



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

**ZPRACOVÁNÍ A VYHLEDÁVÁNÍ DOKUMENTŮ
S VYUŽITÍM VEKTOROVÝCH DATABÁZÍ A JAZY-
KOVÉHO MODELU**

PROCESSING AND RETRIEVAL OF TEXT DOCUMENTS WITH USE OF VECTOR DATABASES
AND A LANGUAGE MODEL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ADAM VALÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VLADIMÍR BARTÍK, Ph.D.

BRNO 2025

Zadání bakalářské práce



Ústav: Ústav informačních systémů (UIFS) 163475
Student: **Valík Adam**
Program: Informační technologie
Název: **Zpracování a vyhledávání dokumentů s využitím vektorových databází a jazykového modelu**
Kategorie: Data mining
Akademický rok: 2024/25

Zadání:

1. Seznamte se s problematikou zpracování přirozeného jazyka a vyhledávání textových dokumentů. Seznamte se s vektorovými databázemi.
2. Analyzujte požadavky na systém pro vyhledávání dokumentů, který rozdělí vstupní text na menší části vhodné velikosti, doplní k těmto částem dodatečné informace a kontext a upraví je do vektorové podoby. Dále umožní tvorbu dotazu uživatelem, provede vyhledání v databázi a zpracování odpovědi jazykovým modelem.
3. Navrhněte systém dle požadavků. Návrh konzultujte s vedoucím.
4. Implementujte navržený systém.
5. Otestujte vytvořený systém na vhodném vzorku dat a experimentálně ověřte úspěšnost vyhledávání.
6. Zhodnoťte dosažené výsledky a další možnosti pokračování tohoto projektu.

Literatura:

- Vaidyamath, R.C.: Vector Databases Unleashed: Navigating the Future of Data Analytics. Independent, 2024. ISBN 979-8321023488.
- Buttcher, S.: Information Retrieval: Implementing and Evaluating Search Engines. MIT Press Ltd., 2016. ISBN 978-0262528870.

Při obhajobě semestrální části projektu je požadováno:

Body 1-3.

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

Vedoucí práce: **Bartík Vladimír, Ing., Ph.D.**

Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.

Datum zadání: 1.11.2024

Termín pro odevzdání: 14.5.2025

Datum schválení: 22.10.2024

Abstrakt

Tato práce řeší problém vyhledávání informací v rozsáhlých sadách nestrukturovaných textových dokumentů. Navržené řešení kombinuje vektorovou databázi a velké jazykové modely v architektuře Retrieval-Augmented Generation (RAG) integrované s cloudovým úložištěm. Texty jsou z dokumentů extrahovány a děleny na menší části (chunks), které jsou ve vektorové podobě uložené do databáze. Informace se vyhledávají hybridním přístupem kombinujícím vektorové a fulltextové vyhledávání s rerankingem, na jejichž základě je generována odpověď. Výsledkem je systém, který umožňuje uživateli klást dotazy v přirozeném jazyce a získávat relevantní odpovědi podložené znalostní bází tvořenou vlastními dokumenty. Experimentální ověření prokázalo účinnost vyhledávání i kvalitu generovaných odpovědí. Význam práce spočívá v možnosti snadno nasadit vyhledávání nad interními dokumenty organizací bez nutnosti trénování vlastního jazykového modelu.

Abstract

This thesis addresses the problem of information retrieval in large collections of unstructured text documents. The proposed solution combines vector database and a large language models within the Retrieval-Augmented Generation (RAG) architecture, integrated with a cloud storage system. Texts are extracted from documents and divided into smaller parts (chunks), which are stored in the database in a vectorized form. Information is retrieved using a hybrid approach combining vector and full-text search with reranking, based on which an answer is generated. The resulting system enables users to ask questions in natural language and receive relevant answers supported by a knowledge base composed of their own documents. Experimental evaluation confirmed the effectiveness of the retrieval process as well as the quality of the generated responses. The significance of the work lies in the ability to easily deploy information retrieval over internal organizational documents without the need to train a custom language model.

Klíčová slova

vektorové databáze, Retrieval-Augmented Generation, zpracování přirozeného jazyka, embedding modely, vyhledávání informací, velké jazykové modely

Keywords

vector databases, Retrieval-Augmented Generation, natural language processing, embedding models, information retrieval, large language models

Citace

VALÍK, Adam. *Zpracování a vyhledávání dokumentů s využitím vektorových databází a jazykového modelu*. Brno, 2025. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Zpracování a vyhledávání dokumentů s využitím vektorových databází a jazykového modelu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vladimíra Bartíka, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Adam Valík
9. května 2025

Poděkování

Děkuji Ing. Vladimíru Bartíkovi, PhD., za vedení této práce a volnost při řešení. Rád bych dále poděkoval Mgr. Jiřímu Polcarovi, Ph.D. a Ing. Filipu Hadačovi za uvedení do tématu, poskytnutí zadání projektu a cenné odborné rady a konzultace v průběhu zpracování této práce.

Obsah

1	Úvod	2
2	Vektorové databáze	3
2.1	Nestrukturovaná data	3
2.2	Vektorová reprezentace dat	4
2.3	Embedding modely	5
2.4	Systém správy vektorových databází	6
2.5	Indexace a vyhledávání	7
2.6	Využití vektorových databází	10
3	Retrieval-Augmented Generation (RAG)	12
3.1	Limitace velkých jazykových modelů	12
3.2	Architektura RAG	13
3.3	Evaluace RAG systémů	15
3.4	Aplikace	18
4	Architektura systému	19
4.1	Zpracování dokumentů	19
4.2	Vyhledávání dokumentů	21
4.3	Integrace komponent	24
4.4	Využití	25
5	Implementace	26
5.1	Použité technologie	26
5.2	Moduly systému	27
5.3	API	33
5.4	Uživatelské rozhraní	34
6	Testování a evaluace	35
6.1	Dataset	35
6.2	Testování modulu vyhledávání	36
6.3	RAGAs evaluace	38
6.4	Výkonnostní testování systému	42
7	Závěr	44
Literatura		45
A	Plakát Excel@FIT	49

Kapitola 1

Úvod

Vyhledávání informací ve velkých objemech dokumentů naráží v dnešní době nejen na samotné množství dat, ale především na jejich různorodost a absenci jednotné struktury. Interní dokumentace firem, odborné články, právní texty či technické manuály bývají často uloženy v různých formátech jako nestrukturovaná data, což komplikuje jejich uložení do tradičních databází a prohledávání na základě klíčových slov. Vzniká tak potřeba nástrojů, které porozumí významu textů a nabídnou rozhraní pro dotazování nad vlastními dokumenty v přirozeném jazyce s co nejrychlejší odezvou.

K moderním přístupům zpracování přirozeného jazyka patří vektorové databáze, které dokáží pracovat i s nestrukturovanými daty, jako jsou texty, obrázky nebo zvuková data, která reprezentují pomocí vektorů. Svou formou zachycují význam dat, čímž je umožňují sémanticky vyhledávat. Ačkoliv zde techniky a algoritmy sémantického vyhledávání již existují řadu let, řešení ve formě systému správy vektorových databází jsou na vzestupu v posledních dvou letech s vzestupem zájmu o AI.

Spojením vyhledávání ve vektorových databázích a předtrénovaného jazykového modelu lze vytvořit systém, kterému se říká Retrieval-Augmented Generation (RAG), neboli generování rozšířené o získávání informací. RAG kombinuje schopnost jazykového modelu generovat srozumitelné odpovědi v přirozeném jazyce, které zakládá na specifických informacích vyhledávaných ve znalostní bázi.

Tento přístup má široké využití. Umožňuje např. vyhledávání nad velkým množstvím interní dokumentace firem nebo v akademickém prostředí při práci s odbornými texty. Vše závisí na poskytnutí dokumentů pro znalostní bázi a jejich pečlivém zpracování pro uložení do vektorové databáze. To přináší absenci potřeby tréninku vlastního jazykového modelu a usnadňuje nasazení pokročilého nástroje zpracování přirozeného jazyka.

Cílem kapitoly 2 je tak shrnout fungování vektorových databází pro pochopení následné integrace s jazykovým modelem (systém RAG), které je popsáno v kapitole 3. Výsledkem této práce je pak vytvoření komplexního systému, který nabídne uživatelům získávat informace z velkého množství vlastních dokumentů. Je toho dosaženo kombinací moderních přístupů zpracování přirozeného jazyka, jako jsou práce s nestrukturovanými daty, embedding modely a vektorové databáze s jejich indexací pro efektivní hybridní vyhledávání k získávání informací. Promptování a generativní schopnosti jazykového modelu nakonec slouží pro navrácení lidsky formulované odpovědi podložené informacemi z báze znalostí. Architektura tohoto systému se nachází v kapitole 4 a jeho implementační detaily v kapitole 5. Na závěr, v kapitole 6, jsou experimentálně doladěny parametry systému a ověřena jeho úspěšnost, škálovatelnost a efektivita.

Kapitola 2

Vektorové databáze

Již v roce 1998 se odhadovalo, že více než 85% veškerých dat neexistuje ve strukturované formě, a tedy je nelze zpracovávat pomocí relačních databází [4]. Před zhruba deseti lety, s vývojem zpracování přirozeného jazyka a hlubokého učení, začaly vznikat trénované modely se schopností reprezentovat nestrukturovaná data jako vektory, které zachovávají význam dat a jejich vlastnosti. V důsledku tomu začal během posledních let vývoj tvorby nového typu databáze jako řešení pro zpracování a vyhledávání nestrukturovaných dat.

„Vektorová databáze je typ databáze, která ukládá data jako vysokorozměrné vektory, což jsou matematické reprezentace rysů nebo atributů.“ [15]. Tyto databáze byly navrženy právě za účelem efektivního ukládání a vyhledávání vektorů. Jejich síla pak spočívá v možnosti provádět vyhledávání na základě sémantické podobnosti, což je zásadní rozdíl oproti tradičním relačním databázím, které operují nad přesnými hodnotami ve sloupcích.

Vektorové databáze se tak stávají klíčovým nástrojem v moderních aplikacích, jako jsou doporučovací systémy, chatboti či systémy sémantického vyhledávání. V následujících podkapitolách je detailněji popsáno, jak vektorové databáze zpracovávají nestrukturovaná data, jak funguje vektorová reprezentace dat a jakým způsobem se provádí indexace pro rychlé vyhledávání.

2.1 Nestrukturovaná data

Lidé i stroje produkují velké množství dat, které nesplňují předem daný formát nebo schéma. Taková data jsou pak náročná na vyhledávání a vyžadují předzpracování a analýzu. Nestrukturovaná data nezapadají do relačních databázových systémů, jelikož je nelze reprezentovat hodnotami uloženými do sloupců tabulky jako data strukturovaná. Spadají tak do datového typu BLOB (Binary Large Object), přes který nelze snadno vyhledávat.

Příkladem nestrukturovaných dat jsou [43]:

- **data vytvořená člověkem:** e-maily, textové dokumenty, poznámky, obrázky, audio nebo video nahrávky,
- **data vytvořená strojem:** údaje ze senzorů, data počítačového vidění, webová a aplikativní data, logy, data z IoT zařízení.

Mezi strukturovanými a nestrukturovanými daty existuje mezíkrok, který se nazývá poulostrukturovaná data. Vznikají přidáním metadat nebo významových tagů, čímž je dodána nestrukturovaným datům jistá forma organizace nebo hierarchie [7]. Například k obrázkům,

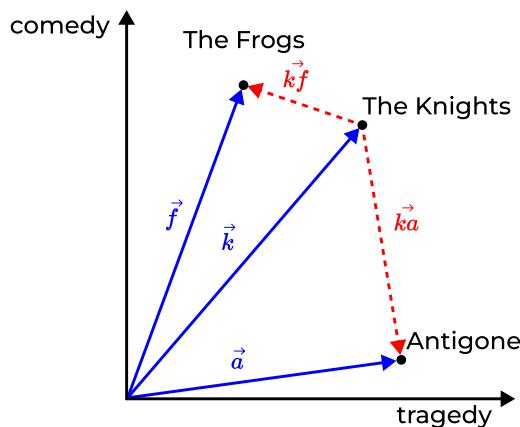
které jsou binárním popisem vlastností jednotlivých pixelů, lze přidat údaje o autorovi, datu a místu vzniku, což umožňuje snadnější filtrování a organizaci dat.

Nestrukturovaná data jsou nejen ukládána, ale procesem vektorizace je jim přidělena sémantika [33]. Proto pro efektivní uložení a vyhledávání nestrukturovaných dat hrají klíčovou roli systémy pro správu vektorových databází.

2.2 Vektorová reprezentace dat

Vektor je v matematice formálně definován jako prvek vektorového prostoru. Pakliže má tento prostor konečné rozměry, lze hovořit o usporádané n -tici čísel, kde n udává dimenzi vektoru. S vektory lze provádět operace vektorové aritmetiky, kde nejdůležitější roli pro vektorové databáze hrají podobnostní funkce, jako kosinová podobnost, Euklidovská vzdálenost nebo skalární součin [38].

V kontextu vektorových databází jsou vektory tvořeny transformací dat na matematickou reprezentaci pomocí modelů strojového učení, které se nazývají embedding modely [15]. Vektory si tedy lze představit jako soubor čísel, kde každé z nich reprezentuje číselné vyjádření určité vlastnosti dat. Např. na obrázku 2.2 je zobrazen 2D prostor. Pakliže jednotlivým složkám dvourozměrného vektoru přidělíme nějaký rys, můžeme ohodnotit např. dramata označující řecké hry na základě toho, jak komické a tragické jsou, a uložit je do tohoto vektorového prostoru.

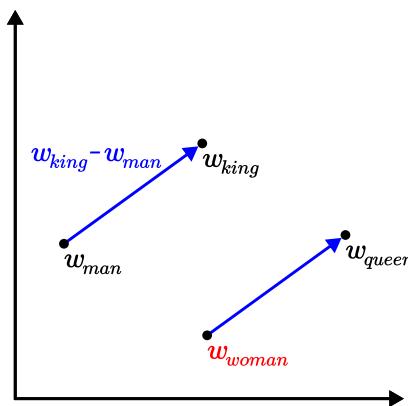


Obrázek 2.1: Příklad vektorového prostoru o dvou dimenzích. Podobnost vlastností dat lze vypočítat vzájemnou vzdáleností mezi vektory, a tak významově podobná data jsou blízko sebe. Tento obrázek byl převzat z [33].

Komplexní data však vyžadují mnohem více popisů atributů, a tak podobně jako nestrukturovaná data, vysoko dimenzionální vektory nejsou pro lidi čitelné a uchopitelné. Pro práci s vektory není nutné užití specializovaných vektorových databází, avšak jak počet vektorů a jejich dimenze roste, vektorový databázový systém poskytuje efektivní řešení pro uložení a vyhledávání nad těmito komplexními daty [33]. Počet dimenzí vektorů se snadno pohybuje od stovek po tisíce s tím, že s vývojem poroste na desítky tisíc. Toto číslo se odvíjí od architektury použitého embedding modelu.

2.3 Embedding modely

Proces vektorizace lze označit jako „transformaci dat na vysoce dimenzionální vektorovou reprezentaci, která zachycuje smysluplné vztahy a vzorce“ [33]. Cílem embedding modelů je zachytit význam dat v numerické podobě, aby byly použitelné pro výpočetní modely. Výsledné vektory, též nazývané *embeddingy*, umožňují strojům práci s nestrukturovanými daty, jelikož jejich poloha v prostoru odráží vzájemné vztahy s ostatními vektory. Významově podobná vstupní data tak budou blízko sebe. Demonstraci vektorové aritmetiky a vzájemného vztahu mezi daty uloženými ve vektorovém prostoru lze ilustrovat na klasickém příkladu [2]: Mějme vektorový prostor obsahující vektorové reprezentace slov ze slovníku. Mějme vektorovou reprezentaci slova *král*. Odečteme-li od slova *král* slovo *muž* a následně přičteme slovo *žena*, nejbližší vektor tomuto vypočtenému vektoru bude ten, který reprezentuje slovo *královna*.



Obrázek 2.2: Příklad vektorové aritmetiky mezi embeddingy ve vektorovém prostoru.
Zde $w_{queen} \approx w_{king} - w_{man} + w_{woman}$.

Embedding modely jsou jedním z produktů strojového učení. Obecně vznikají trénováním na rozsáhlých sadách nestrukturovaných dat, např. textových korpusech, obrázcích nebo zvukových souborech. Tato práce se specificky zaměřuje na zpracování textových dokumentů, kde embedding modely patří k významnému nástroji zpracování přirozeného jazyka.

2.3.1 Word Embeddings

Prvním průlomem ve vývoji embedding modelů byly *word embeddings*, které vektorizují jednotlivá slova. Pro proces konverze se používají nástroje jako Word2vec, který využívá neuronové sítě k naučení asociace slova z velkého textového korpusu. Word2vec byl navržen jako 2 modely [6], kde:

- **CBOW** (Continuous Bag-Of-Words) se zaměřuje na předpovídání aktuálního slova na základě okolního kontextu, zatímco
- **Skip-gram** se na rozdíl od CBOW zaměřuje na předpovídání kontextu pro dané slovo.

Další významnou architekturou je GloVe (Global Vectors) [30].

Cíl reprezentovat každé slovo bodem ve vektorovém prostoru má hlavní nedostatek, jelikož ignoruje možnosti slova, které má v různých kontextech různý význam. Problémem

word embeddings tak je, že vícevýznamová slova jsou přeložena v každém svém významu ve větě na identickou vektorovou reprezentaci. Zachycení správné sémantiky vícevýznamových slov však hraje klíčovou roli v porozumění jazyku NLP systémů. Může se tak stát, že slova *krysa* a *počítac* budou „blízko sebe“, ačkoliv spolu sémanticky nesouvisí, jelikož se budou obě významově blížit slovu *myš* [6].

