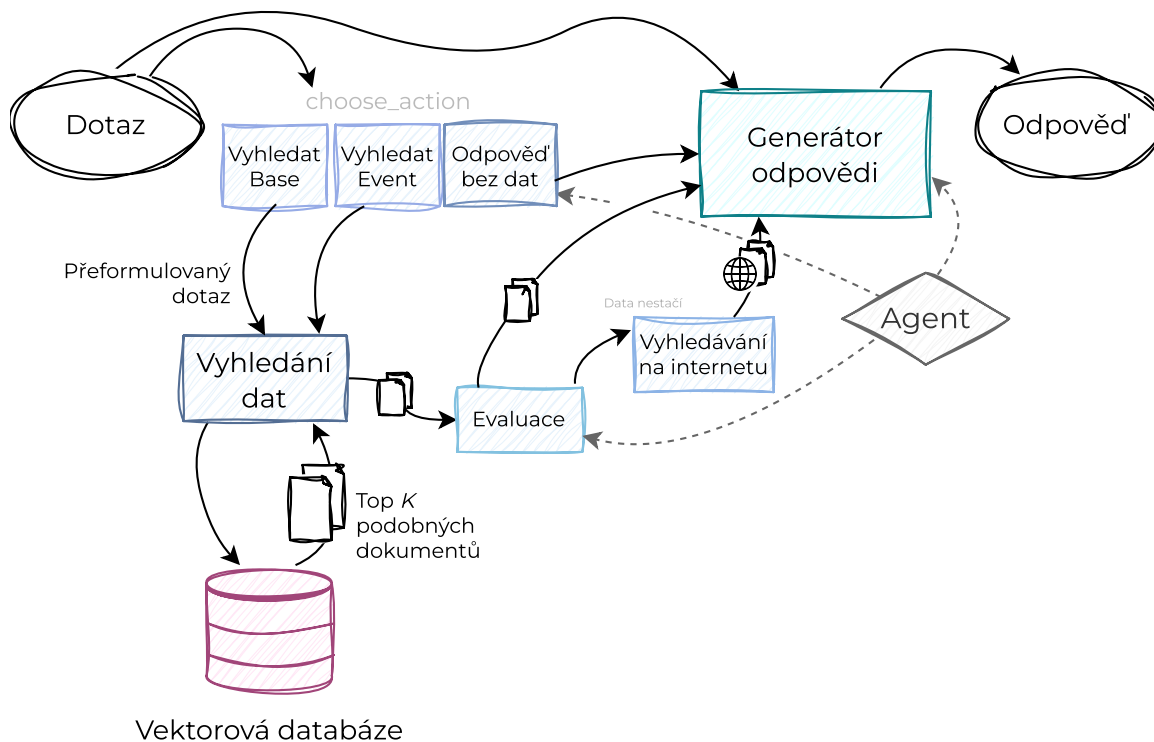
Tabulka 4.12: Srovnání vyhledávacích metod.

Srovnání, kolik výsledků obsahuje text, ke kterému byla vygenerovaná otázka ukazuje tabulka 4.12. Lze pozorovat, že vyhledávání podle klíčových slov je sice celkově lepší, než vyhledávání na základě podobnosti vektorů, ale to si zase vede mnohem lépe v rámci schématu pro kulturní akce. Celkového nejlepšího výsledku dosáhlo hybridní vyhledávání.

4.8 Konstrukce odpovědí

V podstatě se dá říci, že se za vytvářením odpovědí skrývá celá RAG pipeline, kdy se od uživatelského dotazu dostáváme přes získání dat až k vygenerované odpovědi. Je zde však

více mezikroků, než by se mohlo zdát, celý proces je naznačen na schéma na obrázku 4.7. Tato část je v implementaci zahrnuta v modulu `answer_creation`, odkud jsou odpovědi posílány uživatelskému rozhraní. Skládá se z funkcí integrujících inteligentní volání funkcí na základě porozumění kontextu, vyhledávání dat z internetu, transformaci dat a pomocných funkcí pro instrukce extrahování časových údajů z dotazů.



Obrázek 4.7: Schéma generování odpovědi.

Po obdržení je uživatelská zpráva předána k analýze funkcí `choose_action`, která rozhoduje o dalším postupu. Využívá agenta k zavolání potřebné funkce podle charakteru uživatelského dotazu, LLM má na výběr z těchto tří možností:

1. Uživatelská zpráva není dotazem nebo nemůže být relevantní k městu Brnu.
2. Uživatelská zpráva se týká míst, administrativy či obecně Brna.
3. Uživatel se ptá ohledně kulturních akcí.

Není-li relevantní k Brnu, což jsou zprávy jako např. *Hello I'm Kate* či *Tell me a joke*, zavolá agent funkci pro vygenerování odpovědi bez jakéhokoli vyhledávání dat.

Je-li však dotaz relevantní k Brnu, nebo se uživatel zajímá o kulturní akce, dojde k zavolání funkce pro vyhledávání dat v příslušném schématu vektorové databáze. Model je instruován, aby uživatelský dotaz v případě potřeby přeformuloval s důrazem na klíčová slova, pomocí tohoto dotazu jsou následně vyhledávány relevantní úryvky ve vektorové databázi.

V případě uvedení času zájmu v dotazu ke kulturním akcím je model instruován k extrakci těchto časových informací, na jejichž základě jsou pak nalezené informace o událostech

filtrvány, jak bylo popsáno ve výpise 4.8. Extrakce časových informací je prováděna na základě předání aktuálního data LLM. Model následně dle přidávaných příkladů, aktuálního data a uživatelského dotazu vytvoří list dat, kterých se dotaz týká.

Následuje evaluace vyhledaných dat implementovaná ve funkci `eval_answer_data`. Zde je agent instruován k posouzení, zda jsou vyhledaná data relevantní k původnímu netransformovanému dotazu. Tento krok zhodnocení je známý také pod pojmem *chain of thought reasoning*, což popisuje i *Ilín* v [14]. Tato technika zahrnuje strukturování dotazů tak, aby vedly jazykové modely strategickým myšlenkovým řetězcem a zajistily přesnější, pronikavější a logičtější uvažování při generování odpovědí.

Rozhodne-li, že předaná data pro vytvoření odpovědi nestačí, dojde k zavolání funkce pro vyhledávání na internetu. Pro online vyhledávání jsem se rozhodla použít knihovnu `DuckDuckGoSearch`²¹ s nastavením české lokace, přičemž je k dotazu přidáno „(Brno, CZ)“ aby vyhledávač věděl, na co se má zaměřit, a nevracel irelevantní výsledky.

