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 JAZYKOVÉHO MODELU

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

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ADAM VALÍK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

163475

Ústav:	Ústav informačních systémů (UIFS)
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

Abstract

Klíčová slova

Keywords

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. Další informace mi poskytli... Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Adam Valík
27. března 2025

Poděkování

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant apod.).

Obsah

Kapitola 1

Úvod

Především za poslední 2 roky lze zaznamenat napříč světem vzestup zájmu o umělou inteligenci [15]. Vzniká například mnoho velkých jazykových modelů podporující generativní AI ve formě chatbotů, kteří jsou schopni odpovídat na dotazy na základě dohledatelných dat. Tradiční databázové systémy, jako relační a objektově-relační databáze, jsou efektivní při zpracování strukturovaných dat. Vyhledávání v těchto databázích probíhá nejčastěji konkrétními SQL dotazy. Je to efektivní způsob uchovávání a vyhledávání dat například pro informační systémy. Revoluce AI však přináší práci s daty, která není efektivně nebo vůbec proveditelná těmito databázovými systémy, a proto s ní přichází rozvoj vektorových databází. Ty totiž dokáží pracovat i s nestrukturovanými daty, které jsou reprezentovány pomocí vektorů (matematickou reprezentací n-tice čísel). Svou formou dokáží zachytit význam dat, jako jsou slova nebo celé texty, obrázky a zvuková data. Právě proto patří vektorové databáze k moderním přístupům NLP [28, 26].

Spojením metod získávání informací na základě významu pomocí vektorových databází a předtrénovaného jazykového modelu můžeme vytvořit systém, kterému se říká Retrieval-Augmented Generation (RAG), neboli generování rozšířené o získávání informací. Vývoj vlastního LLM je velmi náročný, a proto tento přístup využívá dostupných jazykových modelů. Obohacení dotazování jazykového modelu staví na vlastních zdrojích informací získaných z vektorové databáze, kde je uložen velký objem dat dokumentů. Z navrácených dat se pak formuje dotaz pro jazykový model společně s dotazem uživatele, který produkuje uživateli odpověď. Tento přístup umožňuje například vyhledávání nad velkým množstvím interních dokumentů firmy. Možnosti využití můžeme nalézt i v akademickém prostředí při vyhledávání v odborné literatuře. Důležité je zde správné předzpracování dat, jelikož typicky zdroje tvoří velké množství dokumentů různých typů souborů. Vektorové databáze mohou vyhledávat texty, které mají podobný význam jako dotaz, místo aby se spoléhaly na přesnou shodu klíčových slov, které v tomto případě nemusí být efektivní. Data je však třeba vhodně rozkouskovat, přidat k nim metadata a pomocí embedding modelu vytvořit vektorovou reprezentaci, která je uložena do vektorové databáze. 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 takového komplexního systému, který nabídne uživatelům získávat informace z velkého množství dokumentů. Je toho dosaženo kombinací moderních přístupů NLP, jakou jsou embedding modely, práce s nestrukturovanými daty, vektorové databáze a kombinovaný přístup k získávání informací pomocí full-textového a

vektorového vyhledávání pro co nejlepší výsledky. Data si uchovávají metadata k filtrování a přímým odkazům na původní zdroje. Dále je využit jazykový model pro navrácení lidsky formulované odpovědi.

Kapitola 2

Vektorové databáze

"Vektorová databáze je typ databáze, která ukládá data jako vysokorozměrné vektory, což jsou matematické reprezentace rysů nebo atributů." [18]. Vektorové databáze jsou relativně moderní technologií, která vznikla především jako řešení na zpracování a vyhledávání nestrukturovaných dat. Již v roce 1998 bylo odhadováno, že více než 85% dat neexistuje ve strukturované formě, a tedy je nelze uložit do relačních databází [6]. Před zhruba deseti lety začaly s vývojem zpracování přirozeného jazyka a hlubokého učení vznikat trénované modely. Tyto modely dokáží převést nestrukturovaná data do vektorové reprezentace, která zachovává jejich význam a vlastnosti. Aby však byla existovala možnost efektivně s těmito vektory pracovat, musela vzniknout nová specializovaná databáze. V tom důsledku začal během posledních let vývoj tvorby vektorových databází, kde uložení velkého množství vektorů je jen prvním krokem. Síla těchto systémů plyne z efektivního vyhledávání na základě sémantické podobnosti, zatímco relační databáze provádějí vyhledávání skrze sloupce a porovnávání hodnot. To je klíčové pro moderní aplikace, jako jsou doporučovací systémy, chatboti nebo vyhledávače obrázků. V následujících podkapitolách je detailněji popsáno, jak vektorové databáze zpracovávají nestrukturovaná data, jak funguje vektorová reprezentace dat a jak se provádí indexace pro efektivní 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. V jejich kontextu spadají do datového typu BLOB (Binary Large Object), přes které nelze snadno vyhledávat.