2.3.2 Kontextové modely

Moderní kontextové embedding modely, jako jsou BERT nebo GPT, patří mezi velké jazykové modely (*Large Language Model*, zkráceně LLM). Mimo summarizaci a prediktivní generování textu dokáží tyto LLM tvořit vektorové reprezentace textu. Umí zachytit dynamiku vícevýznamových slov tak, že pro různé významy slov generují různé vektorové reprezentace. Zohledňují tak kontext konkrétní věty na základě okolních slov.

GPT (Generative Pre-Trained Transformer)

Příkladem kontextového modelu je GPT, funguje na architektuře transformerů a byl navržen pro generování textu. Predikuje další slovo ve větě na základě předchozích slov a kontext slova dokáže zachytit tím, že čte větu jednosměrně zleva doprava. Mimo generování textu však dokáže díky porozumění kontextu vytvářet vektorové reprezentace textu [39].

BERT (Bidirectional Encoder Representations from Transformers)

BERT je předtrénovaný model (převážně na celé anglické Wikipedii), který používá transformery k zachycení kontextu slova ve větě, kterou navíc čte oběma směry, zleva doprava i zprava doleva. Tím dokáže lépe zachytit kontext daného slova a vytvořit pro něj vektorovou reprezentaci [9].

S-BERT (Sentence-BERT)

S-BERT je rozšíření modelu BERT. Oproti předchozím modelům je S-BERT efektivnějším nástrojem pro zpracování textových dokumentů, jelikož optimalizuje tvorbu vektorové reprezentace vět (*sentence embeddings*). Přidává speciální tréninkovou fázi, aby mohl efektivně vytvářet vektory z celých kusů textu [31]. Toho lze využít ve vektorových databázích při uložení textových dokumentů a následném vyhledávání pomocí dotazu vektorizovaném tímto modelem.

2.4 Systém správy vektorových databází

Systém správy vektorových databází je technologie, která práci s vektorovými databázemi seskupuje dohromady. Je to specializovaný typ systému správy databází zaměřující se na efektivní správu vysokorozměrných vektorových dat. Toto nezahrnuje nízkorozměrná vektorová data (jako například 2D souřadnice), s nimiž je práce podstatně jednodušší a nevyžaduje dále uvedené optimalizační techniky. VDBMS (z anglického *Vector Database Management System*) většinou podporuje vyhledávání na základě podobnosti prostřednictvím indexace, která je nevyhnutelná pro rychlé vyhledávání ve vektorovém prostoru [33].

Termín vektorová databáze se často používá jako synonymum pro VDBMS. Sama vektorová databáze je však jen soubor dat, zatímco systém správy vektorových databází je celý software. Zajišťuje nejen uložení dat, ale i efektivní vyhledávání relevantních informací a

často si ukládá k samotnému vektoru i nějaký identifikátor nebo metadata, která mohou sloužit k filtrování nebo k poskytování personalizovaných doporučení [33].

Mezi známé systémy správy vektorových databází patří [28]:

- **Weaviate** – open-source databáze s podporou hybridního vyhledávání, která se zaměřuje na vyhledávání dokumentů v grafovém modelu,
- **Pinecone** – proprietární řešení, které je zaměřené na škálování, nabízen jako cloudový systém,
- **Milvus** – systém podporující vícenásobné indexovací metody a distribuci výpočtů,
- **Faiss** – knihovna společnosti Meta s algoritmy pro indexaci a vyhledávání s vektory,
- **pgvector** – rozšíření PostgreSQL, které přidává vektor a indexační metody pro vektorové vyhledávání.

2.5 Indexace a vyhledávání

Po vektorizaci a uložení dat je nad nimi potřeba vytvořit index. Indexace slouží k organizaci dat pro rychlejší vyhledávání. Vektorové indexy zrychlují proces vyhledávání minimalizací porovnávání vektorů rozdělením vektorového prostoru na datové struktury, které lze snadno procházet. Porovnává se tak pouze malá podmnožina prostoru k nalezení nejbližších sousedů [28].

Výběr algoritmu k indexaci záleží na požadavcích. Obecně bývá výpočetně náročný, v ideálním případě ho však není třeba při rozšíření datasetu přepočítávat a umožnuje velmi rychlé vyhledávání. Právě proto se bez něho VDBMS neobejdou [33].

Před vyhledáváním je systému podán dotaz, který je často ve formě přirozeného jazyka, a proto potřebuje být taktéž vektorizován, a to stejným embedding modelem jako vektory uložené v databázi. Dotazy si obvykle uchovávají metadata k vyhledávání, jako např. počet nejbližších vektorů k nalezení [33]. Vektor dotazu je tak „zanesen“ mezi vektory databáze a probíhá vyhledávání. Jak již bylo zmíněno, vyhledávání ve vektorových databázích probíhá na základě významové podobnosti, jinak řečeno vzájemné vzdálenosti mezi vektory, jelikož sémanticky blízké vektory jsou i výpočetně blízko sebe. Proto se tomuto principu říká vyhledání nejbližších sousedů (anglicky *Nearest Neighbor Search*, NNS). Je to „optimizační problém nalezení bodu v dané množině, který je nejbliže (= nejpodobnější) danému bodu“ [15]. Motivace k zavedení indexů a approximace výsledků je znázorněna na následující metodě.

2.5.1 Naivní metoda

Nejjednodušším přístupem k nalezení nejbližšího vektoru je naivní metoda (v anglické literatuře se též používá pojem *brute-force approach* [15], neboli přístup hrubou silou). Naivní algoritmus porovnává každý vektor dotazu se všemi vektory v databázi na základě zvolené podobnostní metriky. Tento přístup má absolutní úspěšnost pro navrácení požadovaného počtu opravdových nejbližších bodů, avšak je velice neefektivní. Složitost tohoto přístupu je $O(NM)$, kde N je počet vektorů v databázi a M je jejich dimenze (velikost embeddingu). Při menším počtu nízkodimensionálních vektorů je tato intuitivní metoda efektivní díky své přesnosti. U vektorových databází však počet dimenzií vektorů roste exponenciálně a dochází k poklesu efektivity mnoha algoritmů strojového učení, což popisuje pojmem prokletí

dimenzionality (z anglického *curse of dimensionality*) [29]. Počet vektorů uložených v databázích velkých projektů navíc může dosahovat několika miliard, a proto VDBMS zavádějí approximaci a indexační metody, které budou popsány v následujících podkapitolách.

2.5.2 Přibližné vyhledávání nejbližšího souseda

Zatímco metody NNS prohledávají vektorový prostor vyčerpávajícím způsobem porovnáváním vektoru se všemi ostatními, metody ANNS (z anglického *Approximate Nearest Neighbor Search*) řeší problém jejich neefektivity na velkém množství vysokodimenziálních vektorů. Na úkor přesnosti vyhledávají nejbližšího souseda pouze přibližně s určitou přesností, poskytují však velmi výrazný nárůst výkonu umožňující provádění vyhledávání v rádech tisíců na sekundu. Je třeba u nich optimalizovat volbu parametrů dle specifikací databáze a vyvážit tak poměr mezi výkonem a přesností. Při hledání k nejbližších sousedů se metody označují jako k -ANNS, pro jednoduchost bude toto označení vynecháno (jde o nastavení parametru, princip zůstává stejný). Výběrem nejpoužívanějších implementací ANNS jsou následující metody a jejich algoritmy.

2.5.3 Hashovací metody

Při zpracování vysoce dimenzionálních dat staví tyto techniky na transformaci vektorů pomocí hashovacích funkcí na kompaktnější reprezentaci ve formě hashovacích klíčů. Redukuje složitost porovnávání dat a urychlují tak vyhledávání i délky specifické sadě hashovacích funkcí, které zajistují, že není třeba prohledávat celou databázi. Příkladem je následující metoda, která při hashování zachovává podobnost mezi daty.

Locality Sensitive Hashing (LSH, v překladu „hashování citlivé na lokalitu“) je algoritmus urychlující přibližné vyhledávání sousedů v rozsáhlých databázích. Jeho principem je použití sady hashovacích funkcí, které jsou citlivé na lokalitu a zachovávají tak informace o podobnosti mezi vektory. To znamená, že blízké vektory (podle zvolené metriky) hashuje do stejného „hashovacího koše“ s vyšší pravděpodobností než vektory vzdálené. LSH pro každý vektor aplikuje sadu hashovacích funkcí, které přidělí vektorům hashovací klíče odpovídající jejich lokalitě. Dle téhoto klíčů lze určit, do kterého hashovacího koše daný vektor spadá. Při vyhledávání je vektor dotazu zpracován stejným principem, takže lze snadno nalézt kandidáty na nejbližší sousedy ve stejném hashovacím koši. Jelikož není potřeba prohledat celou databázi, dochází k výraznému urychlení vyhledávání [41].

2.5.4 Stromové metody

Smyslem stromových metod je zmenšit prohledávaný prostor následováním větví stromu, které s největší pravděpodobností obsahují nejbližší sousedy vektorizovaného dotazu [15].

Approximate Nearest Neighbors Oh Yeah (ANNOY) je algoritmus vyvinutý společností Spotify, původně pro doporučování hudby. Pracuje nejfektivněji ve střednědimenzionálních vektorových prostorech (nižší stovky dimenzí), zatímco při velmi vysoké dimenzionalitě může efektivita klesat. ANNOY funguje na principu náhodných projekcí a využívá stromové struktury k rozdělení datového prostoru. V každém uzlu stromu je prostor rozdělen na podprostory pomocí náhodně zvolené hyperroviny¹. Hyperrovnina je zvolena na základě

¹hyperrovnina je rovina, která dělí vícedimenzionální prostor na dvě části

dvou náhodně vybraných bodů, mezi kterými prochází středová rovina, která je od obou stejně vzdálená. Tento proces pokračuje rekurzivně, dokud podprostor neobsahuje dostačně málo bodů (v závislosti na parametru). Výsledkem je les binárních stromů, kde každý vektor je přiřazen k listovému uzlu na základě své polohy vůči hyperrovinám.

Při vyhledávání nejbližších sousedů prochází ANNOY každý strom od kořene k listovému uzlu, kam připadá vektor dotazu a shromažďuje všechny vektory ve stejných listových uzlech. Poté vypočítá přesnou vzdálenost mezi dotazovaným vektorem a téměř kandidáty a vrátí nejbližší sousedy. Počet stromů a hloubka stromu lze optimalizovat ke kompromisu rychlosti a přesnosti [3, 15].

2.5.5 Grafové metody

Grafové metody poskytují řešení ukládání a vyhledávání vektorů pomocí grafových struktur. Při vytváření indexu přidávají bod po bodu, přičemž je podle algoritmu spojují do grafu pro následný jednoduchý průchod grafem k přibližnému nalezení nejbližších sousedů.

Navigable small world (NSW, v překladu „Prohledávatelný malý svět“) je algoritmus tvořící graf spojováním vektorů s jeho nejbližšími sousedy. NSW graf je zkonstruován přidáváním vektorů do prostoru v náhodném pořadí. Každý přidaný vektor spojí hranou s určitým počtem nejbližších vektorů (heuristikou nejlepší možnosti je vypočtená vzdálenost mezi vektory), přičemž prohledávání začíná na několika náhodných vstupních bodech a funguje hladově, tedy může skončit v lokálním minimu. Výsledný graf se pak blíží konceptu *malého světa*², jelikož body přidány v prvotní fázi vytvoří vzdálená spojení mezi výslednými shluky. Výsledkem je až logaritmické prohledávání grafu k získání přibližných nejbližších sousedů [21].

Hierarchical navigable small world (HNSW, v překladu „Hierarchický prohledávatelný malý svět“) je vylepšením algoritmu NSW o hierarchickou strukturu a představuje jednu z nejvýkonnějších metod ANNS. Vytváří vícevrstvý graf. Body jsou přidávány od nejvyšší vrstvy, kde je jich nejméně. Jakmile nalezne lokální minimum, pokračuje na nižší úrovni s daným bodem jako výchozím. Proces se opakuje s tím, že každá vyšší úroveň obsahuje řádově menší počet uzlů než vrstva nižší. To umožňuje přesvědčivě dosáhnout logaritmické náročnosti, jelikož vyhledávání zprvu udělá „velké kroky“, a jakmile se dostane na nižší úrovni, blíží se relevantnějším uzlům představující nejbližší sousedy. V realitě pak nejvyšší vrstvu mohou představovat stovky uzlů z celkového počtu milionů [21].

2.5.6 Kvantizační metody

Kvantizační metody jsou techniky používané ke snížení výpočetní složitosti při práci s velkými dimenzionálními daty. Kvantizace signálů převádí spojité hodnoty na diskrétní, čímž umožňuje efektivnější ukládání. Kvantizace vektorů pak approximuje jejich hodnoty na reprezentativní vzorky, čímž dochází k výrazné kompresi dat. Zároveň z této approximace vzniká kvantizační chyba, která je však přijatelným kompromisem přinášejícím nárůst výkonu a snížení paměťových nároků. [17].

²malý svět a šest stupňů odloučení ve společnosti je myšlenka, že jsou všichni lidé v průměru 6 sociálních kontaktů od sebe

Product Quantization (PQ, v překladu „Kvantizace produktu“) je další z technik efektivního vyhledávání nejbližších sousedů vysokodimenzionálních prostorů. Hlavní myšlenkou je rozdělit vektorový prostor na menší podprostory a každý z nich kvantizovat samostatně. Vektory se tedy rovnoměrně rozdělí na daný počet částí (podprostory). Pro každý podprostor jsou pomocí shlukovacího algoritmu (např. *k-means*) určeny centroidy a zapsány do tzv. kódovací knihy. Vektor, který je následně přidáván do indexu, je rozdělen na podprostory a každý tento podprostor je přiřazen nejbližšímu centroidu. Výsledný záznam v databázi se skládá z identifikátorů centroidů v kódovací knize každého z podprostorů [17].

Při vyhledávání je vektor dotazu rozdělen na podprostory a kvantizován stejným způsobem jako vektory uložené v databázi. Na základě identifikátorů centroidů z kódovacích knih jsou vybráni kandidáti, jejichž indexy odpovídají nebo se nejvíce podobají dotazu. Pro tyto kandidáty lze původní vektory rekonstruovat a následně vypočítat přesné vzdálenosti k dotazu, což umožňuje nalezení nejbližších sousedů [17].

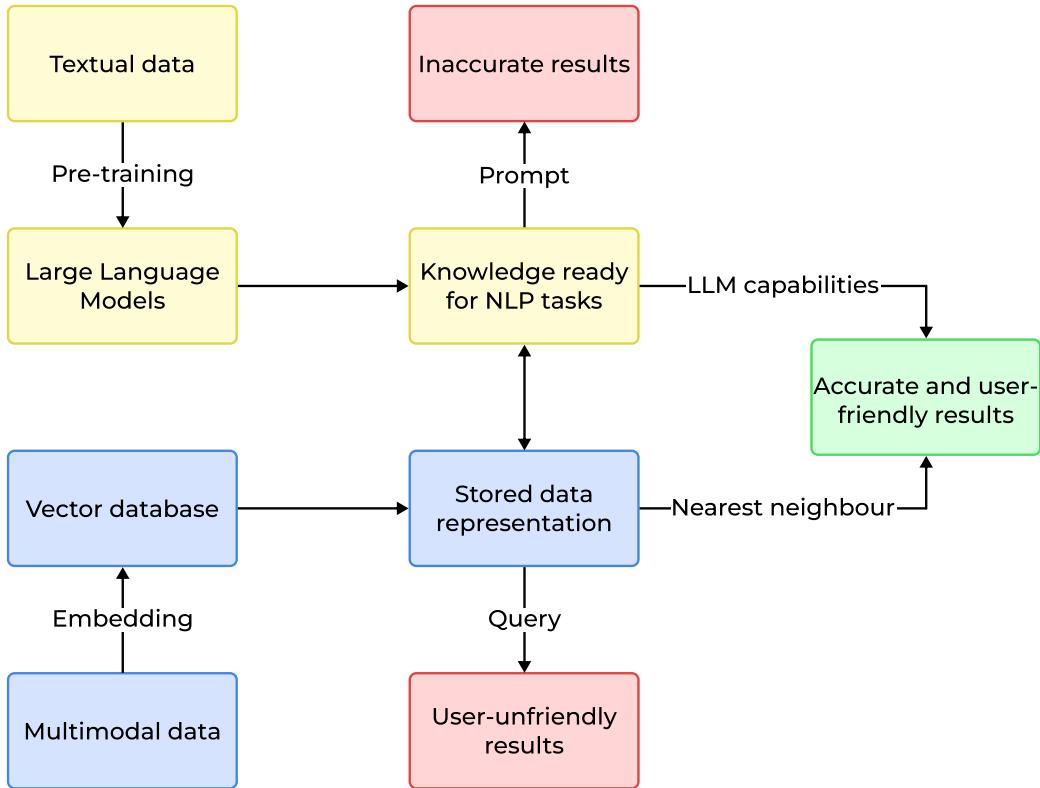
2.6 Využití vektorových databází

Vektorové databáze jsou moderním nástrojem pro vyhledávání dat na základě významové podobnosti. Kterakoliv data zakódovaná embedding modely do vektorů lze pak sémanticky vyhledávat, není třeba se tak omezovat pouze na slova či texty. Ačkoliv je to komplexnější proces, obrázky nebo videa lze také vektorizovat a vyhledávat tak jim podobné. U obrázků se jedná o extrakci vlastností jednotlivých pixelů, která se typicky děje pomocí konvolučních neuronových sítí. Jsou tak zachyceny různé vlastnosti, které jsou převedy do vektorové reprezentace. Podobně může probíhat systém s videi, kde je například zachyceno několik klíčových snímků. Po vyhledání nejbližších sousedů ve vektorové databázi je tak možné získat daná data [33].