At už bylo rozhodnuto pro internetové vyhledávání, či ne, v obou případech jsou dokumenty spolu s historií konverzace předány ke tvorbě odpovědi. Pro generování odpovědi je opět použit agent, tentokrát s instrukcemi pro zohlednění historie konverzace a odpověď v jazyce uživatelského dotazu. Odpovědi voláním funkce je agent přinucen předat nejen text odpovědi, ale i list zdrojů, které použil. Tento způsob získání odpovědi slouží k zamezení halucinování jazykového modelu. Zároveň je tím uživateli umožněno ověření faktické správnosti odpovědi.

Nepodařilo-li se však získat žádné dokumenty, model se pokusí odpovědět bez jakýchkoliv informací, přičemž má nařízeno přiznat, že neví odpověď.

4.9 Uživatelské rozhraní

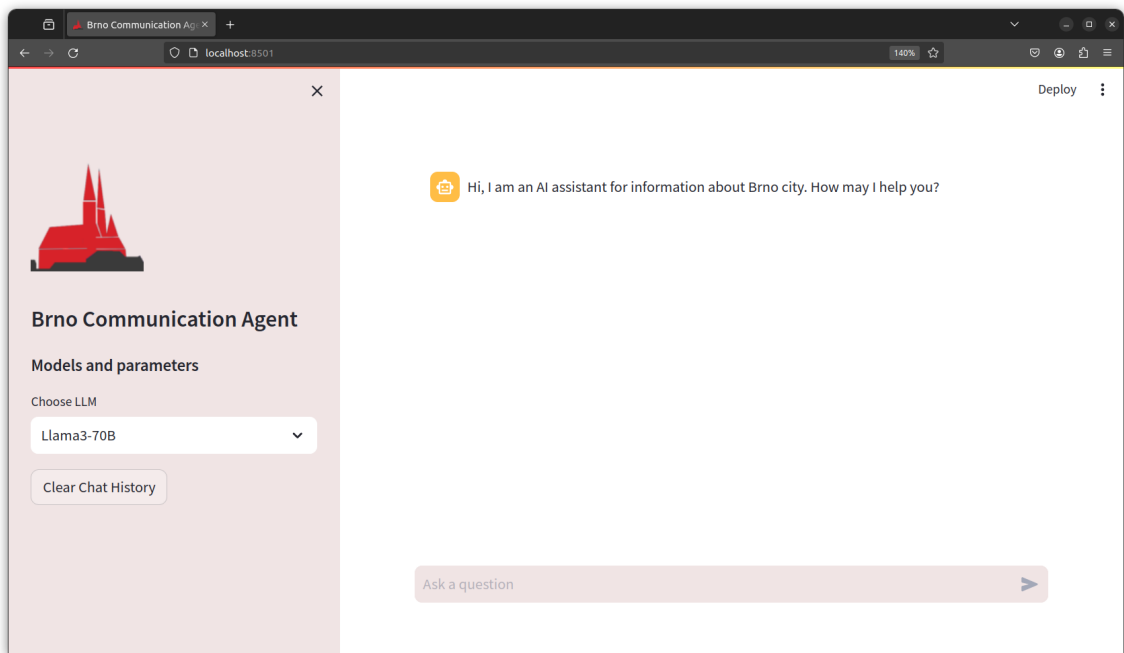
Grafické uživatelské rozhraní představuje rozhraní chatu, do kterého mohou uživatelé zadávat své dotazy a asistent jim odpovídá příslušnými informacemi. K implementaci jsem se rozhodla použít `Streamlit`²², knihovnu určenou pro vytváření interaktivních webových aplikací, která zjednodušuje proces vytváření uživatelsky přívětivých grafických uživatelských rozhraní pro projekty datové vědy a strojového učení. Základem implementace jsou funkce pro nastavení konfigurace stránky, zobrazení obrázků, udržování historie konverzace a získání generované odpovědi na základě uživatelského vstupu pomocí vybraného modelu.

Za účelem testování systému prvotními uživateli jsem vytvořila speciální testovací uživatelské rozhraní vyobrazené na obrázku 4.8, které nabízí možnost volby jazykového modelu, který bude uživatelský vstup zpracovávat a generovat odpovědi.

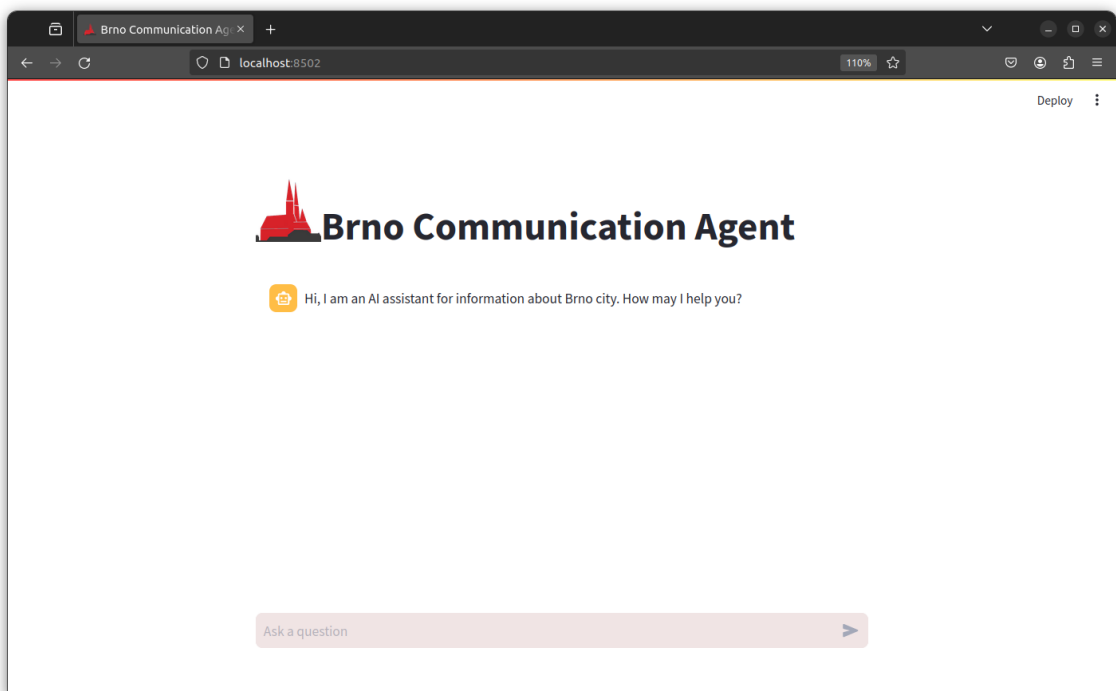
Produkční uživatelské rozhraní lze vidět na obrázku 4.9, zde není možnost volby modelů, ale je fixně nastaven model `gpt-3.5`, jelikož, jak ukazují výsledky v kapitole 4.10 o testování, vychází v poměru kvalita-cena-čas nejlépe.

²¹<https://pypi.org/project/duckduckgo-search/>

²²<https://streamlit.io/>



Obrázek 4.8: Testovací uživatelské rozhraní.



Obrázek 4.9: Produkční uživatelské rozhraní.