Příklady nestrukturovaných dat:

- **Data vytvořená člověkem**: E-maily, textové dokumenty, poznámky, obrázky, audio nebo video nahrávky.
- **Data vytvořená strojem**: Údaje ze senzorů, data počítačového vidění, webová a aplikační data, logy, data z IoT zařízení [47].

Mezi strukturovanými a nestrukturovanými daty existuje mezikrok, který se nazývá polostrukturovaná data. Vznikají přidáním metadat nebo významových tagů, čímž je dodáno nestrukturovaným datům jistá forma organizace nebo hierarchie. Například k obrázkům,

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

Nestrukturovaná data jsou nejen ukládána, ale procesem vektorizace je jim přidělena sémantika [40]. 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 uspořádané n-tici čísel, kde n udává dimenzi vektoru [**?**]. S vektory lze provádět operace vektorové aritmetiky, kde nejdůležitější roli hrají Euklidovská vzdálenost a kosinová podobnost.

V kontextu vektorových databází jsou vektory tvořeny transformací dat na matematickou reprezentaci pomocí modelů strojového učení, které se nazývají embedding modely [18]. Vektor si tedy lze představit jako soubor čísel, kde každé z nich reprezentuje číselné vyjádření určité vlastnosti dat. Mějme 2D prostor, pakliže jednotlivým složkám dvourozměrného vektoru přidělíme nějaký rys, můžeme ohodnotit např. slova označující ovoce, kde první hodnota bude sladkost a druhá hodnota kyselost (na škále 1-10). Vzdálenost mezi vektory ovoce podobné chuti pak bude malá.

Komplexní data však vyžadují mnohem více popisů atributů, a tak podobně jako nestrukturovaná data, vysocedimenziální 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 [40]. Počet dimenzí 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 vysocedimenzionální vektorovou reprezentaci, která zachycuje smysluplné vztahy a vzorce*"[40]. 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 [3]: 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*.

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.

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. Byly navrženy 2 modely Word2vec:

- CBOW (Continuous Bag-Of-Words): Zaměřuje se na předpovídání aktuálního slova na základě okolního kontextu.
- Skip-gram: Na rozdíl od CBOW se zaměřuje na předpovídání kontextu pro dané slovo [8].

Dalším významnou architekturou je GloVe [32].
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 *počítač* budou blízko sebe, ačkoliv spolu sémanticky nesouvisí, jelikož se budou obě blížit slovu *myš* [8].

Kontextové modely

Moderní kontextové embedding modely, jako jsou BERT nebo GPT, patří mezi velké jazykové modely (LLM). Mimo 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, který byl primárně navržen pro generování textu. Predikuje další slovo ve větě na základě předchozích slov a kontext slova tak dokáže zachytit tím, že čte větu jednosměrně zleva doprava. Mimo generování textu však dokáže díky porozumění kontextu vytvářet vektorové repezentace textu [43].

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 [12].

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 [35]. 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 především na efektivní správu vysokorozměrných vektorových dat. Toto nezahrnuje nízkorozměrná vektorová data (jako například 2D souřadnice) s nimiž práce je podstatně jednodušší a nevyžaduje dále uvedené optimalizační techniky. VDBMS (z anglického Vector Database Management System) většinou podporuje vyhledávání na základě podobnosti skrze indexování, které je nevyhnutelné pro rychlé vyhledávání ve vektorovém prostoru [40].

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. Ten se stará nejen o uložení dat, ale i efektivní navracení relevantních informací a často si ukládá k samotnému vektoru i nějaký identifikátor nebo metadata, která mohou sloužit k filtrování nebo k poskytování personalizovaných doporučení [40].

2.5 Indexace a vyhledávání

Po vektorizaci a uložení dat je třeba nad nimi 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 [30].

Výběr algoritmu k indexaci záleží na požadavcích. Obecně bývá výpočetně náročný, v ideálním případě ho však není třeba při rozšíření datasetu přepočítávat a umožňuje velmi rychlé vyhledávání. Právě proto se bez něho VDBMS neobejdou [40].