Tato schopnost je typicky využívána v doporučovacích systémech, kde na základě získaných dat o uživatelech jsou systémy schopny nabízet podobný typ produktů nebo obsahu. Pro tuto práci je však nejrelevantnější využití vektorových databází ve spojení s velkým jazykovým modelem.

2.6.1 Vektorové databáze a jazykové modely

Databáze a velké jazykové modely (LLM) se nachází každý v jiné oblasti výzkumu. Specificky vektorové databáze však svými schopnostmi přináší zajímavé využití v kombinaci s jazykovými modely, které ilustruje obr. 2.3. LLM jsou předtrénované na velkém množství parametrů (v dnešní době se jedná o řády miliard až bilionů), tedy textových dat. Ty poskytují paměť pro jejich prediktivní a generativní schopnosti a jsou významným nástrojem pro úlohy zpracování přirozeného jazyka. Mohou však produkovat nepřesné výsledky z důvodu absence aktuálních informací v době tréninku. Na druhé straně jsou vektorové databáze, které zvládají efektivně ukládat multimodální data (text, obrázky, videa nebo zvuk) pomocí embeddingů. Tato data jsou pak rychle vyhledávána, sama o sobě však nemusí být uživatelsky přívětivá. Kombinací schopností LLM s vyhledáním nejbližších sousedů tak vznikají fakticky přesné a uživatelsky přívětivé výsledky [28].



Obrázek 2.3: Kombinace vektorových databází s velkým jazykovým modelem vzájemně vyvažuje nedostatky jednotlivých přístupů k vyhledávání informací. Převzato z [15].

Vektorové databáze jsou tak schopny vyhledávat dokumenty relevantní k uživatelskému dotazu a dodávat kontext pro dotazy na generativní modely. Tomuto systému se říká *Retrieval-Augmented Generation* a zaměřuje se na něj zbytek této práce.

Kapitola 3

Retrieval-Augmented Generation (RAG)

V publikaci z roku 2021 přišli Lewis et. al. [19] se systémem alternativním k čistě parametřovanému LLM, který by zvýšil přesnost dodávání faktů vyhledáváním informací v externím zdroji. Generování rozšířené o vyhledávání (*Retrieval-Augmented Generation*, zkráceně RAG) je systém, kde „předtrénovaný parametřovaný generativní model je vybavený neparametrickou pamětí prostřednictvím univerzálního přístupu jemného doladění“ [19].

3.1 Limitace velkých jazykových modelů

RAG kombinuje vyhledávací mechanismy s generativními jazykovými modely pro zvýšení přesnosti výstupů založených na konkrétních faktech. Cílí tak na některé klíčové problémy velkých jazykových modelů.

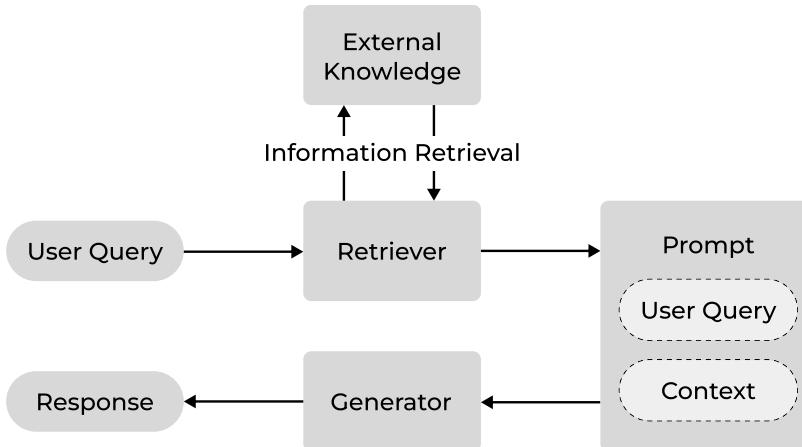
Většina jich plyne z faktu, že LLM uchovávají svoji paměť v parametrech, na kterých byly natrénovány, tedy v *parametřované paměti*. Jelikož jsou trénovány na velice rozsáhlém množství dat, dokáží pochytit velké množství znalostí bez využití jakékoliv externí paměti. Jsou též skvělé v generování lidsky vypadajícího textu. Fungují však na základě pravděpodobnosti slovních sekvencí, nikoliv ověřených faktů. Jsou též omezené na data, která byla přítomna ve fázi tréninku [19, 14]. To vede k určitým limitům:

- **Halucinace:** LLM mohou produkovat odpověď, která sice působí věrohodně, ale je fakticky nesprávná. Může se tak jednat o smyšlené, neexistující informace.
- **Časově omezená znalost:** LLM mají svůj *cut-off date*, tedy datum, kde končí jejich znalosti, typicky datum před trénováním. To vede k tomu, že nemají přístup k aktuálním informacím a jejich znalosti nelze snadno doplnit.
- **Znalosti specifických domén:** LLM jsou typicky obecné modely, a tak jim mohou scházet informace ke specifickým oblastem.
- **Citace:** LLM poskytují své odpovědi bez uvedení zdrojů, to snižuje důvěryhodnost a možnosti použití.
- **Soukromá data:** LLM jako obecné modely jsou trénované na veřejně dostupných datech a nemají přístup k soukromým či proprietárním datům.

RAG se snaží adresovat tyto problémy přidáním vyhledávacího modulu, který staví na neparametrické paměti [19]. Ta je schopná snadno aktualizovat bázi znalostí, vyhledávat je v reálném čase a poskytovat tak relevantní kontext jako vstup jazykovému modelu společně s uživatelským dotazem. Odpověď je pak srozumitelná, uživatelsky přívětivá a podložena konkrétními dokumenty, které dané fakty uvádějí. Pakliže kontext není dostatečný, je žádoucí, aby toto bylo řečeno v odpovědi, a je tak vynucena absence smyšlených informací. Báze znalostí pak staví na poskytnutých informacích, které mohou pokrývat specifickou doménu nebo privátní data.

3.2 Architektura RAG

Vstupem systému je uživatelův dotaz. Dále RAG kombinuje dvě klíčové komponenty. První je *retriever*, mechanismus vyhledávání informací v externím úložišti znalostí. Na základě dotazu vyhledává nejrelevantnější informace, které potenciálně obsahují odpověď. Sada vrácených informací z dokumentů pak tvoří kontext. Ten je společně s uživatelským dotazem formulován jako vstup *generátoru*, který poskytuje uživateli odpověď.



Obrázek 3.1: Diagram komponent architektury RAG. Odpověď na dotaz je generována na základě vyhledaných informací v externích znalostech.

3.2.1 Retriever

Retriever je klíčovým modulem systému, který slouží jako nadstavba generátoru a tvoří základ principu Retrieval-Augmented Generation. Jeho hlavním cílem je identifikace relevantních dokumentů nebo jejich částí, které následně slouží jako kontext pro generativní model. Aby systém fungoval v reálném čase, musí retriever poskytovat výsledky s nízkou latencí, což vyžaduje efektivní indexaci dat (viz kap. 2.5) v předzpracovací fázi, tedy vhodnou organizaci dokumentů. Optimalizované algoritmy indexace pak umožňují rychlé vyhledávání i nad rozsáhlými datovými sadami.

Vyhledávání v retrieveru je typicky realizováno pomocí řídkých (*sparse*) nebo hustých (*dense*) vektorových reprezentací.

Řídké vektory mají vysokou dimenzionalitu a obsahují převážně nulové složky, přičemž nenulové hodnoty obvykle reprezentují váhu nebo frekvenci výskytu termínů v dokumentu. Jsou založeny na metodách tradičního vyhledávání informací pomocí klíčových slov, kde podobnost závisí na přesné shodě termínů. Nejpoužívanějšími metodami jsou:

- **TF-IDF** (*Term Frequency–Inverse Document Frequency*), která vyjadřuje důležitost slova na základě jeho četnosti výskytu v dokumentu [5],
- **BM25** (*Best Match 25*), pravděpodobnostní model relevance, který kombinuje frekvenci výskytu slov a normalizaci délky dokumentu k odhadu skóre relevance mezi dotazem a dokumenty. Jedná se o jeden z nejrozšířenějších algoritmů pro fulltextové vyhledávání [32].

Husté vektory naproti tomu většinou nulové složky neobsahují a využívají neuronové sítě k mapování dotazů a dokumentů do vektorového prostoru s nižší dimenzionalitou. Takové vektory pak dokáží zachytit i sémantické podobnosti (pomocí nejbližších sousedů) namísto přesné shody termínů, dosahují tak mnohem vyšší výkonnosti než fulltextové metody, jsou však výpočetně náročnější a vyžadují embedding modely. Modelem této kategorie pak je **DPR** (*Dense Passage Retriever*), který používá dva nezávislé transformátorové enkódery BERT pro mapování dotazu a dokumentu po částech do hustého vektorového prostoru, kde probíhá sémantické vyhledávání porovnáváním dotazu s pasážemi pomocí skalárního součinu [18].

3.2.2 Generátor

Generativní modul na základě kontextu z navrácených informací a vstupního dotazu produkuje odpovědi ve formě přirozeného jazyka. K tomuto úkolu lze využít kterýkoliv model architektury enkodér-dekodér používané v *seq2seq* transformerových modelech [23]. Tento modul je však obvykle implementován pomocí velkého jazykového modelu (LLM). Nejčastěji se využívají transformerové modely jako GPT [39], které dosahují nejlepších výsledků v generování přirozeného jazyka.

3.2.3 Vylepšení

Zhao P. et al. ve své práci [42] navrhují metody k optimalizaci výsledků pro jednotlivé části RAG, které mohou vést k lepší přesnosti vyhledávání a zlepšení kvality generovaných výsledků. Mezi ty nejpodstatnější v jednotlivých oblastech patří:

1. **Vstup:** vstupem je uživatelský dotaz a ten ovlivňuje kvalitu výsledku vyhledávání.
 - **Transformace dotazu** – transformace dotazu do optimalizované podoby pro vyhledávání (např. dekompozice komplexního dotazu do více poddotazů).
 - **Rozšíření dat** – odstranění nerelevantních informací, dynamická aktualizace dokumentů a přidávání nových.
2. **Retriever:** kvalita odpovědi v RAG systémech závisí na relevanci vyhledaných informací, které jsou následně podány generátoru, ale také na kvalitě externích dat.
 - **Rekurzivní vyhledávání** – vícenásobné vyhledávání pro zpřesnění nalezeného kontextu.

- **Optimalizace chunkování** – optimalizace velikosti chunků (menších částí dokumentu) pro efektivnější vyhledávání.
 - **Doladění retrieveru** – retriever spoléhá na embedding model, který vytváří vektorové reprezentace dat. Kvalitu embedding modelu lze doladit pro specifické oblasti informací ke zlepšení sémantického vyhledávání.
 - **Hybridní vyhledávání** – použití různých metod vyhledávání (např. kombinace vektorového a fulltextového vyhledávání) nebo získávání informací z více různých zdrojů.
 - **Re-ranking** – změna pořadí nalezených dokumentů na základě přezkoumání relevance.
 - **Filtrování metadat** – filtrování dokumentů na základě metadat.
3. **Generátor:** kvalita výstupní odpovědi závisí na generátoru při zpracování vstupního kontextu s dotazem.
- **Prompt engineering** – optimalizace vstupních instrukcí a integrace s navrácenými informacemi.
 - **Doladění generátoru** – ladění parametrů generativního modelu pro zvýšení kvality odpovědí.
4. **Výstup:** zlepšení celého procesu RAG produkovajícího výsledek pak spočívá v identifikaci případů, zdali se vyhledáním skutečně dojde k lepšímu výsledku. Může se stát, že navrácený kontext je pro dotaz nedostatečný. V takových případech je třeba předat zodpovědnost zpět generativnímu modelu, nebo ho v rámci prevence halucinací instruovat, aby neodpovídal bez informací, které by odpověď podložily.

3.3 Evaluace RAG systémů

Hodnocení kvality RAG systémů není jednoduchý proces kvůli jejich vícevrstvé architektuře a závislosti na externích zdrojích informací. Další komplikací při evaluaci RAG systémů je skutečnost, že generativní model (LLM) zde funguje jako *black-box*. Výsledky, jež produkuje, nemusí být plně transparentní a deterministické. Vyhledávání navíc přímo ovlivňuje kvalitu odpovědí generativního modulu, proto je potřeba testovat jak samostatné moduly, tak RAG jako celek. Pro praktické využití RAG systémů tyto skutečnosti pozdvihují důležitost zavedení automatických vyhodnocovacích postupů, metrik a testovacích sad, které adresují reálné případy použití [40].

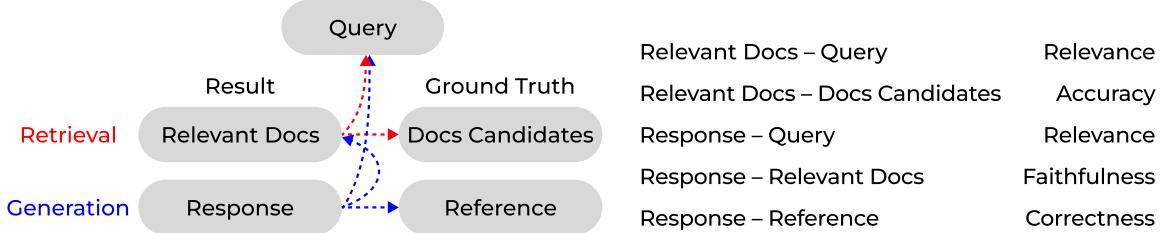
3.3.1 Jednotný proces hodnocení

V průzkumu nástrojů a benchmarků na evaluaci RAG systémů navrhli Yu et. al. [40] jednotný proces hodnocení Auepora (A Unified Evaluation Process of RAG), který pomáhá pochopit složitost srovnávacího hodnocení RAG a vychází z něj tato podkapitola. Auepora je zaměřena na zodpovězení tří klíčových otázek:

1. Co hodnotit?
2. Jak to hodnotit?
3. Jak to měřit?

Cíl hodnocení

Základ tvoří 5 cílů hodnocení, které testují jednotlivé vlastnosti RAG systémů. Definují je kombinace výsledků a referenčních základních pravd (tzv. *ground truths*), které jsou předem dané. Určují tak požadavky na RAG systémy.



Obrázek 3.2: Diagram možností, které vytváří cíle hodnocení. Převzato z [40].

1. Vyhledávání

- **Relevance** – určuje, jak dobře vyhledané dokumenty odpovídají vstupnímu dotazu. Cílem je zjistit, zdali systém vyhledává dokumenty obsahující potřebné informace.
- **Přesnost** – porovnává vyhledané dokumenty se sbírkou určených kandidátních dokumentů a tím měří schopnost systému identifikovat relevantní dokumenty včetně jejich prioritizace před méně relevantními.

2. Generování

- **Relevance** – určuje míru relevance výstupní odpovědi k dotazu, tedy nakolik je vygenerovaná odpověď v souladu se záměrem a obsahem původního dotazu. Zajišťuje, že odpověď souvisí s tématem dotazu a splňuje jeho specifické požadavky.
- **Věrnost** – hodnotí, zda odpověď vychází z informací obsažených v kontextu z vyhledaných dokumentů.
- **Správnost** – měří přesnost mezi vygenerovanou odpovědí a odpovědí referenční, která obsahuje ověřené pravdivé tvrzení. Tím je možno pozorovat, zdali je systém schopen dodávat fakticky správné odpovědi mířené na kontext dotazu.

3. Další požadavky

- **Latence** – rychlosť vyhledávání a produkce odpovědi, která je nezbytná pro uživatele komunikujícího se systémem.

Zbylé tři požadavky vychází z benchmarku RGB (Retrieval-Augmented Generation Benchmark) [8]:

- **Odolnost proti šumu** – šum představují dokumenty relevantní otázce, avšak neobsahující správnou odpověď.
- **Odmítnutí** – aby se předešlo halucinacím LLM, RAG by měl odmítnout odpověď, pakliže obdržený kontext není dostatečný k poskytnutí odpovědi.
- **Odolnosti vůči protichůdným informacím** – generátor by měl identifikovat protichůdná fakta ve vyhledaných dokumentech.

Soubor dat k evaluaci

K vytváření testovacích případů k hodnocení RAG systémů je klíčový výběr vhodného souboru dat. V popisech některých existujících benchmarků jsou využívány specifické datasety, obecně je však vhodné použít např. index Wikipedie díky její rozmanitosti témat a rozsahu článků. Ověřeným přístupem je také využití sady zpráv o aktualitách. Taková data představují informace, na kterých jazykové modely nebyly trénovány, a zodpovědnost tak je plně na vyhledávacím modulu [40].

Metriky hodnocení

Cíle hodnocení uvádí požadavky na systém. Vytvoření hodnotících kritérií, která by odpovídala lidským preferencím a řešila praktické otázky, je náročné. Existuje však již několik evaluačních frameworků, které implementují řadu metrik, díky kterým je hodnocení měřitelné.

Hodnocení samotných výsledků není vždy deterministické, protože některé metriky, jako například relevance odpovědi k dotazu či věrnost vůči zdrojovým dokumentům, mohou mít subjektivní charakter a závisí na interpretaci hodnotitele. Manuální hodnocení je však časově náročné a často subjektivní. Moderním přístupem k automatizované evaluaci je koncept *LLM-as-a-Judge* [20], kde je jazykový model instruován k ohodnocení relevance, věrnosti či kvality na bodové škále. Tento přístup využívá framework popsány v následující podkapitole, který byl zároveň využit k evaluaci implementovaného RAG systému této práce v kap. 6.3.