4.10 Testování a evaluace

Testování a evaluace aplikací má zásadní význam pro zajištění jejich efektivity a spolehlivosti. Pro zajištění dostatečné komplexity vyhodnocení jsem se pro vyhodnocení rozhodla použít jak RAGAs, knihovnu popsanou v kapitole 3.3 o evaluaci RAG, tak vyhodnocení na základě testování s prvotními uživateli.

Užitečnost využití RAGAs při hodnocení spočívá v tom, že poskytuje standardizovaný systém pro hodnocení výkonnosti a efektivity RAG pipeline, který umožňuje kvantifikaci a porovnání výkonnosti pipeline.

Souběžně se začleněním RAGAs hraje klíčovou roli při hodnocení účinnosti a použitelnosti aplikace RAG uživatelské testování. Testující uživatele slouží jako neocenitelný zdroj při identifikaci potenciálních nedostatků, problémů s použitelností a oblastí, které RAGAs nemusí odhalit a je třeba jim věnovat pozornost. Zapojením testovacích uživatelů do procesu vyhodnocování získávám totiž i informace o preferencích, chování a potřebách uživatelů, což pomáhá zajistit, aby byla aplikace orientovaná na uživatele – intuitivní a v souladu s očekáváními a požadavky cílové skupiny.

Toto testování jsem prováděla na stejných LLM, jako je popsáno v kapitole 4.4 o vybraných jazykových modelech, avšak s vynecháním modelu llama2-70b, který k těmto úlohám není uzpůsobený, a modelu gpt-4, který je výrazně dražší než model gpt-3.5, avšak nedosahuje markantně lepších výsledků.

4.10.1 Vyhodnocení pomocí RAGAs

Pro vyhodnocení s použitím RAGAs jsem zvolila stejnou datovou sadu jako pro hodnocení vyhledávání ve vektorové databázi. Tato sada má 100 otázek různé složitosti napříč všemi třídami. Ke každé otázce náleží kontrolní odpověď `ground_truth`.

Vybrané modely jsem s použitím funkce `choose_tools` nechala otázky zodpovědět, přičemž jsem sbírala i kontext, který byl LLM pro vytvoření odpovědi předán. Vstupní dataset pro RAGAs musí dodržovat následující formát:

```
data_samples = {
    'question': list[str],
    'answer': list[str],
    'contexts' : list[list[str]],
    'ground_truth': list[str]
}
dataset = Dataset.from_dict(data_samples)
```

Výpis 4.9: Formát vstupních dat pro RAGAs evaluaci.

Seznam všech RAGAs metrik a jejich vlastností je k dispozici v oficiální RAGAs dokumentaci metrik²³. Metrikami, které jsem zvolila pro tuto evaluaci, jsou přesnost kontextu (Context Precision), relevantnost odpovědi (Answer Relevancy), věrohodnost (Faithfulness) a vyvolání kontextu (Context Recall).

Přesnost kontextu je metrika vyhodnocující, zda jsou všechny relevantní položky přítomné v kontextech hodnoceny výše, či nikoli. Optimálně by měly všechny důležité části

²³<https://docs.ragas.io/en/latest/concepts/metrics/index.html>

dosáhnout nejvyšších hodnot. Souvisí s kvalitou přeformulování uživatelské otázky pro vyhledávání relevantních informací. Tato metrika se vypočítává pomocí otázek, kontrolních odpovědí a kontextů, a její hodnoty se pohybují od 0 do 1; vyšší skóre značí vyšší přesnost. Výpočet lze vidět v rovnicích 4.2 a 4.3, přičemž K je celkový počet úryvků v kontextech a v_k je indikátor relevance na úrovni k .

$$\text{Přesnost kontextu@K} = \frac{\sum_{k=1}^K (\text{Přesnost @k} \times v_k)}{\text{Celkový počet relevantních položek v top } K \text{ výsledcích}} \quad (4.2)$$

$$\text{Přesnost@k} = \frac{\text{pravdivě pozitivní výsledky @k}}{(\text{pravdivě pozitivní výsledky @k} + \text{falešně pozitivní nálezy @k})} \quad (4.3)$$

Metrika pro hodnocení relevance odpovědi se zase zaměřuje na zhodnocení, jak relevantní je vygenerovaná odpověď k danému dotazu. To záleží na schopnosti celé RAG pipeline. Vyšší skóre zde znamená větší relevanci. Tato metrika při výpočtu zohledňuje otázku, kontext a odpověď a je definována jako relativní cosinová podobnost původní otázky s řadou umělých otázek, které byly vygenerovány (reverzní inženýrství) na základě odpovědi, jak demonstruje rovnice 4.4:

$$\text{relevance odpovědi} = \frac{1}{N} \sum_{i=1}^N \cos(E_{g_i}, E_o) \quad (4.4)$$

Přičemž E_{g_i} je embedding vygenerované otázky i , E_o je embedding původní otázky a N je počet vygenerovaných otázek, který je standardně 3.

Věrohodnost měří, věcnou shodnost vygenerované odpovědi s daným kontextem, což záleží na schopnosti LLM využít kontext pro generování odpovědi. Vypočítá se na základě odpovědi a načteného kontextu, přičemž skóre se pohybuje od 0 do 1, kde vyšší skóre znamená lepší věrohodnost. Věrohodná odpověď uvádí tvrzení, která lze odvodit přímo z kontextu, což se určí porovnáním každého tvrzení v odpovědi s poskytnutým kontextem.

Poslední zvolená metrika vyvolání kontextu hodnotí, jak dobře se načtený kontext shoduje s kontrolní odpovědí. To souvisí jak s transformací dotazu pro vyhledávání v databázi, tak samotným hledáním. Skóre se pohybuje od 0 do 1, přičemž vyšší skóre znamená lepší shodu. U každé věty v kontrolní odpovědi se zkoumá, zda ji lze spojit s vyhledaným kontextem. V ideálním případě by měly být všechny věty v kontrolní odpovědi přiřaditelné k vyhledanému kontextu.

| Model | Přesnost kontextu | Relevance odpovědi | Věrohodnost | Vyvolání kontextu | Čas |
|------------|-------------------|--------------------|-------------|-------------------|--------|
| llama3-70b | 0.803 | 0.899 | 0.868 | 0.689 | 2m 8s |
| llama3-8b | 0.708 | 0.818 | 0.820 | 0.561 | 2m 6s |
| mixtral | 0.835 | 0.899 | 0.924 | 0.648 | 1m 48s |
| gpt-3.5 | 0.886 | 0.925 | 0.864 | 0.699 | 1m 02s |

Tabulka 4.13: Srovnání výkonnosti modelů vybranými metrikami.