Dotazování pak předchází vyhledávání. Dotazy jsou podávány systému často v přirozeném jazyce, a proto potřebují být taktéž vektorizovány a to stejným modelem jako vektory uložené v databázi. Dotazy si obvykle uchovávají metadata k vyhledávání jako například počet nejbližších vektorů k nalezení. [40]. 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. Tomuto principu se říká vyhledání nejbližších sousedů (anglicky *Nearest Neighbor Seach*, NNS). Je to *"optimalizační problém nalezení bodu v dané množině, který je nejblíže (= nejpodobnější) danému bodu"* [18]. Motivace k zavedení optimalizace indexy a aproximace výsledků je znázorněna na následující metodě.

Naivní metoda

Nejjednodušším přístupem k nalezení nejbližšího vektoru je naivní metoda (v anglické literatuře se též používá pojem *brute-force approach* [18], neboli přistup hrubou silou). Naivní algoritmus počítá vzdálenost mezi každým bodem vektorového prostoru a vektorem dotazu. Tento přístup má absolutní úspěšnost pro navrácení požadovaného počtu opravdových nejbližších bodů, avšak je velice neefektivní. Je-li použita pro výpočet vzdálenosti např. Euklidovská vzdálenost, která se mezi dvěma body p a q v n-rozměrném prostoru vypočítá jako odmocnina součtu čtverců rozdílů odpovídajících souřadnic,

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} [25] \tag{2.1}$$

bude časová náročnost naivního algoritmu *O(NM)*, kde N je počet vektorů v datasetu a M je dimenze vektoru (velikost embeddingu). Při menším počtu nízkodimenzionálních vektorů je tato intuitivní metoda efektivní díky své přesnosti. U vektorových databází však počet dimenzí vektorů roste exponenciálně a dochází k poklesu efektivity mnoha algoritmů strojového učení, což popisuje pojem prokletí dimnezionality (z anglického *curse of dimensionality*) [31]. Navíc počet vektorů uložených v databází velkých projektů může dosahovat několika miliard, a proto VDBMS zavádějí aproximaci a indexační metody, které budou popsány v následujících podkapitolách.

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 řádech 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 k-ANNS, pro jednoduchost bude toto označení vynecháno (jde o nastavení parametru, princip však zůstává stejný). Výběrem nejpoužívanějších implementací ANNS jsou následující metody a jejich algoritmy.

Hashovací metody

Při zpracování vysocedimenzionálních dat staví tyto techniky na transformaci vektorů pomocí hashovacích funkcí na kompaktnější reprezentaci ve formě hashovacích klíčů. Redukují složitost porovnávání dat a urychlují tak vyhledávání i díky specifické sadě hashovacích funkcí, které zajišťují, že není třeba prohledávat celou databází. 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 přídělí vektorům hashovací klíče odpovídající

jejich lokalitě. Dle těchto 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 celá databáze, dochází k výraznému urychlení vyhledávání [39] [45].

Stromové metody

Smyslem stromových metod je zmenšit prohledávaný prostor následováním větví stromu, které s největší pravděpodobností obsahují nejbližší sousedy vektorizovaného dotazu [18].

Approximate Nearest Neighbors Oh Yeah (ANNOY) je algoritmus vyvinutý společností Spotify, původně pro doporučování hudby. Pracuje nejefektivněji ve střednědimenzionálních vektorového prostorech (nižší stovky domenzí), 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 hyperroviny[1]. Hyperrovina je zvolena na základě dvou náhodně vybraných bodů, mezi kterými prochází středová rovina, která je od obou stejně vzdálená. Tento proces pokračuje rekurzivně, dokud podprostor neobsahuje dostatečně 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 těmito kandidáty a vrátí nejbližší sousedy. Počet stromů a hloubka stromu lze optimalizovat ke kompromisu rychlosti a přesnosti [18] [4].

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 [24].

Hierarchical navigable small world (HNSW, v překladu "Hierarchický prohledávatelný malý svět") je vylepšením algoritmu NSW o hierarchickou strukturu a představuje

[1] rovina, která dělí více dimenzionální prostor na dvě části

[2] fenomén malého světa a šest stupňů odloučení ve společnosti je myšlenka, že jsou všichni lidé v průměru 6 sociálních konktaktů od sebe

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 úrovně, blíží se relevantnějším uzlům představující nejbližší sousedy. V realitě pak nejvyšší vrstvu může představovat stovky uzlů z celkového počtu milionů [24].

Kvantizační metody

Kvantizační metody jsou techniky používané ke snížení výpočetní složitosti při práci s vysoce dimenzionálními daty. Kvantizace signálů převádí spojité hodnoty na diskrétní, čímž umožňuje efektivnější ukládání. Kvantizace vektorů pak aproximuje jejich hodnoty na reprezentativní vzorky, čímž dochází k výrazné kompresi dat. Zároveň z této aproximace vzniká kvantizační chyba, která je však přijatelným kompromisem přinášejícím nárůst výkonu a snížení paměťových nároků. Následující metoda je jedna z nejvýznamějších kvantizačních metod v oblasti ANNS [19].