3.3.2 RAGAs: Automatizovaná evaluace RAG systémů

Systém RAGAs (*Retrieval Augmented Generation Assessment*) [10] předkládá soubor metrik, které lze použít k automatizovanému hodnocení RAG systémů bez nutnosti lidské anotace (poskytnutí referenčních odpovědí). Zároveň využívá vhodné instruování LLM k dělání rozhodnutí, na základě kterých je vypočítáno skóre a vyčísleno od 0 do 1. Níže uvedené metriky vychází z cílů hodnocení uvedených v kapitole 3.3.1. Es et. al. [10] pak experimenty dokázali, že pomocí frameworku je dosaženo hodnocení více se shodujícího s hodnocením lidmi, nežli ohodnocení čistě pomocí jazykového modelu.

Věrnost

Odpověď je věrná kontextu, pakliže tvrzení uvedená v odpovědi lze odvodit z kontextu. Výstupní odpověď je tak pomocí jazykového modelu rozdělena na jednotlivá tvrzení, kde každé je následně ohodnoceno, zdali souvisí s kontextem.

Výsledné skóre F je vypočteno jako

$$F = \frac{V}{S}, \quad (3.1)$$

kde V je počet výroků označených LLM jako podložené kontextem a S je celkový počet tvrzení.

Relevance odpovědi

Odpověď je relevantní, pakliže se vztahuje ke vstupní otázce. Vygenerovaná odpověď zde slouží jako vstup jazykovému modelu, který generuje otázky k dané odpovědi. Otázky jsou

pak embedding modelem převedeny na vektory a je vypočtena vzájemná podobnost mezi původní otázkou a těmi vygenerovanými.

Výsledné skóre AR je vypočteno jako

$$AR = \frac{1}{n} \sum_{i=1}^n \text{sim}(q, q_i), \quad (3.2)$$

kde $\text{sim}(q, q_i)$ je kosinová podobnost mezi původní a vygenerovanou otázkou a n je počet vygenerovaných otázek.

Relevance kontextu

Kontext je relevantní, pakliže obsahuje informace potřebné k zodpovězení dotazu. Z kontextu jsou pomocí LLM extrahovány věty, které jsou relevantní k dotazu. Výsledné skóre je vypočteno jako poměr počtu extrahovaných vět k celkovému počtu vět v kontextu.

3.4 Aplikace

RAG nachází uplatnění tam, kde je třeba poskytovat odpovědi založené na rozsáhlých datech, která nejsou součástí tréninkové sady jazykového modelu. V této sekci jsou popsány praktické scénáře využití RAG.

3.4.1 Open-Domain Question Answering (OpenQA)

RAG je často využíván v tzv. *Open-Domain Question Answering*, kde je cílem zodpovědět dotaz pomocí rozsáhlé báze znalostí (např. Wikipedie). RAG je tak pro LLM tím, čím je pro studenty tzv. *open-book* zkouška. Při této zkoušce si studenti mohou přinést referenční materiály, jako jsou učebnice nebo poznámky, které mohou použít pro vyhledání informací k zodpovězení otázky a namísto jejich schopnosti memorizace jsou tak testovány jejich uvažovací schopnosti [22]. Stejně tak se RAG zaměřuje na schopnost dynamicky získávat aktuální informace, což zlepšuje faktickou správnost odpovědí, které jsou navíc důvěryhodné a transparentní díky podložení zdroji.

3.4.2 Doménově specifické systémy

Další aplikací je vyhledávání v doménově specifických dokumentech. Může jít o interní firemní dokumentaci, manuály, právní předpisy nebo vědecké články. V těchto scénářích je RAG využíván k získávání odpovědí v oblastech, kde LLM selhávají kvůli neznalosti informací, které se od jejich tréninku změnily nebo ani nemohly být přítomny (např. proprietární data).

Kapitola 4

Architektura systému

Předchozí kapitoly shrnují dosažený vědecký pokrok v oblastech vektorových databází a Retrieval-Augmented Generation. Slouží tak jako teoretický základ, který vytvořil kontext pro použití těchto technologií k návrhu a implementaci vlastního systému na zpracování a vyhledávání dokumentů.

Tato kapitola si dává za cíl představit architekturu systému, který vznikl v rámci této práce. Principem vychází z přístupu RAG. Hlavní myšlenkou je dotazování systému, který odpovídá na otázky pomocí externího zdroje informací (na bázi Q&A). Zdroj informací zde představují textové dokumenty propojené se systémem, které jsou uložené ve vektorové databázi. Jedná se tak o vyhledávání nad těmito dokumenty, přičemž otázka je podána přirozeným jazykem a odpověď je uživatelsky přívětivá, podložená fakty s odkazy na zdroje informací. Využití systému není nijak limitováno, záleží pouze na tom, jaká data na textové bázi má systém k dispozici.

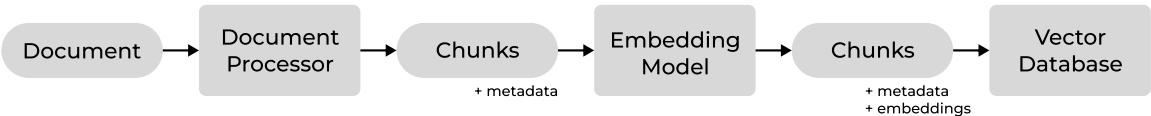
Systém je navrhnut tak, aby optimalizoval přesnost vyhledávání, která je nutná pro formulaci správné odpovědi generátorem. Vyhledávání informací z dokumentů je cílové použití, jedná se však až o druhou fázi. První fázi systému je zpracování dokumentů a jejich uložení do databáze.

4.1 Zpracování dokumentů

Prvním procesem je zpracování dokumentů. Vstupem je dokument, který je konvertován na prostý text. Ačkoliv byl systém testován především na textových dokumentech, podporovanými typy souborů jsou `.txt`, `.doc`, `.docx`, `.pdf`, `.jpg`, `.png` a `.heic`. Na binární soubory, jako jsou obrázky a PDF dokumenty, se aplikuje OCR (optické rozpoznávání znaků) k extrakci textu. Extrahovaný text prochází předzpracováním, které zahrnuje čištění textu a případnou detekci nadpisů. Dále je rozdělen na tzv. chunky, které představují kratší pasáže textu. To vše je úkol modulu `DocumentProcessor`.

Každý chunk je obohacen o metadata zahrnující informace o dokumentu, ze kterého textová pasáž pochází, a pro snadnější lokalizaci i ve které sekci nebo na které straně se nachází. Dále obsahuje údaje o přístupových právech k dokumentu, které umožňují filtrování.

Jakmile jsou dokumenty rozděleny na chunky a obohaceny o metadata, dalším krokem je jejich vektorizace. Text každého chunku je převeden do vektorové reprezentace pomocí embedding modelu. Ty jsou přidány k chunkům, které se ukládají do vektorové databáze.



Obrázek 4.1: Diagram zpracování dokumentu až po jeho uložení do databáze

4.1.1 Chunking

Chunking je proces dělení textu na sekce velikostně optimalizované pro zpracování jazykovými modely a zachycení kontextu. Zatímco dokumenty jsou většinou příliš velké a zahrnují široký, mnohdy různorodý kontext, chunking dělí dokumenty na logické sekce zachycující jeden kontext, které jsou navíc velikostně přizpůsobené vektorizaci embedding modelem, který má svůj limit.

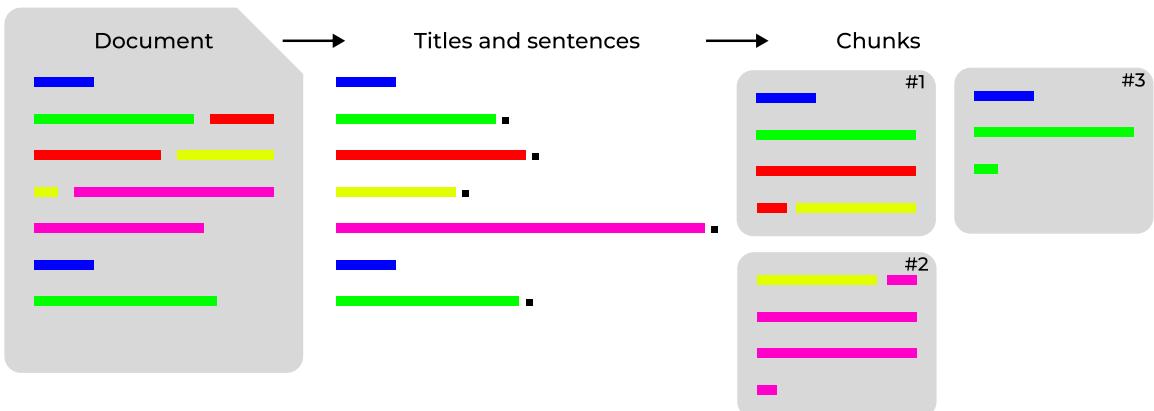
Strategie chunkingu se dělí podle komplexnosti. Naivním přístupem je dělení textu na bloky o fixní délce, to však může rozdělit jeden kontext do více pasáží, navíc i uprostřed slova. Rekurzivní přístup pak dělí text podle struktury (odstavce, nadpisů, věty) a dodržuje limit velikosti chunku.

Metoda, kterou systém implementuje, se snaží co nejvíce zachovat kontext a strukturu textu, přičemž velikost chunku omezuje vstupní limit embedding modelu. Ten je počítán v tokenech, a proto i text je tokenizérem příslušného embedding modelu přepočítáván na tokeny. Na obrázku 4.2 je zobrazen princip tokenizace¹.

The example sentence is being tokenized. Note that punctuation usually takes its own token. In the English language, a token takes about four characters on average.

Obrázek 4.2: Vizualizace tokenizace

Dokument je rozdělen na jednotlivé věty, které jsou přidávány do chunku. Pakliže by počet tokenů ve větě přesáhl limit chunku, aktuální chunk se uzavře a věta je přidána do nového chunku společně s poslední větou předchozího chunku. Tím dochází k překryvu, který zabraňuje ztrátě kontextu mezi sousedními částmi textu. Nadpisů pak vždy tvoří počátek nového chunku, protože označují začátek nového logického celku.



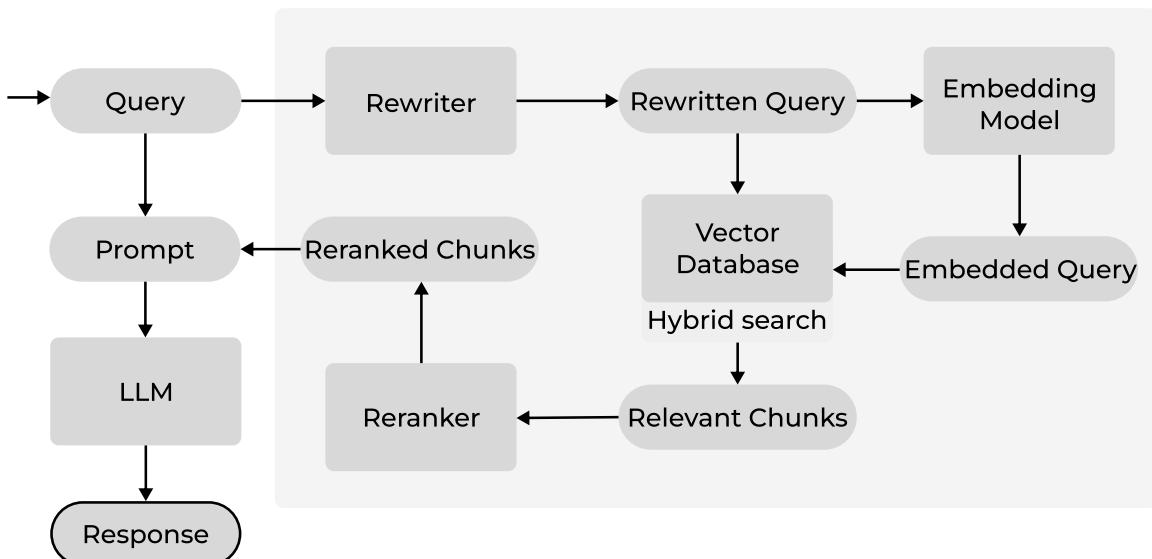
Obrázek 4.3: Chunking

¹nástroj na vizualizaci tokenizace je dostupný na <https://gpt-tokenizer.dev/>

4.2 Vyhledávání dokumentů

Vyhledávání dokumentů pro vytvoření kontextu a poskytnutí odpovědi formulované jazykovým modelem je hlavním případem užití systému. Může k němu dojít, jakmile vektorová databáze obsahuje zpracovaná data, která slouží jako báze znalostí.

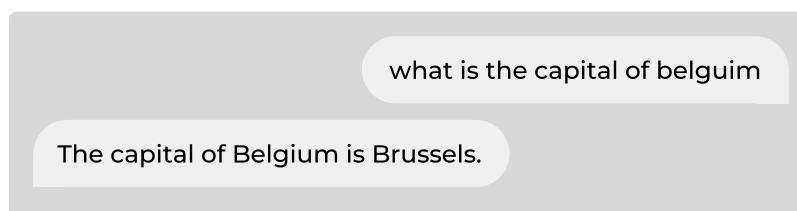
Dotaz uživatele je přeformulován do podoby optimalizované pro vyhledávání. Přeformulovaný dotaz je použit pro hybridní vyhledávání, které navrácí relevantní chunky. Ty jsou seřazeny dle skóre vyhledávání, jejich pořadí a relevance je však upřesněna pomocí modulu **Reranker**. Tyto chunky pak slouží jako kontext pro jazykový model, který pomocí něj zodpovídá dotaz uživatele.



Obrázek 4.4: Diagram komponent vyhledávání dokumentů a generování odpovědi

4.2.1 Přeformulování dotazu

Modul **Rewriter** má v systému dva módy. První slouží k optimalizaci dotazů pro vyhledávání. Uživatelé často formulují své dotazy neefektivně tím, že používají nadbytečná nebo vycpávková slova. Při zadávání dotazu uživatelem přes klávesnici navíc může dojít k překlepům, které by mohly negativně ovlivnit výsledek vyhledávání. Tento modul využívá jazykový model a vhodně ho instruuje k přeformulování dotazu do stručnější a informativnější podoby, aniž by změnil jeho původní význam.



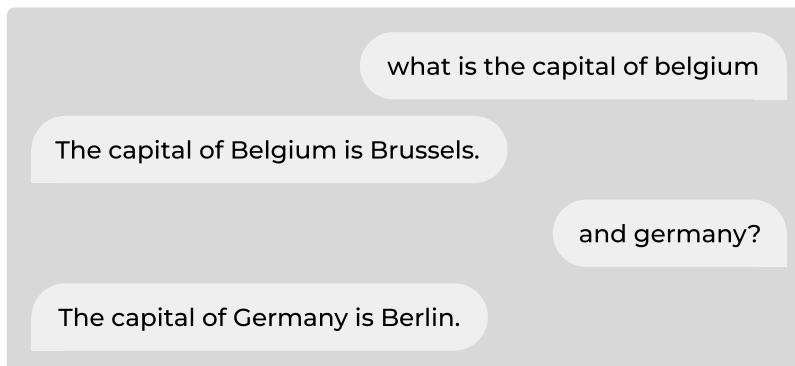
Obrázek 4.5: Příklad uživatelského vstupu (nahoře), který je pro vyhledávání přeformulován na dotaz „capital of Belgium“, a odpovědi

Klade si tak za cíl zpřesnit obě složky hybridního vyhledávání. U fulltextového vyhledávání je dotaz omezen na relevantní slova a u sémantického vyhledávání vede zjednodušený dotaz k lepší vektorové reprezentaci, která přesněji odpovídá uloženým embeddingům a eliminuje zkreslení způsobené nadbytečnými slovy.

Vhodně formulované dotazy pak ponechává beze změny. V případě nesmyslných nebo nejednoznačných dotazů k přeformulování také nedochází, aby nebyly ovlivněny další fáze vyhledávání, které s těmito případy počítají.

4.2.2 Chatová historie

Druhým módem modulu **Rewriter** je podpora chatové historie a navazujících otázek. Při zadání neúplného dotazu, který se odkazuje k předešlým otázkám a odpovědím, je tento dotaz s pomocí tří posledních párů dotaz–odpověď přeformulován jazykovým modelem do kompletní podoby.



Obrázek 4.6: Příklad navazující otázky „and germany?“, která je pro vyhledávání i výsledný prompt generátoru přeformulována na dotaz „What is the capital of Germany?“

Tento mód přináší novou funkctionalitu, pozorováním však bylo zjištěno celkové snížení výkonu systému, jelikož se efektivita tohoto módu projevuje pouze u navazujících dotazů, které potřebují být obohaceny. Při úplných dotazech, které první mód zjednoduší, může u tohoto módu dojít k mírnému přeformulování dotazu a následné nekonzistenci mezi otázkou a odpovědí. Generátor totiž odpovídá na přeformulovaný dotaz, zatímco rewriting bez chatové historie slouží pouze pro vyhledávání.

4.2.3 Hybridní vyhledávání

Hybridní vyhledávání kombinuje fulltextové a vektorové vyhledávání k využití obou metod pro přesnější výsledky. Motivací tohoto přístupu je fakt, že dotazy bývají různorodě zaměřené.

- **Fulltextové vyhledávání** je přesné při hledání konkrétních klíčových slov. Pokud ale uživatel použije synonyma nebo volnější formulaci dotazu, bývá nepřesné. Vstupem fulltextovému vyhledávání je přeformulovaný dotaz.
- **Vektorové vyhledávání** dokáže najít sémanticky podobné výsledky i při odlišné formulaci, ale nemusí vždy zaručit, že nalezené dokumenty obsahují specifická klíčová slova, což může vést k nižší přesnosti u některých dotazů. Vstupem vektorovému

vyhledávání je vektorová reprezentace přeformulovaného dotazu, vytvořená stejným embedding modelem, který byl použit ke zpracování dokumentů.