Průměr všech 100 výsledků pro každý model ke všem metrikám zároveň s průměrným časem odpovědi ukazuje tabulka 4.13. Lze vidět, že výsledky jsou dosti vyrovnané i když různé modely vykazují v těchto metrikách různou úroveň výkonnosti. Například model llama3-70b vykazuje dobrou přesnost kontextu a relevanci odpovědi, ale o něco nižší

věrohodnost a vyvolání kontextu ve srovnání s modelem gpt-3.5, který vykazuje vyšší celkový výkon ve všech metrikách. Jazykové modely llama3-8b a mixtral se nacházejí přibližně uprostřed a vykazují slušné výsledky, ale ne tak vysoké jako model gpt-3.5.

Pro další srovnání jsem rozdělila tuto datovou sadu na jednodušší a složitější otázky, přičemž jednodušší otázka se týká základních informací a složitější otázka má mnohdy i více předmětů. Výsledky modelů na jednodušších dotazech lze vidět v tabulce 4.14. Všechny

| Model | Přesnost kontextu | Relevance odpovědi | Věrohodnost | Vyvolání kontextu |
|------------|-------------------|--------------------|-------------|-------------------|
| llama3-70b | 0.939 | 0.928 | 0.872 | 0.836 |
| llama3-8b | 0.881 | 0.874 | 0.842 | 0.712 |
| mixtral | 0.913 | 0.923 | 0.974 | 0.848 |
| gpt-3.5 | 0.960 | 0.951 | 0.899 | 0.791 |

Tabulka 4.14: Srovnání výkonnosti modelů na jednodušších dotazech.

modely jsou dle očekávání oproti průměru všech výsledků lepší ve všech oblastech. Model llama3-70b vykazuje výrazně lepší výsledky přesnosti kontextu, a to z hodnoty 0,803 na 0,939. Ačkoli ve věrohodnosti jinde nejsou velké rozdíly, mixtral vykazuje výrazné zlepšení. Nejlepších výsledků stále dosahují modely llama3-70b a gpt-3.5.

V tabulce srovnání modelů na složitějších dotazech 4.15 si lze všimnout, že jsou výsledky vyvolání kontextu velmi vyrovnané. Mimo relevanci odpovědi zde ve výsledcích pochopitelně došlo ke znatelnému zhoršení.

| Model | Přesnost kontextu | Relevance odpovědi | Věrohodnost | Vyvolání kontextu |
|------------|-------------------|--------------------|-------------|-------------------|
| llama3-70b | 0.652 | 0.871 | 0.849 | 0.615 |
| llama3-8b | 0.598 | 0.805 | 0.821 | 0.542 |
| mixtral | 0.658 | 0.869 | 0.902 | 0.611 |
| gpt-3.5 | 0.631 | 0.901 | 0.890 | 0.624 |

Tabulka 4.15: Srovnání výkonnosti modelů na složitějších dotazech.

Na základě této evaluace s použitím RAGAs je možné s jistotou říci, že nejlepších výsledků dosahují modely llama3-70b a gpt-3.5, přičemž výsledky modelu mixtral oproti nim vykazují jen mírné zhoršení. Je zajímavé, že rozdíl mezi open-source modely a gpt není až tak patrný. Mimo metriky je však nutné vzít v úvahu i čas a cenu odpovědi. Jelikož se požadavky na systém mohou lišit, obecně nejvhodnější model nelze stanovit.

4.10.2 Testování s uživateli

Pro vykonání testování s uživateli se mi podařilo přemluvit 13 účastníků. Testování probíhalo zejména individuálně, kdy účastník dostal aplikaci k vyzkoušení na mém počítači, s použitím experimentálního uživatelského rozhraní, a to ať už fyzicky, či přes vzdálené připojení.

Každý účastník hodnotil všechny vybrané modely, přičemž 10 účastníků se ptalo v anglickém jazyce a 3 v češtině. Pro hodnocení jsem vytvořila formulář pomocí služby Google

Forms. Ve formuláři je na výběr z předem daných jazykových modelů, a dále se skládá ze tří hlavních částí:

1. dotaz vztahující se ke kulturním akcím,
2. dotaz k čemukoliv jinému vztahujícímu k městu Brnu,
3. irelevantní dotaz.

Hlavní komponenta formuláře je ukázána na obrázku 4.10. Tato komponenta je stejná pro oba typy relevantního dotazu, jedná se o hodnocení relevance odpovědi, spokojenosti s odpovědí a spokojenosti s rychlostí odezvy na stupnici 1–5, kde 1 znamená nehorší a 5 nejlepší. Hodnocení irelevantních dotazů je podobné, místo spokojenosti s odpovědí se však hodnotí korektnost chování asistenta.

The form consists of several sections:

- Vybraný model ***: A dropdown menu with the text "Vyberte" and a downward arrow.
- Otázka uživatele ***: A text input field with the label "Vaše odpověď".
- Asistentova odpověď ***: A text input field with the label "Vaše odpověď".
- Relevance odpovědi ***: A Likert scale with five radio buttons labeled 1, 2, 3, 4, and 5. The scale is anchored with "Irelevantní" on the left and "Relevantní" on the right.
- Spokojenost s odpovědí ***: A Likert scale with five radio buttons labeled 1, 2, 3, 4, and 5. The scale is anchored with "Nespokojený" on the left and "Spokojený" on the right.
- Spokojenost s rychlostí odezvy**: A Likert scale with five radio buttons labeled 1, 2, 3, 4, and 5. The scale is anchored with "Nespokojený" on the left and "Spokojený" on the right.

Obrázek 4.10: Hlavní komponenta formuláře pro hodnocení respondenty.

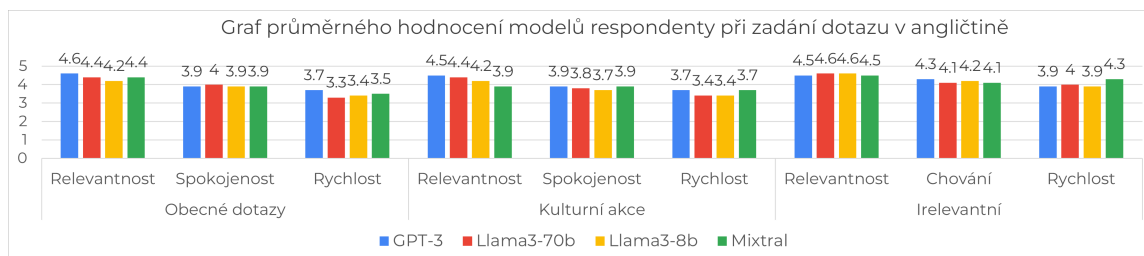
| | Obecné dotazy | | | Kulturní akce | | | Irelevantní | | |
|------------|---------------|-------------|----------|---------------|-------------|----------|--------------|---------|----------|
| | Relevantnost | Spokojenost | Rychlost | Relevantnost | Spokojenost | Rychlost | Relevantnost | Chování | Rychlost |
| gpt-3.5 | 4,6 | 3,9 | 3,7 | 4,5 | 3,9 | 3,7 | 4,5 | 4,3 | 3,9 |
| llama3-70b | 4,4 | 4 | 3,3 | 4,4 | 3,8 | 3,4 | 4,6 | 4,1 | 4 |
| llama3-8b | 4,2 | 3,9 | 3,4 | 4,2 | 3,7 | 3,4 | 4,6 | 4,2 | 3,9 |
| mixtral | 4,4 | 3,9 | 3,5 | 3,9 | 3,9 | 3,7 | 4,5 | 4,1 | 4,3 |

Tabulka 4.16: Průměrné hodnocení modelů uživateli při zadání dotazu v angličtině.

| | Obecné dotazy | | | Kulturní akce | | | Irelevantní | | |
|------------|---------------|-------------|----------|---------------|-------------|----------|--------------|---------|----------|
| | Relevantnost | Spokojenost | Rychlost | Relevantnost | Spokojenost | Rychlost | Relevantnost | Chování | Rychlost |
| gpt-3.5 | 4,7 | 3,7 | 3,7 | 4,7 | 3,7 | 3,7 | 4,7 | 4,7 | 4,3 |
| llama3-70b | 4,3 | 3,3 | 3,3 | 4,3 | 3,3 | 3,3 | 3,7 | 4,3 | 3,7 |
| llama3-8b | 3,7 | 3,0 | 3,7 | 4,3 | 2,7 | 3,7 | 4,7 | 4,3 | 3,7 |
| mixtral | 4,0 | 2,7 | 3,3 | 3,3 | 3,3 | 4,0 | 4,7 | 4,3 | 4,0 |

Tabulka 4.17: Průměrné hodnocení modelů uživateli při zadání dotazu v češtině.

Průměry hodnocení dotazů v anglickém jazyce shrnuje tabulka 4.16. Výsledky vykazují podobný trend jako hodnocení RAGAs, kdy mezi modely nejsou dramatické rozdíly, ale model gpt-3.5 dosahuje nejlepšího hodnocení, což vizualizuje graf v obrázku 4.11. Dotazy na kulturní akce dopadly hůře než ostatní, překvapivě dobře se systém vypořádává s irelevantními dotazy.



Obrázek 4.11: Graf hodnocení modelů respondenty při zadání dotazu v angličtině.

Celkově lze pozorovat, že pro dotazy v anglickém jazyce gpt-3.5 a llama3-70b dosahují nejlepších výsledků ve všech třech metrikách a kategoriích, zatímco mixtral vykazuje také konkurenceschopný výkon.

Naproti tomu hodnocení uživatelů u dotazů v češtině, které popisuje tabulka 4.17, vykazuje znatelný pokles. V anglických dotazech dosahoval gpt-3.5 konzistentně vyššího skóre než v českých dotazech ve všech kategoriích, s výjimkou irelevantních dotazů, kde dosáhl stejného skóre. Model gpt-3.5 si však stále udržuje náskok, llama3-70b, llama3-8b a mixtral taktéž vykazují pokles výkonnosti s nižším hodnocením relevance, spokojenosti a rychlosti v porovnání se svými anglickými protějšky.

Aplikace si zasloužila průměrné hodnocení 4,1/5. Na konci formuláře byli testovací uživatelé dotázáni na celkový pocit z aplikace, zde systém získal ohodnocení 3,95/5. Zároveň respondenti dostali možnost přidat osobní komentář. Zde se objevily 3 výstupy, přičemž 2 z nich chválily vzhled aplikace a 1 kritizoval dobu odezvy.

Celkově nejlépe hodnotili respondenti relevantnost odpovědí, kde prvotřídně dopadl model gpt-3.5, ostatní modely dopadly srovnatelně. Zatímco se samotnými odpověďmi byli uživatelé spokojeni, s časem jejich vykonání už tolik ne. Zde je nutné podotknout, že čas závisí na rychlosti odpovědi LLM, což nelze úplně ovlivnit, avšak lze optimalizovat ostatní kroky a experimentovat se službami, jež modely poskytují.

Pozornost přitahuje rozdílnost kvality výsledků mezi anglickým a českým jazykem. Když jsem rozhodovala, jakým způsobem řešit multilinguálnost, porovnávala jsem aktuální možnosti open-source vícejazyčných embedding modelů a modelů na překlad. Tehdejší výstupem bylo, že se použití nevyplatí, jelikož mají tyto modely značně nižší výkonnost. A jelikož jsou cílovou skupinou uživatelů aplikace zejména cizinci, rozhodla jsem se oproti vyrovnání s češtinou upřednostnit výkon pro anglický jazyk.

Dalším provedeným pokusem bylo srovnání odpovědi aplikace s odpověďmi ChatGPT²⁴. Bylo provedeno celkem 26 dotazů v angličtině, kde se 13 týkalo informací spojených s městem Brnem a 13 s kulturními akcemi ve městě Brně. Pro dotazy ohledně města Brna byla odpověď systému označena respondenty jako ta lepší v 65 % případů. Dotazy týkající se kulturních akcí dopadly pro vyvíjený systém lépe v 84 % případů.

²⁴<https://chatgpt.com/>

Kapitola 5

Závěr

Cílem této diplomové práce bylo vytvořit komunikačního agenta poskytujícího uživatelům informace o městě Brně. Výsledkem je tedy funkční dialogový systém, který může být využit jak turisty hledajícími informace o Brně, tak i místními obyvateli zajímajícími se o kulturní události nebo jiné informace týkající se jejich města.

V práci popisuji možnosti vytváření komunikačních agentů a srovnávám řešení potřeby pro více informací, než obsahují trénovací data velkých jazykových modelů. Aby bylo zajištěno, že má systém komunikačního agenta k dispozici potřebné informace, je postaven na bázi RAG, tedy spojení jazykového modelu a vektorové databáze. Systém navíc obsahuje komponenty pro získávání dat sestávající ze získávání informací, jejich klasifikace a strukturování pomocí syntaktické analýzy, přičemž se získaná data spolu s dalšími parametry ukládají do příslušných znalostních databází. Jádrem implementace je agent pro komunikaci s velkými jazykovými modely, který podporuje jak zpoplatněná API, tak open-source modely spouštěné lokálně. Tento klíčový prvek je kromě komunikace s uživatelem používán i při zpracování dat a volbě postupu pro generování odpovědi, evaluaci získaných dat pro odpověď a rozhoduje o potřebě vyhledávání dalších informací na internetu.

Nad rámec zadání v práci srovnávám schopnosti vybraných jazykových modelů na všech úlohách, ve kterých je systém na modelech závislý. Zároveň jsem provedla komplexní vyhodnocení celého systému s použitím RAGAs, kde s některými open-source jazykovými modely dosahuji výsledků srovnatelných v výkonem modelů GPT. Výslednou aplikaci jsem testovala také s pomocí testovacích uživatelů. Respondenti chválili vzhled aplikace, relevantnost i obsah odpovědí pro anglický i český jazyk, přičemž testování v angličtině dopadlo lépe. Aplikace si zasloužila průměrné hodnocení respondentů 4,1 z pěti. Výsledky práce prezentuje i vytvořený plakát, jehož náhled je k vidění v příloze E.