Hybridní vyhledávání využívá parametr α , který určuje váhu mezi fulltextovým a vektorovým vyhledáváním. Je vyčíslen hodnotou $\alpha \in \langle 0, 1 \rangle$, kde 0 se přiklání fulltextovému vyhledávání a 1 vektorovému. Výsledné skóre hybridního vyhledávání je pak vypočteno jako vážený průměr normalizovaných hodnot skóre jednotlivých vyhledávání:

$$s = \alpha \cdot \frac{s_v - \min(s_v)}{\max(s_v) - \min(s_v)} + (1 - \alpha) \cdot \frac{s_t - \min(s_t)}{\max(s_t) - \min(s_t)}, \quad (4.1)$$

kde s_t je skóre fulltextového vyhledávání (např. BM25) a s_v je skóre vektorového vyhledávání, které může být založeno např. na kosinové podobnosti nebo Euklidovské vzdálenosti.

4.2.4 Reranking

Tato část vychází převážně ze zdroje [1], kde je proces vyhledávání označen za dvoufázový, protože důležitým krokem po hybridním vyhledávání je tzv. *reranking*. Jedná se o proces přeuspřádání či zpřesnění výsledků původního vyhledávání podle míry relevance k dotazu uživatele.

Samotné vyhledávání často upřednostňuje rychlosť před přesností, jelikož probíhá nad rozsáhlým množstvím dat (viz ANNS 2.5.2). V této fázi se typicky využívá architektura *Bi-encoder*, kde jsou dotaz a dokument zakódovány odděleně a jejich relevance je určena pomocí podobnostní funkce. Tento přístup je rychlý, ale může u něho docházet ke ztrátě sémantických detailů. Jeho výsledkem je pevně daný počet nejrelevantnějších chunků.

Reranking využívá předtrénovaný model typu *Cross-encoder*, který dotaz a jednotlivé chunky zpracovává společně jako jeden vstup a dokáže tak detailněji zachytit jejich sémantickou souvislost. Každé dvojici dotaz–chunk je přiřazeno skóre relevance, podle kterého jsou výsledky seřazeny. Zároveň provádí filtr tím, že informace s nedostatečnou mírou relevance z kontextu odstraní. Vzhledem k vyšší výpočetní náročnosti se reranking aplikuje pouze na omezený počet chunků, právě proto následuje samotnému vyhledávání.

Výsledkem by měla být vyšší kvalita odpovědi generované jazykovým modelem, protože se do promptu dostávají pouze skutečně relevantní informace a LLM tak není zahlceno přebytečnými informacemi.

4.2.5 Prompt Engineering

Posledním krokem před samotným generováním odpovědi jazykovým modelem je návrh promptu, který pak slouží jako jeho vstup. Cílem *prompt engineeringu* je navrhnout takový vstup, který co nejlépe instruuje model ke správné odpovědi dodržující zamýšlený formát.

Návrh promptů v RAG systémech je zásadní, jelikož model nesmí odpovídat na základě natrénovaných dat, ale musí kombinovat dotaz s poskytnutým kontextem. Dle dokumentace OpenAI [27] patří mezi základní principy prompt engineeringu:

- **Využití rolí** – při jednom volání lze využít více zpráv různé priority, konkrétně `developer` zpráva typicky obsahuje instrukce a pravidla a má vyšší prioritu, `user` zpráva pak obsahuje vstup, na který jsou pravidla uplatněna.
- **Formátování a struktura promptu** – pomocí Markdown formátování a XML tagů lze pomocí modelu s pochopením instrukcí a logicky strukturovat prompt. Struktura

promptu pak obecně bývá členěna do sekcí v následujícím pořadí: identita, instrukce, příklady a kontext.

- **Identita** – nastavení identity (mimo samotné instrukce) poskytuje modelu rámec chování a stylu odpovědi, říká mu, jaká je jeho role.
- **Few-shot learning** – tato metoda poskytuje v promptu pár ukázkových příkladů vstup–výstup, ze kterých si model implicitně vyvodí požadované chování.
- **Kontext** – poskytnutí kontextu s relevantními informacemi k zodpovězení dotazu je celý princip Retrieval-Augmented Generation.

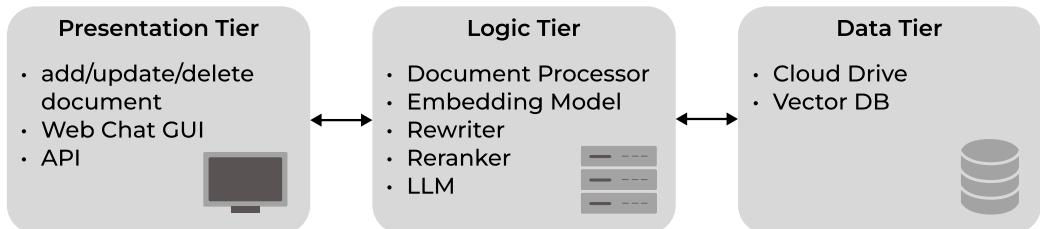
Prompty mají předdefinované instrukce a následně jsou tvořeny a formátovány dynamicky v kódu na základě dotazu a vyhledaného kontextu. Použité prompty v systému jsou uvedeny v kapitolách 5.2.4 a 5.2.6.

4.3 Integrace komponent

Předchozí podkapitoly uvedly princip dvou hlavních fází systému:

1. **Zpracování**, kde vstupem je textový dokument, který je zpracován a uložen do vektorové databáze.
2. **Vyhledávání**, kde vstupem je dotaz uživatele a výstupem je odpověď formulovaná LLM pomocí vyhledaného kontextu.

Tyto fáze se mohou neustále opakovat a společně s integrací cloudového úložiště a webového uživatelského rozhraní tvoří celou architekturu systému. Jednotlivé komponenty lze pak rozdělit podle jejich funkční odpovědnosti na třívrstvou architekturu znázorněnou na obrázku 4.7.



Obrázek 4.7: Třívrstvá architektura systému

4.3.1 Synchronizace s cloudovým úložištěm

Součástí systému je integrace s cloudovým úložištěm, které slouží pro správu dokumentů tvorících bázi znalostí. Uživatel může dokumenty nahrávat, upravovat nebo mazat ve vyhrazené složce v prostředí cloudové služby, kterou si se systémem propojí.

Tato složka je systémem monitorována. Jakmile se její stav změní, systém je o tom prostřednictvím API cloudové služby notifikován a zahajuje synchronizační rutinu, která obnáší detekci změn a zajištění jejich promítnutí do databáze.

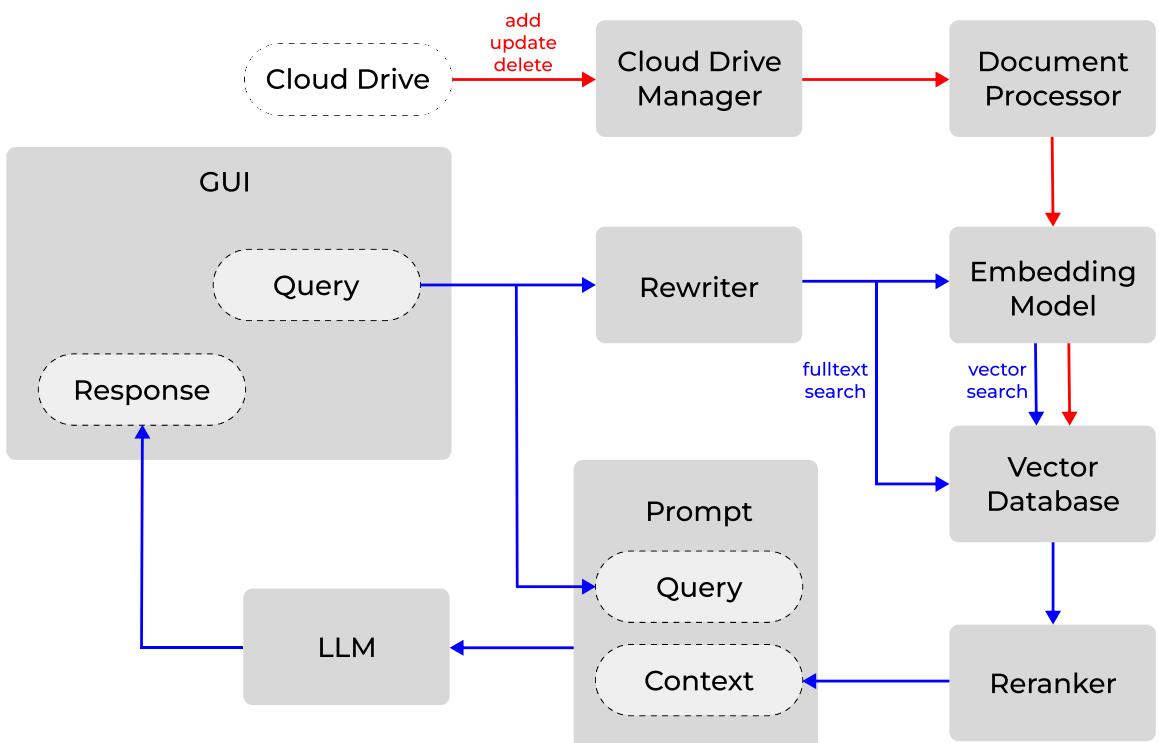
Při detekci nového dokumentu je tento dokument stažen a prochází fází zpracování. V případě smazání dokumentu jsou příslušné chunky z databáze odstraněny a při detekci

změn v již nahraném dokumentu dochází ke kombinaci těchto procesů, tedy smazání a znovuzpracování.

Tímto způsobem je zajištěno, že znalostní báze systému vždy reflektuje aktuální stav cloudového úložiště automatickou synchronizací.

4.3.2 Tok dat v systému

Na obrázku 4.8 je znázorněna celková architektura systému, vzájemné propojení všech komponent a tok dat mezi nimi. Červené šipky označují tok dat ve spojení synchronizace cloudového úložiště, které nahraje dokumenty, které podléhají procesu zpracování popsaného v kap. 4.1. Modré šipky značí proces vyhledávání dokumentů a poskytnutí odpovědi na dotaz z kap. 4.2.



Obrázek 4.8: Schéma komponent znázorňující tok dat mezi jednotlivými moduly systému. Červené šipky označují proces nahrání dokumentů z cloudového úložiště, zpracování a uložení do vektorové databáze. Modré šipky představují proces mezi dotazem uživatele až po zobrazení odpovědi systémem v uživatelském rozhraní.

4.4 Využití

Systém byl navržen pro vyhledávání nad interními textovými dokumenty firem. Pro účely testování byl použit dataset článků Wikipedie, a tak využití systému je obecné a přímo závisí na poskytnutých datech tvořících znalostní bázi. Pakliže dokumenty následují předpovídatelný formát, fázi zpracování je možné doladit konkrétnímu použití, avšak navržena byla pro univerzální použití.

Kapitola 5

Implementace

Tato kapitola se věnuje technické realizaci systému navrženého v předchozí kapitole. Popisuje především použité technologie, knihovny, modely a jejich výběr. Následuje způsob implementace jednotlivých komponent. Dále je zde vysvětleno, jakým způsobem systém komunikuje s externími službami a jak jsou zajištěny interakce mezi jednotlivými moduly systému a uživatelským rozhraním prostřednictvím API. Při zmínkách použitých technologií u implementačních detailů jsou uvedeny citace k dokumentacím, které byly při implementaci využity. Implementace je zveřejněna na portálu GitHub pod licencí MIT¹.

5.1 Použité technologie

Při implementaci systému byla využita existující řešení pro jednotlivé dílčí úkoly systému, které byly integrovány do funkčního celku Retrieval-Augmented Generation.

Systém je implementován v jazyce Python. Vývoj a testování systému probíhalo lokálně pomocí technologie Docker Compose, která umožnila spouštění jednotlivých komponent (API server, databáze, frontend) v konzistentním prostředí. Níže je uveden přehled hlavních knihoven a nástrojů, které byly v rámci implementace použity:

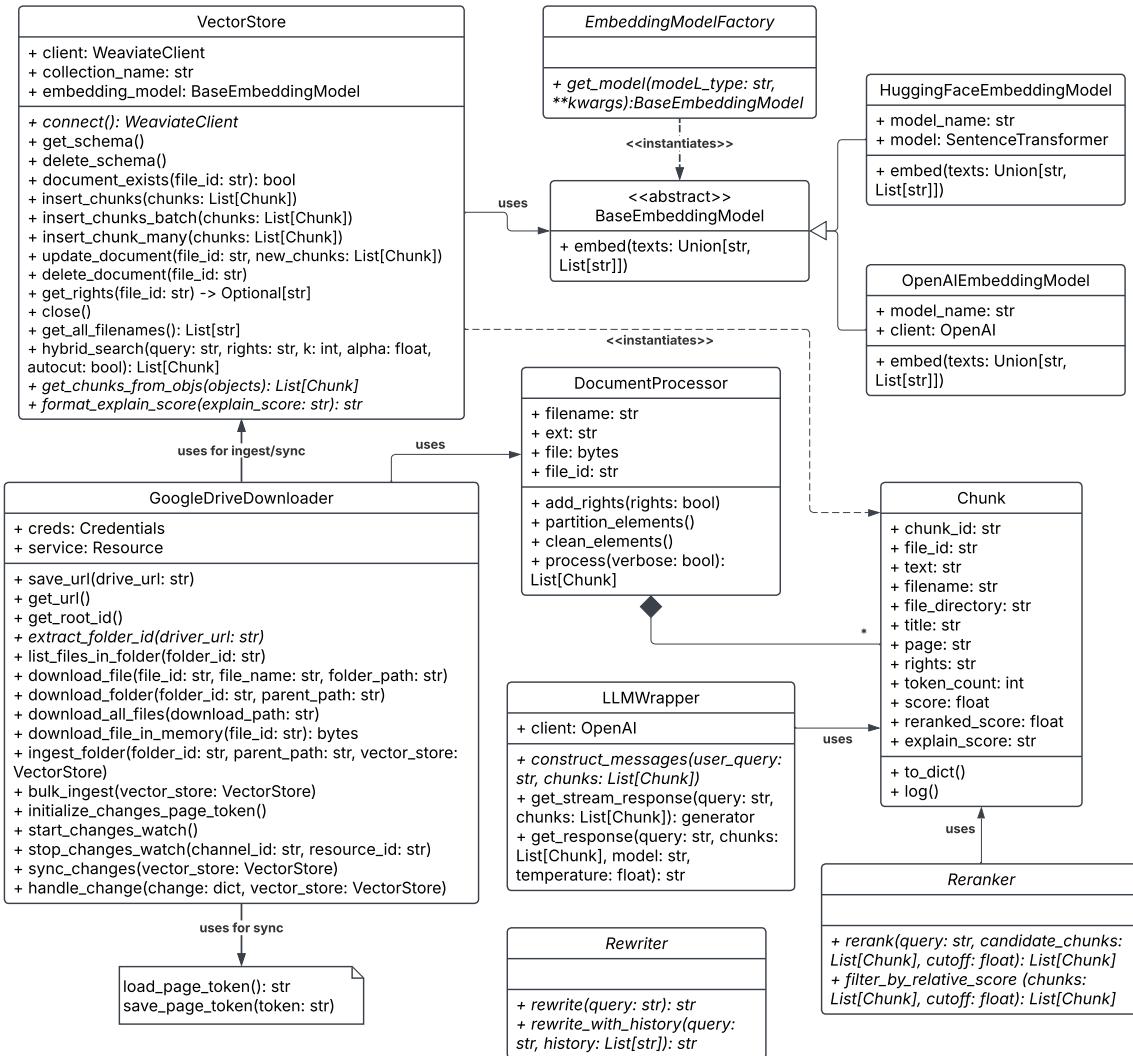
- **FastAPI** – webový framework k propojení backendu a frontendu skrze API.
- **Unstructured** – knihovna s rozhraním pro zpracování nestrukturovaných dat.
- **NLTK** – knihovna pro zpracování přirozeného jazyka pro rozdelení textu na věty.
- **Transformers** (HuggingFace) – knihovna pro práci s předtrénovanými jazykovými modely, využita pro tokenizér.
- **Sentence Transformers** (HuggingFace) – knihovna s embedding a reranker modely.
- **Weaviate** – open-source vektorová databáze s podporou hybridního vyhledávání.
- **OpenAI API** – přístup ke generativním modelům GPT.
- **Google Drive API** – rozhraní pro sledování změn a stahování dokumentů pro synchronizaci databáze se složkou na Google Drive cloudovém úložišti.

Konkrétní použití těchto technologií je rozebráno v následujících sekci.

¹<https://github.com/adamvalik/BP>

5.2 Moduly systému

Systém byl navržen modulárně tak, aby jednotlivé zodpovědnosti byly rozděleny do tříd, které lze samostatně testovat nebo případně snadno měnit. Na obrázku 5.1 je uveden diagram tříd s vyznačenými hlavními vztahy mezi jednotlivými moduly a jejich atributy a metodami.



Obrázek 5.1: Diagram tříd. Kurzívou jsou označeny statické metody.