Možným rozšířením práce je integrace nástrojů, které byly představeny až po jejím vzniku. To může přinést optimalizaci systému v mnoha aspektech, jako je i čas vygenerované odpovědi, který však závisí zejména na schopnostech jazykových modelů. Navázat lze i novým průzkumem vícejazyčných open-source modelů a vyrovnat tak rozdíly ve výkonnosti systému mezi anglickým a českým jazykem.

Z osobního pohledu mohu říci, že tato práce splnila má očekávání v oblasti získaných znalostí a zkušeností. Vybrala jsem si ji, protože mě toto odvětví zajímalo, a to i přes to, že jsem k danému tématu neměla takřka žádné počáteční znalosti. A právě díky ní jsem se rozhodla, že se tomuto odvětví budu věnovat i nadále.

Literatura

- [1] BESBES, A. *9 Effective Techniques To Boost Retrieval Augmented Generation (RAG) Systems*. 2024. Dostupné z: <https://medium.com/towards-data-science/9-effective-techniques-to-boost-retrieval-augmented-generation-rag-systems-210ace375049>.
- [2] BROGGS, J. *Getting Started with Hybrid Search*. 2023. Dostupné z: <https://www.pinecone.io/learn/hybrid-search-intro/>.
- [3] CARDENAS, E. *Hybrid Search Explained*. 2023. Dostupné z: <https://weaviate.io/blog/hybrid-search-explained>.
- [4] CHAUBARD, F.; MUNDRA, R. a SOCHER, R. *CS 224D: Deep Learning for NLP - Lecture Notes: Part I*. Stanford University, 2016. Dostupné z: http://cs224d.stanford.edu/lecture_notes/notes1.pdf.
- [5] CHEN, H.; PASUNURU, R.; WESTON, J. a CELIKYILMAZ, A. *Walking Down the Memory Maze: Beyond Context Limit through Interactive Reading*. 2023.
- [6] CHEN, T.; WANG, H.; CHEN, S.; YU, W.; MA, K. et al. *Dense X Retrieval: What Retrieval Granularity Should We Use?* 2023.
- [7] FRÖBERG, E. *Picking a vector database: a comparison and guide for 2023*. 2023. Dostupné z: <https://benchmark.vectorview.ai/vector dbs.html>.
- [8] GAO, L.; MA, X.; LIN, J. a CALLAN, J. *Precise Zero-Shot Dense Retrieval without Relevance Labels*. 2022.
- [9] GAO, Y.; XIONG, Y.; GAO, X.; JIA, K.; PAN, J. et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024.
- [10] HAM, L. *Why is Vector Search so fast?* 2022. Dostupné z: <https://weaviate.io/blog/why-is-vector-search-so-fast>.
- [11] HOTZ, H. *RAG vs Finetuning — Which Is the Best Tool to Boost Your LLM Application?* 2023. Dostupné z: <https://towardsdatascience.com/rag-vs-finetuning-which-is-the-best-tool-to-boost-your-llm-application-94654b1eaba7>.
- [12] HUGET, M.-P. Agent Communication. In: SHEHORY, O. a STURM, A., ed. *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, s. 101–133. ISBN 978-3-642-54432-3.

- [13] IBRAHIM, M. *An Overview of Large Language Models (LLMs)*. 2023. Dostupné z: <https://wandb.ai/mostafaibrahim17/ml-articles/reports/An-Overview-of-Large-Language-Models-LLMs---VmlldzozODA3MzQz>.
- [14] ILIN, I. *Advanced RAG Techniques: an Illustrated Overview*. 2023. Dostupné z: <https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6>.
- [15] JAGERMAN, R.; ZHUANG, H.; QIN, Z.; WANG, X. a BENDERSKY, M. *Query Expansion by Prompting Large Language Models*. 2023.
- [16] JIANG, H.; WU, Q.; LUO, X.; LI, D.; LIN, C.-Y. et al. *LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression*. 2023.
- [17] JURAFSKY, D. a MARTIN, J. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 3. vyd. únor 2024.
- [18] KARPUKHIN, V.; OĞUZ, B.; MIN, S.; LEWIS, P.; WU, L. et al. *Dense Passage Retrieval for Open-Domain Question Answering*. 2020.
- [19] KEARY, T. *12 Practical Large Language Model (LLM) Applications*. 2024. Dostupné z: <https://www.techopedia.com/12-practical-large-language-model-llm-applications>.
- [20] LEWIS, P.; PEREZ, E.; PIKTUS, A.; PETRONI, F.; KARPUKHIN, V. et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021.
- [21] LI, X.; LIU, Z.; XIONG, C.; YU, S.; GU, Y. et al. *Structure-Aware Language Model Pretraining Improves Dense Retrieval on Structured Data*. 2023.
- [22] LITMAN, R.; ANSHEL, O.; TSIPER, S.; LITMAN, R.; MAZOR, S. et al. *SCATTER: Selective Context Attentional Scene Text Recognizer*. 2020.
- [23] MA, C. *Your RAG Needs Some Scaffolding*. 2023. Dostupné z: <https://medium.com/@machangsha/your-rag-needs-some-scaffolding-faae72234798>.
- [24] MA, X.; GONG, Y.; HE, P.; ZHAO, H. a DUAN, N. *Query Rewriting for Retrieval-Augmented Large Language Models*. 2023.
- [25] MANGRULKAR, S. a PAUL, S. *PEFT: Parameter-Efficient Fine-Tuning of Billion-Scale Models on Low-Resource Hardware*. 2023. Dostupné z: <https://huggingface.co/blog/peft>.
- [26] MARVIN, G.; HELLEN, N.; JJINGO, D. a NAKATUMBA NABENDE, J. Prompt Engineering in Large Language Models. In: JACOB, I. J.; PIRAMUTHU, S. a FALKOWSKI GILSKI, P., ed. *Data Intelligence and Cognitive Informatics*. Singapore: Springer Nature Singapore, 2024, s. 387–402. ISBN 978-981-99-7962-2.
- [27] MIKOLOV, T.; SUTSKEVER, I.; CHEN, K.; CORRADO, G. a DEAN, J. *Distributed Representations of Words and Phrases and their Compositionality*. 2013.

- [28] MULKAR, A. *How Large Language Models Work: Unveiling the Transformer Architecture*. 2023. Dostupné z: <https://ankushmulkar.medium.com/how-large-language-models-work-unveiling-the-transformer-architecture-12d9e604848b>.
- [29] PATHAK, P. *Large Language Models 101: History, Evolution and Future*. 2023. Dostupné z: <https://www.scribbledata.io/blog/large-language-models-history-evolutions-and-future/>.
- [30] POTH, C.; STERZ, H.; PAUL, I.; PURKAYASTHA, S.; ENGLÄNDER, L. et al. *Adapters: A Unified Library for Parameter-Efficient and Modular Transfer Learning*. 2023.
- [31] RAROQUE, C. *5+ Large Language Model Applications to Use For 2024*. 2024. Dostupné z: <https://aloo.co/blog/large-language-model-applications>.
- [32] RAUDASCHL, A. H. *Forget RAG, the Future is RAG-Fusion*. 2023. Dostupné z: <https://towardsdatascience.com/forget-rag-the-future-is-rag-fusion-1147298d8ad1>.
- [33] RAVFOGEL, S.; PYATKIN, V.; COHEN, A. D.; MANEVICH, A. a GOLDBERG, Y. *Retrieving Texts based on Abstract Descriptions*. 2023.
- [34] REIMERS, N. *Pretrained Models*. 2024. Dostupné z: https://www.sbert.net/docs/pretrained_models.html.
- [35] REZNIKOV, I. *How exactly LLM generates text?* 2023. Dostupné z: <https://www.linkedin.com/pulse/how-exactly-llm-generates-text-ivan-reznikov/>.
- [36] SAEED, M. *A Gentle Introduction to Positional Encoding in Transformer Models, Part 1*. 2023. Dostupné z: <https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/>.
- [37] SHIN, J.; TANG, C.; MOHATI, T.; NAYEBI, M.; WANG, S. et al. *Prompt Engineering or Fine Tuning: An Empirical Assessment of Large Language Models in Automated Software Engineering Tasks*. 2023.
- [38] VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L. et al. *Attention Is All You Need*. 2023.
- [39] WANG, Z.; NG, P.; NALLAPATI, R. a XIANG, B. Retrieval, Re-ranking and Multi-task Learning for Knowledge-Base Question Answering. In: MERLO, P.; TIEDEMANN, J. a TSARFATY, R., ed. *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Online: Association for Computational Linguistics, Duben 2021, s. 347–357. Dostupné z: <https://aclanthology.org/2021.eacl-main.26>.
- [40] XU, P.; PING, W.; WU, X.; MCAFEE, L.; ZHU, C. et al. *Retrieval meets Long Context Large Language Models*. 2023.

Příloha A

Vzorové použití aplikace

Persona: Marco Español

Věk: 54 | Občanství: Španělsko | Role: Turista

Q: *What are some must-visit landmarks and attractions in Brno, Czech Republic?*

A: Brno, in the Czech Republic, is a city with a rich history and vibrant culture. Some top destinations to visit there include:

1. Špilberk Castle: A historic fortress with a museum showcasing the city's history
 2. Villa Tugendhat: An iconic UNESCO-listed modernist building
 3. Cathedral of St. Peter and Paul: A stunning Gothic cathedral dominating the city skyline
 4. Brno City Center: Explore the vibrant streets, squares like Freedom Square
 5. Labyrinth Under Vegetable Market: A network of underground passages and cellars
- [<https://www.gotobrna.cz/en/explore-brno/go-to-brnos-top-destinations/>]

Persona: Andrea Nováková

Věk: 23 | Občanství: Česká republika | Role: Obyvatelka Brna

Q: *Kde lze v Brně založit živnost?*

A: Pro založení živnosti v Brně máte několik možností:

1. Osobní návštěva: Adresa pro osobní podání je Malinovského nám. 624/3, 602 00 Brno.
 2. Elektronické podání: Podání lze vyřídit elektronicky prostřednictvím aplikace JRF (Jednotný registrační formulář) na elektronickou podatelnu živnostenského rejstříku (RŽP), do datové schránky Magistrátu města Brna (a7kbrnn), nebo na e-mailovou adresu podatelny MMB (posta@brno.cz). Pokud podáváte žádost poštou, vyplňte podání v aplikaci JRF, podepište žádost a odešlete ji na adresu podatelny Magistrátu města Brna (Malinovského nám. 3, 601 67 Brno).
 3. Elektronické objednání: Pro elektronické objednání navštivte kontaktní místo Czech POINT na poště a vyřídte podání žádosti v listinné nebo elektronické podobě.
- [<https://www.brno.cz/w/vznik-zivnostenskeho-opravneni>]

Persona: Pablo Italiano

Věk: 45 | Občanství: Itálie | Role: Turista

Q: *Where to eat good pasta in Brno?*

A: If you're craving good pasta in Brno, you might want to check out Butcher's Grill & Pasta. Despite being known for its American and Asian cuisines, this spot offers a selection of pasta dishes, including gluten-free options upon request. They've mastered BBQ, but their pasta, salads, and soups are also worth trying! [https://www.gotobrna.cz/wp-content/uploads/2023/10/GB23_pruvodce-A6_EN_dvoustrany_web.pdf]

Persona: Ivan Hrvatski

Věk: 28 | Občanství: Chorvatsko | Role: Expat

Q: *Do I need a residence permit to move to Brno if I am an EU citizen?*

A: All citizens of the EU, Iceland, Norway, Liechtenstein or Switzerland can enter and stay in the Czech Republic without any special permit, solely on the basis of a travel document or an identity card. [<https://www.brnoexpatcentre.eu/im-an-expat/visa-residence/>]

Příloha B

Vzorové dotazy

1. What is the population of Brno, Czech Republic?
2. What are the top universities in Brno, Czech Republic?
3. What is the origin of the name "Brno"?
4. What are some popular art galleries in Brno?
5. What is the history of the Špilberk Castle in Brno?
6. Can you provide information about the transportation system in Brno?
7. What are the notable architectural landmarks in Brno?
8. What events are taking place in Brno this weekend?
9. When was the University of Technology founded?
10. Who is the architect of Villa Tugendhat?
11. Where was Leoš Janáček born?
12. What are some must-visit landmarks and attractions in Brno, Czech Republic?
13. Do I need a residence permit to move to Brno if I am an EU citizen?
14. How is the weather and climate in Brno throughout the year?
15. What are the notable historical events associated with Brno?
16. Are there any public libraries in Brno?
17. Are there any specific requirements for obtaining a driver's license in Brno?
18. Can you suggest me a theatrical play for this weekend?
19. What are the requirements for obtaining a work permit in Brno for non-EU citizens?
20. How does the process of registering with the local municipality as a resident in Brno work for newcomers?
21. What are the options for obtaining a public transport pass in Brno?

Příloha C

Zdroje informací pro databázi

Zde se nachází seznam zdrojů, které tvoří základ pro sběr dat. Tyto adresy jsou procházeny, jsou z nich získávána data a sbírány rozšiřující url adresy. Vyhovují-li nově nalezené adresy daným parametrům, je s nimi proces opakován. Takto sběr probíhá po dobu předem nastaveného počtu iterací.