5.2.1 DocumentProcessor

Třída `DocumentProcessor` slouží ke zpracování dokumentů. Při inicializaci dostane cestu k uloženému dokumentu nebo dokument v paměti po bytech. Metoda `process()` pak vrací seznam vytvořených chunků k uložení do databáze. Zpracování probíhá ve třech krocích:

1. **Extrakce textu** je realizována pomocí knihovny `Unstructured` [36]. Ta poskytuje metody `partition()` pro různé typy souborů, které mají výstupem seznam elementů reprezentujících logické celky dokumentu. Každý element obsahuje extraovaný text, metadata a kategorie popisující účel v dokumentu (např. záhlaví, náro

tivní text, nadpis apod. – možné kategorie elementu se odvíjí od typu dokumentu). **Unstructured** je high-level knihovna a pro zpracování různých typů souborů využívá externí nástroje, které je třeba mít nainstalované:

- pro **DOC** a **DOCX** je zapotřebí **libreoffice**,
 - pro binární soubory jako **PDF**, **JPG**, **PNG** a **HEIC** slouží k extrakci textu optické rozpoznávání znaků (OCR) za použití modulu **tesseract**,
 - nástroj **poppler** je pak využíván k renderování **PDF** dokumentů.
2. **Čištění textu** je mezikrokem zpracování a upravuje podobu textu pro lepší výsledky chunkování především odstraněním přebytečných mezer, emoji a odrážek v seznamech. Konfigurovatelným krokem k předpovídání struktury datasetu je pak detekce nezaznamenaných nebo naopak odstranění přebytečných nadpisů, což vede k lepšímu dělení textu. Tento krok tak lze manuálně přizpůsobit pro sady dokumentů se specifickým očekávaným formátem pro lepší zpracování.
3. **Chunkování** je implementováno podle principu popsáного v kapitole 4.1.1.

Text jednotlivých elementů je dělen na věty. Věty jsou detekovány pomocí funkce `nltk.tokenize.sent_tokenize(text)` [25] a dále analyzovány na počet tokenů. O tokenizaci se stará `AutoTokenizer` [16], jehož architektura je odvozena od použitého embedding modelu.

Embedding model `all-mpnet-base-v2`, pro který jsou chunky připravovány, má limit 384 tokenů na vstupu a tomu je velikost chunku přizpůsobena, jelikož přesah limitu vstupu tento embedding model zahazuje. Menší chunky by naopak nemusely poskytnout dostatek kontextu.

Pro uzavření chunku a vytvoření nového slouží mimo limit taky detekce nadpisů. Ta využívá kategorie elementů **Title**.

5.2.2 EmbeddingModel

Pro výběr embedding modelu byla navržena abstraktní třída `BaseEmbeddingModel` definující rozhraní `embed()`. Ta slouží jako základ pro komponentu embedding modelu, která je zodpovědná za vektorizaci textových pasáží chunků pro uložení do vektorové databáze a také vektorizaci dotazu pro následné vyhledávání.

Třída `HuggingFaceEmbeddingModel` využívá modely ze sady `sentence-transformers`. Pro systém byl využit model `all-mpnet-base-v2`, který generuje embeddingy o dimenzi 768 a je navržen pro obecné použití. Z dostupných modelů poskytuje nejlepší kvalitu, je však pomalejší oproti běžně používanému `all-MiniLM-L6-v2` (s dimenzí 384) [35]. Vyšší dimenzionalita umožňuje modelu zachytit jemnější významové detaily v datech, což může vést k lepší sémantické reprezentaci, ale většímu náruku na výkon a paměť.

Třída `OpenAIEmbeddingModel` využívá rozhraní OpenAI API [26] k získání embeddingů. Model `text-embedding-3-small` generuje embeddingy o dimenzi 1536. Tento způsob byl zkoumán jako alternativní, nicméně tato služba je zpoplatněná a závislá na externím poskytovateli, kterému by bylo třeba při prvotním zpracování dokumentů zaslat všechna data, a tak bylo preferováno využití lokálního modelu z knihovny `sentence-transformers` pro finální verzi systému pro snížení nákladů a zvýšení kontroly nad soukromím.

Výběr konkrétní implementace se provádí tovární třídou `EmbeddingModelFactory`, která podle parametru `model_type` vrací odpovídající instanci embedding modelu.

5.2.3 VectorStore

Pro implementaci externího úložiště znalostí s hybridním vyhledáváním byla využita open-source vektorová databáze Weaviate a její API [37], která poskytuje jednoduché řešení pro vývoj v docker kontejneru podporující hybridní vyhledávání. Při každém přístupu do databáze je inicializován klient, který se k ní připojí.

Pro účely ukládání chunků pak slouží jedna kolekce:

```
collection = client.collections.create(
    name=collection_name,
    vector_index_config=Configure.VectorIndex.hnsw(
        distance_metric=VectorDistances.COSINE,
    ),
    properties=[
        Property(name="chunk_id", data_type=DataType.TEXT),
        Property(name="file_id", data_type=DataType.TEXT),
        Property(name="text", data_type=DataType.TEXT),
        Property(name="filename", data_type=DataType.TEXT),
        Property(name="file_directory", data_type=DataType.TEXT),
        Property(name="title", data_type=DataType.TEXT),
        Property(name="page", data_type=DataType.TEXT),
        Property(name="rights", data_type=DataType.TEXT)
    ],
)
```

Konfigurace kolekce využívá algoritmus HNSW (viz kap. 2.5.5) k indexaci a kosinovu podobnost jako podobnostní funkci při vektorovém vyhledávání. Pro hybridní vyhledávání bylo využito rozhraní metody `collection.query.hybrid()`, které implementuje vektorová databáze Weaviate a kombinuje vektorové vyhledávání a BM25. Strategie volby parametrů byla zvolena na základě testování a evaluace systému (viz kap. 6) následovně:

```
response = collection.query.hybrid(
    query=query,
    vector=embedding_model.embed(query)[0],
    alpha=0.55,
    fusion_type=HybridFusion.RELATIVE_SCORE,
    return_metadata=MetadataQuery(score=True, explain_score=True),
    auto_limit=3,
    filters=Filter.by_property("rights").equal(rights)
)
```

`HybridFusion.RELATIVE_SCORE` označuje algoritmus sloučení hybridního skóre, kde relativní skóre je vyjádřeno vzorcem zmíněným v kapitole 4.2.3. Parametr `auto-limit` představuje tzv. *autocut*, který nevrací pevný počet vyhledaných záznamů, ale daný počet prvních shluků záznamů dle jejich skóre, např. pro {0,97; 0,96; 0,955; 0,92; 0,91; 0,89}:

- `auto_limit=1` → {0,97; 0,96; 0,955}
- `auto_limit=2` → {0,97; 0,96; 0,955; 0,92; 0,91}
- `auto_limit=3` → {0,97; 0,96; 0,955; 0,92; 0,91; 0,89}

Tento přístup tak dokáže navracet variabilní počet nejrelevantnějších chunků. Jelikož dotazy mohou být vágní a nemířit tak na konkrétní informace, je tento parametr nastaven volněji, aby byl vyhledán širší kontext relevantních chunků, který je zpracován rerankingem. Dále je aplikován filtr dle přístupových práv uživatele k ilustraci filtrování metadat.

5.2.4 Rewriter

Modul `Rewriter` zaobaluje oba jeho módy. Statická metoda `rewrite()` využívá OpenAI API [27] pro volání modelu gpt-4o k přeformulování dotazu uživatele do podoby optimálnější pro vyhledávání. Použitý prompt vypadá následovně:

```
system: You are a query rewriting assistant in a Retrieval-Augmented Generation (RAG) system that uses both semantic and keyword-based search (hybrid retrieval).
```

Your task is to rewrite user queries to improve retrieval performance by:

- Making the query **more specific, structured, and searchable**.
- Emphasizing **important keywords or phrases** that are likely to appear in source documents.
- Removing filler phrases like 'tell me about', 'please explain', 'can you describe', etc.
- Avoiding full-sentence questions unless absolutely necessary.
- Keeping the rewritten query as **concise** as possible.
- Never adding assumptions or new topics not present in the original query.
- If the query is nonsensical, return it unchanged.

Examples:

- Input: 'Tell me something about AI agents' → Output: 'AI agents overview'
- Input: 'Can you explain vector databases in machine learning?' → Output: 'vector databases in machine learning'
- Input: 'aodwhuoaed' → Output: 'aodwhuoaed'

user: {query}

Podporu chatové historie a navazujících dotazů řeší metoda `rewrite_with_history()`, lišící se promptem, který zahrnuje i poslední tři páry dotaz–odpověď pro uvedení kontextu pro přeformulování dotazu, pakliže je neúplný:

```
system: You are a query rewriting assistant in a Retrieval-Augmented Generation (RAG) system that uses both semantic and keyword-based search (hybrid retrieval).
```

Your task is to rewrite user queries to improve retrieval performance by:

- Keep the rewritten query semantically equivalent to the original while keeping the keywords.
- Do NOT add information or assumptions not present in the original query or context.
- If the query is nonsensical, too short, or unrewritable, just return it unchanged.

The user query may be a follow-up question. Use the recent chat history below **only to resolve ambiguous or incomplete queries**.

Recent Chat History:

```
Q: {history_question}  
A: {history_answer}  
...  
  
user: {query}
```

5.2.5 Reranker

Modul **Reranker** mění pořadí a upřesňuje míru relevance chunků, které byly vyhledány. Využívá model *cross-encoder/ms-marco-MiniLM-L-6-v2* [34]. Pomocí metody `rank(query, candidates)` pak určuje skóre relevance, dle kterého jsou chunky nově řazeny pro formulaci kontextu. Skóre se pohybuje přibližně v intervalu $[-10, 10]$.

Na seřazené chunky je aplikován filtr pomocí metody `filter_by_relative_score()`. Ta nejprve posune všechna skóre rerankingu tak, aby byla všechna nezáporná. Následně vypočíte filtrační práh jako podíl z maximálního skóre daný parametrem `cutoff`, jehož hodnota byla evaluací (kap. 6.3.3) stanovena na 0,5. Tento výpočet bere v úvahu dynamickou podobu rozsahu skóre rerankingu jednotlivých záznamů. Vyhledané záznamy se skórem nižším než tento vypočtený práh jsou označeny za nerelevantní vyřazením z kontextu.

5.2.6 LLMWrapper

Nadstavbu nad voláním jazykového modelu obstarává modul **LLMWrapper**. Vstupem je dotaz a chunky tvořící kontext k zodpovězení dotazu. Prvním krokem je sestavení promptu, který definuje instrukce pro model a formát očekávané odpovědi:

```
system: You are a factual answer generator in a Retrieval-Augmented  
Generation (RAG) system.
```

```
You will receive a user query and a list of context chunks retrieved from  
a document store. Each chunk contains text from a specific file, with  
optional section titles or page numbers.
```

```
Your job is to answer the user's question using **only** the provided  
context. Cite the sources used in your answer using the format:
```

```
[File: filename, Section: title, Page: page]. If section or page is  
missing, omit them from the citation.
```

****Rules:****

- Do NOT use any knowledge outside the context.
- Do NOT speculate or invent information.
- If at least one chunk contains relevant information, answer using only that content.
- Do NOT include a fallback message unless **none** of the chunks are even partially relevant.
- Do NOT try to give an exhaustive answer - partial answers are acceptable.

- Do NOT explain what is missing.
- Format your answer using **markdown**, be **short and factual**, and avoid repetition.

If there is truly no relevant information in any chunk, respond with:
My knowledge base does not provide information for this query.

Context:

```
[File: {filename}, Section: {title}, Page: {page}]
{text}
...
```

```
user: {query}
```

Druhým krokem je samotné volání jazykového modelu, které probíhá prostřednictvím rozhraní OpenAI API [27] za využití modelu gpt-4o, přičemž odpověď je generována po částech formou *streamu*, čímž se zlepšuje vnímání odezvy systému ze strany uživatele. Teplo modelu je nastavena na nízkou hodnotu 0,2, což zajišťuje konzistentnější odpověď a nižší míru kreativity odpovědi vhodnou pro Q&A.

Kód pro generování odpovědi pak vypadá následovně:

```
stream = client.chat.completions.create(
    model="gpt-4o",
    messages=prompt,
    temperature=0.2
    stream=True
)

for response in stream:
    yield response.choices[0].delta.content
```

5.2.7 GoogleDriveDownloader

Třída `GoogleDriveDownloader` zajišťuje integraci systému s cloudovým úložištěm Google Drive. Jejím úkolem je stažení a synchronizace dokumentů tvořících znalostní bázi, a to jak v rámci prvotního zpracování, tak průběžně pomocí sledování změn.

Dokumenty jsou z cloudu stahovány metodou `download_file_in_memory()`, která vrací binární obsah dokumentu, který je předán ke zpracování a nahrán do databáze. Tento proces tak probíhá přímo v paměti bez nutnosti uložení na disk.

Při připojení složky je provedeno prvotní nahrání všech dokumentů realizováno v rámci metody `bulk_ingest()`. Dokumenty složky jsou zpracovávány rekurzivně včetně vnořených podadresářů.

Pro zajištění automatické synchronizace stavu dokumentů využívá třída API rozhraní Google Drive Changes API [13]. Metodou `start_changes_watch()` je registrován tzv. *webhook*, který upozorňuje systém na každou změnu. Tyto změny jsou zpracovány metodou `sync_changes()`, která využívá `page_token`, který identifikuje poslední stav změn. To umožňuje získat pouze změny od poslední synchronizace, které jsou obstarávány metodou `handle_change()` pro učinění změn (nový dokument, aktualizace, smazání).

5.3 API

Serverová část systému komunikuje s frontendem skrze API realizované webovým frameworkem FastAPI.

5.3.1 API endpointy

- GET `/driveurl` – Ukládá URL sledované složky.
- POST `/ingest_folder` – Provádí prvotní nahrání dokumentů složky do vektorové databáze metodou `bulk_ingest()` třídy `GoogleDriveDownloader`.
- POST `/delete_schema` – smaže celou databázi.
- POST `/query` – provádí celý proces vyhledávání a vrací odpověď v podobě *streamu*.
- POST `/webhook` – obstarává přicházející notifikace změn ve sledované složce cloudového úložiště Google Drive.
- GET `/sync` – manuálně spouští synchronizaci změn sledované složky.
- GET `/filenames` – vypíše názvy dokumentů uložených v databázi.

5.3.2 Webhook

Koncový bod POST `/webhook` je pro Google Drive Changes API [13] zveřejněn pomocí nástroje `ngrok` [24]. Tím je zajištěno, že při jakýchkoliv změnách provedených ve sledované složce bude odeslána zpráva na tento koncový bod. Ten pak přijímá parametr `resource_state`, který označuje stav příchozí zprávy. Při stavu `changes` je zapnuta synchronizační rutina.

5.3.3 StreamingResponse

Koncový bod API POST `/query` vrací `StreamingResponse` [12] pro poskytnutí odpovědi po částech tak, jak je generována. Na frontend posílá tok zpráv ve formátu JSON, přičemž první zpráva obsahuje serializované chunky, které byly použity pro tvorbu kontextu, a zbytek zpráv tvoří odpověď z generátoru, který vrací `LLMWrapper`. Generátor, sloužící jako vstup pro `StreamingResponse`, pak vypadá následovně:

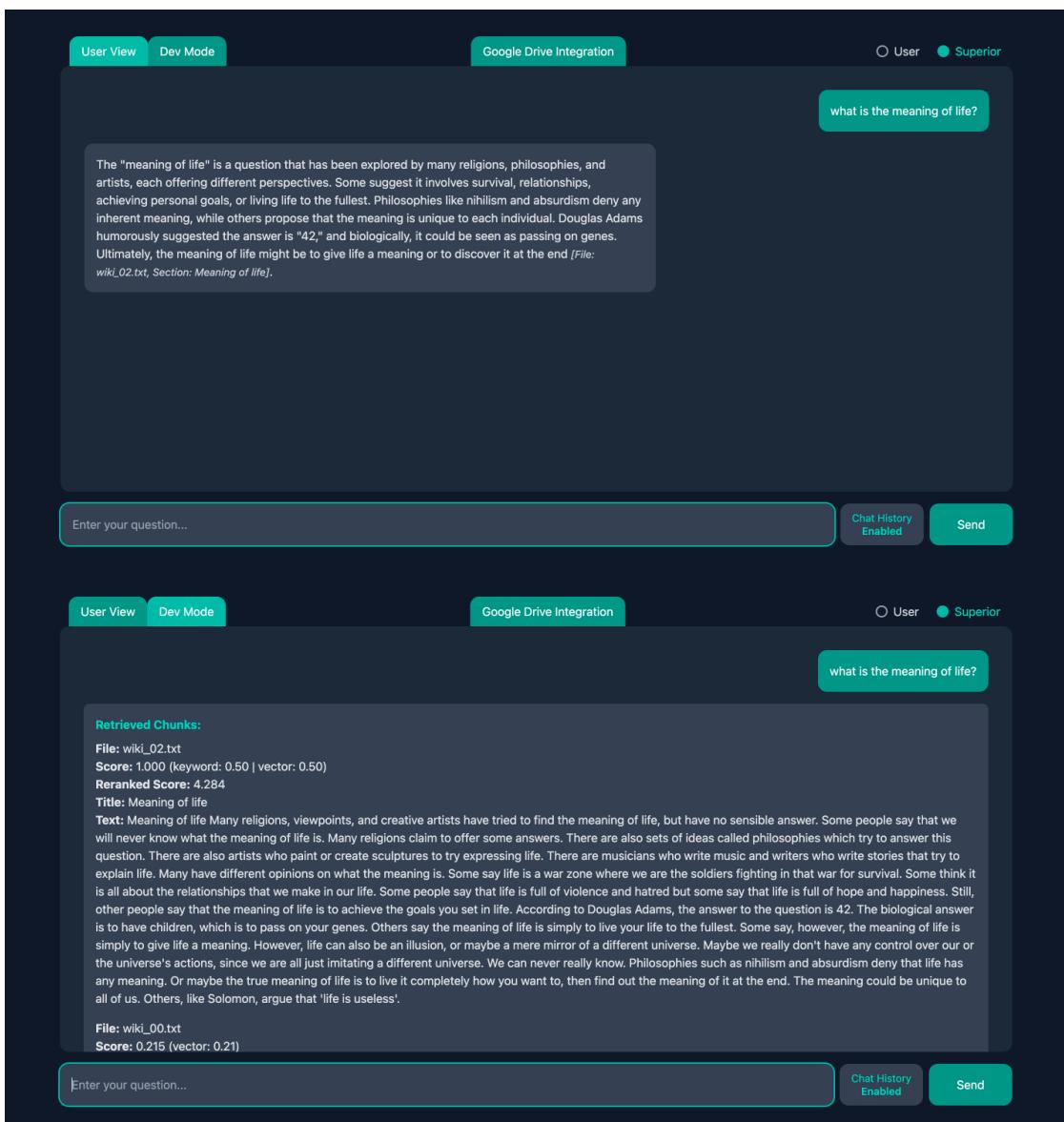
```
def stream():
    serialized_chunks = [vars(chunk) for chunk in reranked_chunks]
    yield json.dumps({
        "text": None,
        "metadata": {"chunks": serialized_chunks}
    }) + "\n"
    for llm_response in llm_wrapper.get_stream_response(
        query, reranked_chunks
    ):
        response.append(llm_response)
        yield json.dumps({
            "text": llm_response,
            "metadata": None
        }) + "\n"
```

5.4 Uživatelské rozhraní

Uživatelské rozhraní slouží jako hlavní přístupový bod pro uživatele systému. Je navrženo jako webová aplikace s cílem umožnit dotazování nad vlastní znalostní bází.