- <https://www.gotobrno.cz/en/>
- <https://www.brno.cz/>
- <https://www.brnoexpatcentre.eu/>
- <https://encyklopedie.brna.cz/home-mmb/>
- <https://en.wikipedia.org/wiki/Brno>
- <https://www.mzm.cz/en/>
- <https://moravska-galerie.cz/?lang=en>
- <https://www.tnbrno.cz/en/>
- <https://divadlo.net/brno/akce/>
- <https://www.brno-stred.cz/kalendar-akci>
- <https://www.kultura365.cz/?okres=brno-mesto>
- <https://www.kudyznudy.cz/vyhledavani?tag=Brno>
- <https://www.informuji.cz/akce/jhm/dnes/>
- <https://www.ticbrno.cz/>
- <https://www.muni.cz/>
- <https://www.vut.cz/>
- <https://www.bts.cz/sporty.asp>

Příloha D

Manuál

Aplikace byla vyvíjena v jazyce Python 3.10 a je kompatibilní i s verzí 3.9. Pro správné fungování projektu je nutné nainstalovat všechny potřebné balíčky, to lze díky souboru `requirements.txt` provést jednoduše příkazem `pip install -r requirements.txt`

Zároveň je potřebný lokálně běžící MySQL server pro znalostní databázi zdrojů informací. Pokyny pro stažení a instalaci jsou dostupné na [oficiálních stránkách MySQL](#). Pro lokální spuštění Weaviate vektorové databáze je nutné mít nainstalovaný [Docker](#).

Tato aplikace je závislá na připojení k serveru s velkým jazykovým modelem, proto je nutné v konfiguracích správně nastavit url adresu k serveru a API klíč, je-li pro model potřeba. Zároveň je nutné poskytnout API klíče pro LlamaCloud a Cohere re-ranker.

Pro nastavení choulostivých údajů, jako jsou API klíče a přístup k databázi, je v projektu používán `.env` soubor, který je potřeba vytvořit dle přiloženého `env.example`, který obsahuje:

```
LLAMA_CLOUD_API_KEY="<llx-key>"
OPENAI_API_KEY="<sk-key>"
LLAMA_API_KEY="<LL-key>"
LOCAL_API_KEY="local"
COHERE_APIKEY="<cohere-key>"

DB_HOST=<host>
DB_USER=<user>
DB_PASSWORD=<password>
DATABASE=<db_name>
```

Výpis D.1: Požadované proměnné prostředí.

Dalším nastavením, které je potřeba provést, je obsah souboru `cfq/config.ini`. Mimo jiné jsou v tomto souboru nastaveny cesty k složkám s PDF a souborům s daty pro inicializaci sběru dat, a právě zde jsou definované i url k API:

```
[API_INFOS]
LLAMA_URL = https://api.llama-api.com
OPENAI_URL = https://api.openai.com/v1
LOCAL_URL = http://localhost:8801/v1/
```

Výpis D.2: Požadované proměnné prostředí.

Sběr dat

Prvním krokem je vytvoření SQL databáze, to lze provést příkazem:

```
python src/data_acquisition/sources_store/sources_db.py
```

Zároveň je potřeba mít spuštěnou vektorovou databázi:

```
docker-compose up -d
```

Prvotní zahájení sběru lze docílit instrukcí:

```
python src/data_acquisition/data_acquisition_manager.py
```

Následně je potřeba zahájit CRON operace, k čemuž je soubor `cron_files/cron_setup.sh`. Spustit ho lze např. takto:

```
sudo bash cron_files/cron_setup.sh
```

Takto je provedeno nastavení automatického sběru a aktualizace dat, data jsou ukládána do vektorové databáze, odkud následně agent čerpá.

Aplikace

Není-li tak, je nutné spustit vektorovou databázi:

```
docker-compose up -d
```

Dialogovou aplikaci lze pak spustit následovně:

```
streamlit run src/app/production.py
```

Tím se v okně prohlížeče spustí dialogová aplikace, se kterou lze konverzovat. Možnosti spuštění a nastavení aplikace jsou také popsány v příloženém souboru `README.md`.

Příloha E

Plakát

Komunikační agent pro informace o Brně

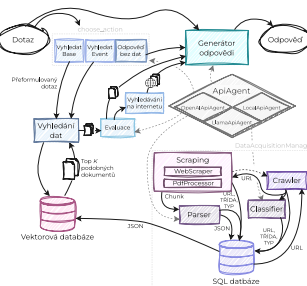
Autor: Bc. Kateřina Neprašová | Vedoucí práce: Smrž Pavel, doc. RNDr., Ph.D.

Cíl práce:

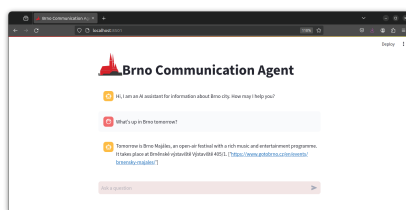
Tato diplomová práce se zaměřuje na doménově specifické komunikační agenty, cílem je vytvořit funkčního komunikačního agenta pro turisty i místní obyvatele Brna, poskytujícího relevantní a aktuální informace podle individuálních potřeb uživatelů.

Návrh systému:

Aby bylo zajištěno, že má komunikační agent k dispozici potřebné informace, je postaven na generování s rozšířeným vyhledáváním (RAG, Retrieval-Augmented Generation), tedy spojení jazykového modelu a vektorové databáze. Systém navíc obsahuje komponenty pro získávání dat, která se spolu s dalšími parametry ukládají do příslušných znalostních databází. Jádrem implementace je agent pro komunikaci s velkými jazykovými modely, který podporuje jak zpoplatněná API, tak open-source modely spouštěné lokálně. Tento klíčový prvek je kromě komunikace s uživatelem používán i při zpracování dat a volbě postupu pro generování odpovědi, evaluaci získaných dat pro odpověď a rozhoduje o potřebě vyhledávání dalších informací na internetu.



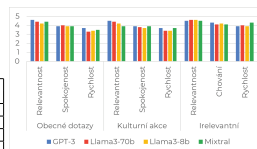
Výsledná aplikace:



Výsledky evaluace:

| Model | Přesnost kontextu | Relevance odpovědi | Věrohodnost | Vývolání kontextu |
|------------|-------------------|--------------------|-------------|-------------------|
| llama3-70b | 0.803 | 0.899 | 0.898 | 0.689 |
| llama3-8b | 0.708 | 0.818 | 0.820 | 0.561 |
| mistral | 0.835 | 0.899 | 0.924 | 0.648 |
| gpt-3.5 | 0.886 | 0.925 | 0.864 | 0.699 |

Srovnání výkonnosti modelů vybranými RAGAs metrikami:



Průměrné hodnocení odpovědi systému respondenty dle modelu

Obrázek E.1: Náhled plakátu prezentujícího cíle a výsledky této práce.