Webové rozhraní je vytvořeno a nastylováno pomocí frameworků Vue.js a Tailwind CSS. Komunikace s backendem probíhá prostřednictvím REST API. Aplikace podporuje streamování odpovědi od jazykového modelu po jednotlivých blocích, což zlepšuje uživatelský dojem a snižuje vnímanou dobu odezvy.

Pro demonstraci funkcionality jsou k dispozici dva pohledy. Uživatelský pohled zobrazuje zamýšlené rozhraní s dotazy a odpověďmi, zatímco vývojářský pohled zobrazuje k daným dotazům vyhledané chunky a jejich skóre, které tvořily kontext pro odpověď. Dále je možné vypnout nebo zapnout chatovou historii a přepínat role uživatele pro filtrování dle přístupových práv.



Obrázek 5.2: Webová aplikace: nahoře je pohled uživatele a dole pohled vývojáře

Kapitola 6

Testování a evaluace

V kapitole 3.3 byla diskutována teoretická východiska evaluace RAG systémů, která poukazují na komplexnost samotného hodnocení. Cílem této kapitoly je experimentálně ověřit výkonnost implementovaného systému na konkrétním datovém vzorku s využitím zvolených evaluačních metod. Nejprve je popsán použitý dataset. Evaluace poté probíhá na třech úrovních:

1. Testování vyhledávacího modulu, kde je ověřena schopnost systému vyhledávat dokumenty, přesněji jejich relevantní části, na základě dotazu.
2. Komplexní evaluace celé RAG pipeline pomocí frameworku RAGAs.
3. Výkonnostní testování systému.

Při tvorbě testovacích sad byl použit LLM, aby nedošlo k ovlivnění lidským faktorem. Tento přístup je časově efektivní a zároveň snižuje riziko neúmyslného zkreslení, které by mohlo vzniknout při ruční formulaci dotazů.

Výsledky přesto nelze považovat za absolutní, jelikož mimo kvalitu systému závisí výsledky na datech, na kterých je systém testován, a také na samotných dotazech. Jejich smyslem je tak poskytnout indikativní pohled na chování systému v realistickém scénáři.

6.1 Dataset

Pro účely testování a evaluace byl využit dataset *Plain text Wikipedia (SimpleEnglish)*¹. Jedná se o neanotovaný (nesupervised) textový korpus obsahující všechny články ze Simple English Wikipedie, varianty Wikipedie psané ve zjednodušené angličtině. Extrahované články jsou předzpracovány na čistý text zbavený formátování, odkazů, šablon a jiných prvků a tvoří tak předpovídatelný formát.

Rozsah celého datasetu činí 249 396 článků rozdělených do 171 souborů o velikosti do 1MB. Tento dataset byl zvolen především díky obsahové rozmanitosti témat, jazykové jednoduchosti a také uniformní povaze dat usnadňující zpracování. To vytváří konzistentní datovou bázi pro testování a evaluaci.

Pro testování a evaluaci byla použita podmnožina tohoto datasetu činící 30 dokumentů obsahujících celkem přes 17 500 článků (po zpracování přes 29 500 chunků).

¹<https://www.kaggle.com/datasets/ffatty/plain-text-wikipedia-simpleenglish>

6.2 Testování modulu vyhledávání

Cílem této části je otestovat účinnost vyhledávacího modulu systému na schopnosti vrátit chunk obsahující odpověď na daný dotaz.

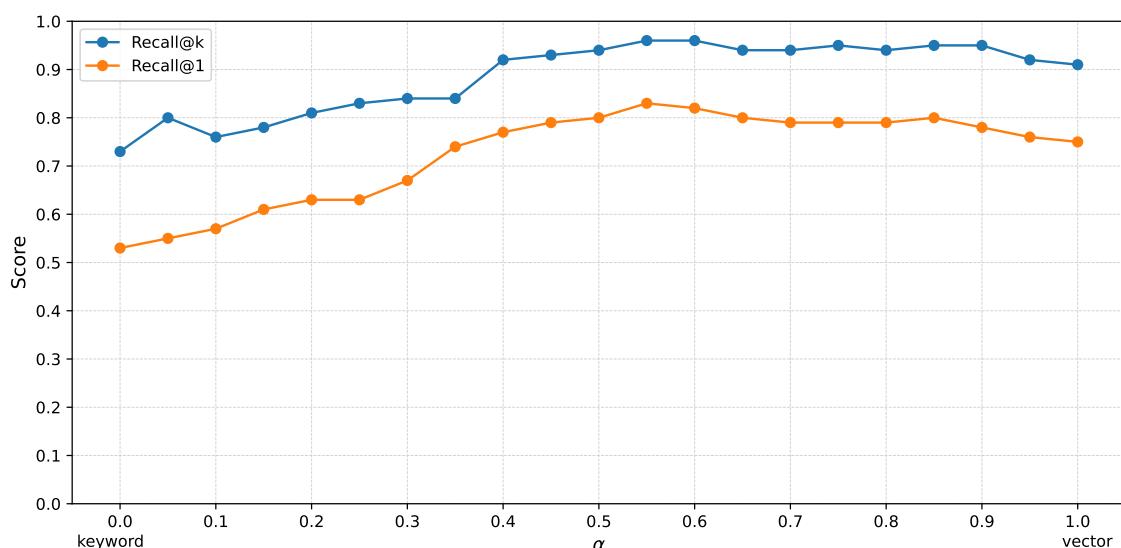
Pro každý testovací případ bylo zvoleno ID chunku ze zpracovaného datasetu a z jeho textu byl odvozen dotaz. Tyto dotazy byly generovány pomocí chatbota ChatGPT, který využíval jazykový model GPT-4-turbo a byl instruován, aby vygeneroval dotaz, který směřuje k nějaké informaci v textu sémanticky nebo pomocí klíčových slov tak, aby se odpověď na dotaz nacházela v daném chunku. Testovací soubor formátu *JSON Lines* tak obsahuje 100 párů `{query, expected_chunk_id}`.

Testování probíhá formou vyhodnocení úspěšnosti vyhledávání při výskytu chunku na první pozici jako nejrelevantnější chunk (Recall@1) a výskytu ve finálním kontextu (Recall@k). Obě tyto metriky jsou vyčísleny od 0 do 1, kde 1 představuje 100% úspěšnost. Jelikož jsou dotazy mířeny na konkrétní záznamy, výchozí metoda tvoření kontextu je nastavena na parametr `auto_limit=1` (viz 5.2.3) bez rerankingu.

6.2.1 Parametr α

Poměr mezi váhami vektorového a fulltextového vyhledávání v hybridním přístupu řídí parametr α . V grafu 6.1 jsou zobrazeny výsledky testu pro celý rozsah parametru k vyhodnocení: $\alpha \in \langle 0, 1 \rangle$ po krocích 0,05.

Nejlepších hodnot metrik Recall@1 a Recall@k bylo dosaženo při $\alpha = 0,55$. To odpovídá vyváženému poměru mezi oběma typy vyhledávání mírně inklinujícímu k vektorovému, které se projevilo v testování jako silnější. Výrazné omezení jednoho z přístupů k vyhledávání však vede ke zhoršení výkonu, což potvrzuje předpokládaný výsledek a smysl použití hybridního vyhledávání.



Obrázek 6.1: Porovnání Recall@1 a Recall@k pro různé hodnoty parametru α při hybridním vyhledávání bez rerankingu, kde kontext tvoří první shluk chunků

6.2.2 Strategie vyhledávání a vliv rerankingu

Reranking má vliv nejen na pořadí chunků ve výsledku, ale také na výslednou velikost kontextu, který je následně předáván jazykovému modelu. To ilustrují výsledky v tabulce 6.1. Test probíhal hybridním vyhledáváním ($\alpha = 0,55$) se zapnutým/vypnutým rerankingem pro hodnoty $\text{auto_limit} \in \{1; 2; 3\}$ pro prozkoumání širšího kontextu a při rerankingu také pro hodnoty $\text{cutoff} \in \{0,3; 0,5; 0,7\}$ filtračního prahu (parametr filtrační funkce modulu `Reranker` – kap. 5.2.5).

auto_limit	cutoff	Recall@1	Recall@k	Pořadí	Kontext
1	off	0,83	0,96	1,18	2,57
	0,3	0,86	0,93	1,08	1,53
	0,5	0,86	0,93	1,08	1,38
	0,7	0,86	0,91	1,05	1,23
2	off	0,83	0,97	1,21	8,80
	0,3	0,84	0,96	1,15	3,86
	0,5	0,84	0,96	1,15	2,45
	0,7	0,84	0,95	1,13	1,59
3	off	0,83	0,98	1,27	17,58
	0,3	0,82	0,97	1,18	7,52
	0,5	0,82	0,97	1,18	3,67
	0,7	0,82	0,96	1,17	1,90

Tabulka 6.1: Porovnání Recall@1 a Recall@k, průměrného pořadí referenčního chunku a průměrného počtu chunků v kontextu. Hodnota `off` sloupce `cutoff` značí vypnutý reranking.

Výsledky ukazují, že samotné hybridní vyhledávání dosahuje dobrých výsledků, jelikož v 96% případů byl chunk poskytující odpověď zahrnut v kontextu. Při testování se reranking neprojevil jako výrazné zlepšení výsledků. Z hodnot průměrného pořadí referenčního chunku je však patrné, že reranking dokáže přesněji odhalit sémantické detaily a většinově to tak vedlo k mírnému zlepšení Recall@1. Menší velikost kontextu pak má smysl za účelem redukce šumu, ale může také vést ke snížení nákladů na provozování systému. Vyšší míra filtrace však potenciálně vedla k horším hodnotám Recall@k. Dochází tak ke kompromisu mezi velikostí a přesností kontextu.

6.2.3 Vliv rewritingu

Rewriting potenciálně zlepšuje vyhledávání přeformulováním dotazu, v tomto testu však při použití modulu `Rewriter` došlo ke snížení hodnot metriky Recall@1 z 0,83 na 0,79 a metriky Recall@k z 0,96 na 0,94 (při výchozí konfiguraci a parametru $\alpha = 0,55$). Nepotvrdil se tedy pozitivní vliv, avšak dotazy v testovací sadě jsou již vhodně a napřímo formulované jazykovým modelem.

Proto byla celá testovací sada upravena s využitím ChatGPT přidáním výplňkových slov, překlepů a reformulací dotazů do hovorovější podoby. Například:

- *Why can eating raw food be risky? → Eating food raw... risky why, tho?*

- *What is the wavelength range of red light? → Whatz the wavlenght range of red ligh?*
- *What is colonization and how is it depicted in science fiction → What is colonnization and how is it depiccted in scifi?*

Zde se výrazně potvrdil účinek přepisování dotazů pro potenciální kompenzování lidské chyby, což shrnují výsledky v tab. 6.2.

Testy	Rewriter	Recall@1	Recall@k
původní	vypnut	0,83	0,96
	zapnut	0,79	0,94
upravené	vypnut	0,65	0,81
	zapnut	0,82	0,92

Tabulka 6.2: Porovnání Recall@1 a Recall@k pro původní a upravenou testovací sadu s a bez použití přepisování dotazů

6.3 RAGAs evaluace

V této části je hodnocena výkonnost celého RAG jako end-to-end pipeline a jsou ověřeny některé strategie nastavení systému. Hodnocení bylo provedeno pomocí nástroje RAGAs [11], který umožňuje automatizovanou evaluaci pomocí LLM bez nutnosti lidského hodnocení. Jako *LLM-as-a-Judge* zde slouží model `gpt-4o-mini`. Pro rozšíření sledovaných metrik oproti kapitole 3.3.2, kde je shrnut princip a výpočet základních metrik, bylo též využito referenčních odpovědí.

6.3.1 Hodnotící metriky

Níže je seznam sledovaných metrik a popis, co hodnotí:

- **Context Precision** – zdali jsou vyhledané informace relevantní dotazu a v kontextu se nevyskytuje zbytečný šum.
- **Context Recall** – jestli obsahují vyhledané záznamy informace k poskytnutí správné odpovědi a fáze retrievalu funguje, jak má.
- **Response Relevancy** – na kolik je vygenerovaná odpověď relevantní k dotazu a měří tak její užitečnost.
- **Faithfulness** – do jaké míry je odpověď podložena vyhledaným kontextem, což kontroluje, zdali nedochází k halucinacím.
- **Factual Correctness (F1 score)** – jak dobře je odpověď fakticky správná porovnáním s referenční odpovědí.



Obrázek 6.2: Metriky hodnocení

Hodnoty těchto metrik se pohybují v intervalu $\langle 0, 1 \rangle$, kde vyšší hodnota znamená lepší výsledek. K těmto metrikám pak byl přidán ukazatel průměrné velikosti vstupního dotazu jazykového modelu:

- **Average input tokens** – průměrný počet tokenů v promptu, čímž je sledována velikost poskytnutého kontextu.

6.3.2 Příprava testovací sady

Modul `TestsetGenerator`, který je součástí frameworku RAGAs, využívá embedding model `all-mpnet-base-v2` a jazykový model `gpt-4o-mini` pro generování testovacích případů ve formě dvojice: dotaz a tzv. *golden answer* (referenční odpověď). Pro každý testovací případ byla poskytnuta část náhodně vybraného dokumentu z datasetu. Získané testovací případy se dělí na:

- **single-hop** – dotaz je zodpověditelný na základě jediné věty nebo jednoho konkrétního úseku textu,
- **multi-hop** – dotaz vyžaduje zkombinování více informací z různých částí dokumentu.

Tímto způsobem bylo připraveno 50 single-hop testů a 30 multi-hop testů. Každý testovací případ byl nakonec obohacen o vyhledaný kontext a vygenerovanou odpověď, které byly získány systémem s testovanou strategií.

6.3.3 Strategie vyhledávání a vliv rerankingu

Cílem tohoto měření (na single-hop testovacích případech) je ověřit, do jaké míry může modul `Reranker` zvýšit výkon systému při end-to-end evaluaci. Výchozí konfigurace spoléhá na hybridní vyhledávání prvního shluku zaznamů (`auto_limit=1`), použití rerankingu však umožňuje nejprve načíst širší množinu relevantních informací (`auto_limit=3`) a následně ji upřesnit pomocí hodnocení relevance. Testovány byly opět tři hodnoty filtračního prahu $\{0,3, 0,5, 0,7\}$. V tabulce 6.3 jsou naměřené hodnoty metrik pro jednotlivé prahy.

Metrika	off	0,3	0,5	0,7
Context Precision	0,803	0,837	0,844	0,833
Context Recall	0,798	0,882	0,885	0,824
Response Relevancy	0,836	0,890	0,868	0,832
Faithfulness	0,799	0,885	0,880	0,777
Factual Correctness (F1)	0,518	0,570	0,587	0,519
Avg input tokens	1063	2585	1637	973

Tabulka 6.3: Porovnání výsledků při různých hodnotách filtračního prahu rerankingu ve srovnání s vypnutým rerankingem

Měření ukazuje výrazný pozitivní vliv rerankingu ve všech metrikách. Zatímco hodnota prahu 0,7 se projevila jako příliš přísná, retrieval se nejvíce zlepšil při `cutoff` = 0,5, kde metrika Context Recall stoupla o 0,087. Hodnocení věrnosti a relevance odpovědi bylo nejvyšší při `cutoff` = 0,3, avšak za cenu kontextu většího o 63% než u hodnoty = 0,5.

6.3.4 Velikost chunku a volba embedding modelu

Cílem této evaluace (na single-hop testovacích případech) je zjistit optimální limit velikosti chunků a výběr embedding modelu. Jako možné scénaře byly testovány:

1. model `all-mpnet-base-v2`, limit 384 tokenů (limit modelu)
2. model `all-mpnet-base-v2`, limit 200 tokenů
3. model `all-MiniLM-L6-v2`, limit 256 tokenů (limit modelu)

Oba embedding modely jsou nejlépe hodnocené z knihovny `sentence-transformers`, kde alternativní `all-MiniLM-L6-v2` vykazuje až 5krát nižší latenci. Výsledky měření jsou v tabulce 6.4.

Metrika	1.	2.	3.
Context Precision	0,803	0,827	0,785
Context Recall	0,798	0,686	0,716
Response Relevancy	0,836	0,849	0,777
Faithfulness	0,799	0,781	0,742
Factual Correctness (F1)	0,518	0,571	0,513
Avg input tokens	1063	801	772

Tabulka 6.4: Porovnání výsledků tří scénářů výběru embedding modelu a velikosti chunků (při výchozí strategii vyhledávání bez rerankingu)

Ačkoliv nabízí model `all-MiniLM-L6-v2` výrazně nižší latenci, výsledky evaluace poukazují na nižší kvalitu vyhledávání a výstupu při jeho použití. Rozdíl v limitu velikosti chunku pak nevykazuje jednoznačné výsledky pro daný dataset a testovací sadu.

6.3.5 Evaluace systému

V tab. 6.5 je evaluací ověřena úspěšnost vyhledávání a generování odpovědi pro jednodušší (single-hop) i komplexnější (multi-hop) dotazy. RAG systémy staví na vhodném nastavení parametrů a zvolení optimalizačních technik pro zlepšení výsledků, z nichž klíčové byly ověřeny v předešlém testování. Systém je nastaven tak, jak je popsáno v kapitolách 4 a 5.

Metrika	single-hop	multi-hop
Context Precision	0,882	0,678
Context Recall	0,842	0,733
Response Relevancy	0,886	0,778
Faithfulness	0,843	0,848
Factual Correctness (F1)	0,501	0,483
Avg input tokens	1581	2156

Tabulka 6.5: Výsledky evaluace systému na single-hop a multi-hop testovacích případech

Z výsledků vyplývá, že vyhledávací modul dosahuje dobrých výsledků při vyhledávání relevantních informací k přímému dotazu. Generátor pak poskytuje odpovědi relevantní dotazu a podložené informacemi z kontextu, čímž je redukován vliv halucinací. Tyto metriky mají hodnoty v rozmezí od 0,84 do 0,89.

F1 skóre dosahuje hodnoty pouze kolem 0,5, což může naznačovat nižší faktickou správnost generovaných odpovědí. Vzhledem k tomu, že formát a rozsah odpovědi není jednoznačně definován, generátor může odpovědět správně jinou, ale stejně relevantní formulací. Tato metrika se tak u otevřených otázek projevuje jako mírně zavádějící u nekterých případů. Například:

Dotaz: *What are the countries in Scandinavia that are not republics?*

Odpověď: *The countries in Scandinavia that are not republics are Denmark, Norway, and Sweden. These countries are monarchies. [File: wiki_06.txt, Section: Scandinavia]*

Reference: *Countries in Scandinavia that are not republics include those with a king or other monarch and free elections, which are classified as constitutional monarchies.*

Z hlediska faktického obsahu jsou obě odpovědi správné, avšak se liší natolik, že metrika F1 jim přiřadí skóre 0. V jiném případě byla hodnota 0,5 dosaženého skóre, přestože odpověď obsahovala klíčovou informaci, pouze s jinou formulací:

Dotaz: *What historical execution method was used in Scotland before the guillotine became prominent in France?*

Odpověď: *Before the guillotine became prominent in France, Scotland used a machine called the „Scottish Maiden“ for executions by decapitation. [File: wiki_12.txt, Section: Guillotine]*

Reference: *In Scotland, a machine called the 'Scottish Maiden' was used for executions before the guillotine became prominent in France.*

6.3.6 Škálovatelnost systému

Pro ověření škálovatelnosti systému byla provedena celková evaluace jako v předchozí podkapitole, avšak s kompletním datasetem. Ten po zpracování obsahuje téměř 270 000 chunků, což je přibližně 9,3krát více než v dosavadním testování.

Metrika	single-hop	multi-hop
Context Precision	0,875	0,752
Context Recall	0,792	0,708
Response Relevancy	0,888	0,718
Faithfulness	0,856	0,810
Factual Correctness (F1)	0,545	0,410
Avg input tokens	1337	2447

Tabulka 6.6: Výsledky evaluace systému na single-hop a multi-hop testovacích případech s kompletním datasetem

Z tab. 6.6 je patrné, že je systém snadno škálovatelný, jelikož nedošlo k výrazným změnám jak v hodnotách metrik, tak ve velikosti kontextu v tokenech. Průměrná absolutní odchylnka mezi naměřenými hodnotami metrik vychází na 0,023 pro single-hop a 0,054 pro multi-hop testovací případy, což není dostatečná změna na překonání různorodosti hodnocení LLM.

6.4 Výkonnostní testování systému

Pro určení rychlosti odezvy systému proběhlo měření latence jednotlivých procesů. Testování probíhalo lokálně na vývojovém zařízení Apple MacBook Air s čipem M1 (8jádrový CPU, architektura ARM64), operační pamětí 8 GB a operačním systémem macOS 15.4.1. Docker daemon byl omezen na 4GB RAM a server, webová aplikace i vektorová databáze byly spuštěny virtuálně v jednotlivých docker kontejnerech a bez GPU akcelerace.

6.4.1 Nahrání datasetu

Zpracování celého datasetu z Simple English Wikipedie (z lokálního úložiště) na bez mála 270 000 chunků, jejich vektorizace a uložení po dávkách do vektorové databáze trvalo systému celkem 2 hodiny a 42 minut. Jednotlivé fáze probíhaly v následujících časech:

- zpracování dokumentů: 7 minut,
- generování embeddingů: 2 hodiny a 28 minut,
- zápis vektorů do databáze: 7 minut.

Embedding model tak představuje hlavní časové zatížení při fázi zpracování a nahrávání dokumentů do databáze.

6.4.2 Dotazování

Tab. 6.7 ukazuje průměrné hodnoty naměřené při dotazování systému prostřednictvím uživatelského rozhraní po deseti dotazech. Fáze odpověď udává dobu od poskytnutí dotazu až po počátek generování odpovědi, jelikož je pak zpráva generována postupně a tím je zajištěna interakce s uživatelem. Z měření lze pozorovat, že vyhledávání ve vektorové databázi mezi nižšími stovkami tisíc záznamů je velice rychlé, zatímco reranking způsobuje největší prodlevu. Pro zlepšení uživatelské zkušenosti by namísto čekání na generování odpovědi (průměrně 7,33 s) mohla být přidána informace o aktuálním stavu vyhledávání.

Fáze	Odezva (s)
Rewriting	1,25
Připojení k databázi	2,85
Vyhledávání	0,82
Reranking	2,53
Odpověď	7,33

Tabulka 6.7: Průměrné trvání jednotlivých fází systému. Fáze *odpověď* označuje dobu mezi posláním dotazu a počátkem generování odpovědi, kterou uživatel čeká.

Kapitola 7

Závěr

Tato bakalářská práce shrnuje princip vektorových databází a prezentuje návrh a implementaci systému pro zpracování a vyhledávání dokumentů pomocí architektury Retrieval-Augmented Generation s využitím vektorové databáze a jazykových modelů. Systém umožňuje synchronizaci s cloudovým úložištěm, zpracování dokumentů různých formátů na chunky a jejich uložení do vektorové databáze, která slouží jako báze znalostí. Následně lze skrze uživatelské rozhraní podávat dotazy v přirozeném jazyce, které jsou zpracovány a využity k získání relevantních pasáží dokumentů pomocí kombinace vektorového a full-textového vyhledávání. Vyhledané informace prochází rerankingem a slouží jako kontext pro odpověď. Ta je podložena zdroji a generována pro uživatele opět v přirozeném jazyce. Systém podporuje i filtrování metadat a chatovou historii pro navazující otázky.

Přínos práce spočívá v realizaci modulárního systému, který propojuje moderní technologie z oblasti zpracování přirozeného jazyka a práce s nestrukturovanými daty do funkčního řešení, které lze využít pro vyhledávání ve velkých datových kolekcích. Implementovaný systém byl navíc otestován na vyhledávání a následně podroben evaluaci. Výsledky byly využity k ladění parametrů, ale také k ověření celkové účinnosti navrženého řešení a škálovatelnosti systému.

Pro účely této práce byl systém vyvinut a testován lokálně. Budoucí rozvoj tohoto projektu tak spočívá v přípravě systému pro nasazení v produkčních podmírkách pro jednotlivce nebo celé společnosti. Filtrování metadat, např. na základě přístupových práv uživatelů, je v systému již připraveno, avšak je třeba ho zavést pro konkrétní užití při přístupu více uživatelů k jedné databázi. Další možnosti rozvoje zahrnují rozšíření podpory pro zpracování více formátů dokumentů (např. PPTX, XLSX) a důkladnější práci s PDF soubory, dále zpracování multimodálních dat nebo podporu vícejazyčnosti.

Tato práce byla prezentována dne 6. 5. 2025 na studentské konferenci Excel@FIT a získala ocenění za výjimečnou práci od firem Red Hat a Gen Digital. Plakát vytvořený pro tuto událost je součástí přílohy A.

Literatura

- [1] AHMED, S. What is Reranking in Retrieval-Augmented Generation (RAG)? *Medium* online. 2. listopadu 2024. Dostupné z: <https://medium.com/@sahin.samia/what-is-reranking-in-retrieval-augmented-generation-rag-ee3dd93540ee>. [cit. 2025-04-28].
- [2] ALLEN, C. a HOSPEDALES, T. Analogies Explained: Towards Understanding Word Embeddings. *Proceedings of the 36th International Conference on Machine Learning*. Long Beach, California: PMLR, Květen 2019, sv. 97.
- [3] BERNHARDSSON, E. Annoy: Approximate Nearest Neighbors Oh Yeah. *GitHub repository* online. 2. května 2015. Dostupné z: <https://github.com/spotify/annoy>. [cit. 2025-04-28].
- [4] BLUMBERG, R. a ATRE, S. The Problem with Unstructured Data. *DM Review*, Únor 2003, sv. 13, s. 42–46.
- [5] BÜTTCHER, S.; CLARKE, C. L. A. a CORMACK, G. V. *Information Retrieval: Implementing and Evaluating Search Engines*. 1. vyd. Cambridge, Massachusetts: MIT Press, 2010. ISBN 978-0-262-02651-2.
- [6] CAMACHO COLLADOS, J. a PILEHVAR, M. T. From Word to Sense Embeddings: A Survey on Vector Representations of Meaning. *Journal of Artificial Intelligence Research*. El Segundo, CA, USA: AI Access Foundation, Září 2018, sv. 63, č. 1, s. 743–788. ISSN 1076-9757.
- [7] CHAUDHARY, A. Semi-Structured Data: Definition and Examples. *Datamation* online. 30. listopadu 2023. Dostupné z: <https://www.datamation.com/big-data/semi-structured-data/>. [cit. 2025-04-28].
- [8] CHEN, J. et al. Benchmarking Large Language Models in Retrieval-Augmented Generation. *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI Press, Únor 2024, č. 1980, s. 17754–17762. AAAI'24/IAAI'24/EAAI'24.
- [9] DEVLIN, J. et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*. arXiv, Říjen 2018, abs/1810.04805.
- [10] ES, S. et al. RAGAS: Automated Evaluation of Retrieval Augmented Generation. *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*. St. Julians, Malta: Association for Computational Linguistics, Březen 2024, 2309.15217, s. 150–158.

- [11] EXPLODINGGRADIENTS. *Ragas Documentation* online. Documentation. 2025. Dostupné z: <https://docs.ragas.io/en/stable/>. [cit. 2025-03-25].
- [12] FASTAPI. *FastAPI – Custom Response - StreamingResponse* online. Documentation. 2025. Dostupné z: <https://fastapi.tiangolo.com/advanced/custom-response/#streamingresponse>. [cit. 2025-03-25].
- [13] GOOGLE. *Google Drive API overview* online. Documentation. 2025. Dostupné z: <https://developers.google.com/workspace/drive/api/guides/about-sdk>. [cit. 2025-03-25].
- [14] GUPTA, S.; RANJAN, R. a SINGH, S. N. A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions. arXiv, Říjen 2024, abs/2410.12837.
- [15] HAN, Y.; LIU, C. a WANG, P. A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge. *CoRR*. arXiv, Říjen 2023, abs/2310.11703.
- [16] HUGGINGFACE. *AutoTokenizer* online. Documentation. 2025. Dostupné z: https://huggingface.co/docs/transformers/v4.49.0/en/model_doc/auto#transformers.AutoTokenizer. [cit. 2025-03-25].
- [17] JÉGOU, H.; DOUZE, M. a SCHMID, C. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. IEEE, Leden 2011, sv. 33, č. 1, s. 117–128. ISSN 0162-8828.
- [18] KARPUKHIN, V. et al. Dense Passage Retrieval for Open-Domain Question Answering. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Listopad 2020, 2004.04906, s. 6769–6781.
- [19] LEWIS, P. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Proceedings of the 34th International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., Duben 2020, č. 793. NIPS ’20.
- [20] LI, H. et al. LLMs-as-Judges: A Comprehensive Survey on LLM-based Evaluation Methods. arXiv, Prosinec 2024, abs/2412.05579.
- [21] MALKOV, Y. A. a YASHUNIN, D. A. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *IEEE Transactions on Pattern Analysis & Machine Intelligence*. Los Alamitos, CA, USA: IEEE Computer Society, Duben 2020, sv. 42, č. 04, s. 824–836. ISSN 1939-3539.
- [22] MONIGATTI, L. Retrieval-Augmented Generation (RAG): From Theory to LangChain Implementation. *Towards Data Science* online. 14. listopadu 2023. Dostupné z: <https://towardsdatascience.com/retrieval-augmented-generation-rag-from-theory-to-langchain-implementation-4e9bd5f6a4f2>. [cit. 2025-04-28].
- [23] MUREL, J. a NOBLE, J. What is an encoder-decoder model? *IBM* online. 1. října 2024. Dostupné z: <https://www.ibm.com/think/topics/encoder-decoder-model>. [cit. 2025-04-28].

- [24] NGROK. *Ngrok Documentation* online. Documentation. 2025. Dostupné z: <https://ngrok.com/docs>. [cit. 2025-03-25].
- [25] NLTK. *Nltk.tokenize.sent_tokenize* online. Documentation. 2025. Dostupné z: https://www.nltk.org/api/nltk.tokenize.sent_tokenize.html. [cit. 2025-03-25].
- [26] OPENAI. *Embeddings* online. Documentation. 2025. Dostupné z: <https://platform.openai.com/docs/guides/embeddings>. [cit. 2025-03-25].
- [27] OPENAI. *Text generation and prompting* online. Documentation. 2025. Dostupné z: <https://platform.openai.com/docs/guides/text?api-mode=chat>. [cit. 2025-03-25].
- [28] PAN, J. J.; WANG, J. a LI, G. Survey of Vector Database Management Systems. *The VLDB Journal*, Září 2024, sv. 33, č. 5, s. 1591–1615. ISSN 0949-877X.
- [29] PENG, D.; GUI, Z. a WU, H. Interpreting the Curse of Dimensionality from Distance Concentration and Manifold Effect. *ArXiv*. arXiv, Leden 2024, abs/2401.00422.
- [30] PENNINGTON, J.; SOCHER, R. a MANNING, C. D. GloVe: Global Vectors for Word Representation. The Standford Natural Language Processing Group, Srpen 2014.
- [31] REIMERS, N. a GUREVYCH, I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Listopad 2019, 1908.10084, s. 3982–3992.
- [32] ROBERTSON, S. a ZARAGOZA, H. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval*. Hanover, MA, USA: Now Publishers Inc., Duben 2009, sv. 3, č. 4, s. 333–389. ISSN 1554-0669.
- [33] TAIPALUS, T. Vector database management systems: Fundamental concepts, use-cases, and current challenges. *Cognitive Systems Research*. Elsevier BV, Červen 2024, sv. 85, s. 101216. ISSN 1389-0417.
- [34] UKPLAB. *Cross Encoder – Pretrained Models* online. Documentation. 2025. Dostupné z: https://sbert.net/docs/cross_encoder/pretrained_models.html. [cit. 2025-03-25].
- [35] UKPLAB. *Sentence Transformer – Pretrained Models* online. Documentation. 2025. Dostupné z: https://sbert.net/docs/sentence_transformer/pretrained_models.html. [cit. 2025-03-25].
- [36] UNSTRUCTURED. *Unstructured Open Source* online. Documentation. 2025. Dostupné z: <https://docs.unstructured.io/open-source/introduction/overview>. [cit. 2025-03-25].
- [37] WEAVIATE. *Weaviate Docs* online. Documentation. 2025. Dostupné z: <https://weaviate.io/developers/weaviate>. [cit. 2025-03-25].
- [38] WEISSTEIN, E. Vector. *MathWorld – A Wolfram Web Resource* online. Dostupné z: <https://mathworld.wolfram.com/Vector.html>. [cit. 2025-04-28].

- [39] YENDURI, G. et al. Generative Pre-trained Transformer: A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions. *IEEE Access*, Leden 2024, PP, 2305.10435, s. 1–1.
- [40] YU, H. et al. Evaluation of Retrieval-Augmented Generation: A Survey. *Big Data*. Springer Nature Singapore, Červenec 2024, 2405.07437, s. 102–120. ISSN 1865-0937.
- [41] ZHANG, B.; LIU, X. a LANG, B. Fast Graph Similarity Search via Locality Sensitive Hashing. *Advances in Multimedia Information Processing – PCM 2015*. Cham: Springer International Publishing, Listopad 2015, s. 623–633. ISSN 978-3-319-24075-6.
- [42] ZHAO, P. et al. Retrieval-Augmented Generation for AI-Generated Content: A Survey. arXiv, Červen 2024, abs/2402.19473.
- [43] ZILLIZ. Introduction to Unstructured Data. *Zilliz* online. 29. září 2022. Dostupné z: <https://zilliz.com/learn/introduction-to-unstructured-data>. [cit. 2025-04-28].

Příloha A

Plakát Excel@FIT