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 podporostory 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 [19].

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 [19].

2.6 Využití vektorových databází

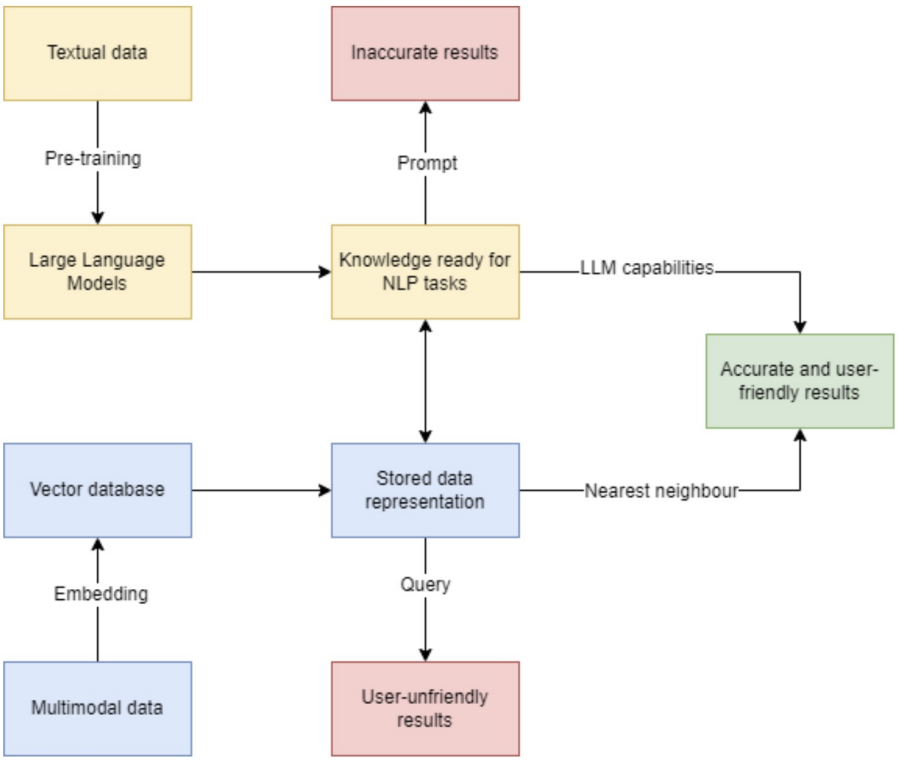
Vektorové databáze jsou moderním nástrojem pro vyhledávání dat na základě sémantické, významové podobnosti. Kterákoliv data zakódovaná embedding modely do vektorů lze pak sémanticky vyhledávat, není třeba se tak omezovat pouze na slova či texty. Ačkoliv je to komplexnejší proces, obrázky nebo videa lze taktéž vektroizovat a vyhledávat tak jim podobné. U obrázků jde pak 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. [40]

Tato schopnost je typicky využívána v doporučovacích systémech, kde na základě získaných dat o uživatele jsou systémy schopny nabízet podobný typ produktů nebo obsahu.

Pro tuto práci je však nejrelevantnější využití vektorových databází ve spojení s velkým jazykovým modelem.

Vektorové databáze a jazykové modely

Databáze a velké jazykové modely se nachází každý v jiné oblasti výzkumu. Specificky vektorové databáze však svými schopnostmi přináší zajímavé využití v kombinaci s jazykovými modely, které ilustruje 4.1. 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. Můžou 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, zvuk) pomocí embeddingů. Tato data jsou pak rychle vyhledávána, sama o sobě však nemusí být uživatelsky přívětivá. Kombinací schopností LLM a vyhledáním nejbližších sousedů tak vznikají fakticky přesné a uživatelky přívětivé výsledky [30].



Obrázek 2.1: Kombinace vektorových databízí s velkým jazykovým modelem vzájemně vyvažuje nedostatky jednotlivých přístupů k navracení informací. Převzato z [30].

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* (generování rozšířené o vyhledávání) a věnuje se mu zbytek této práce.

Kapitola 3

RAG: Retrieval-Augmented Generation

V publikaci z roku 2021 přišli Lewis et. al. [22] se systémem alternativím k čistě parametrizovanému LLM, který by zvýšil přesnost dodávání faktů vyhledáváním informací v externím zdroji. Generování rozšířené o vyhledávání (RAG) je systém, kde "*předtrénovaný parametrizovaný generativní model je vybavený neparametrickou pamětí prostřednictvím univerzálního přístupu jemného doladění.*"[22]

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.

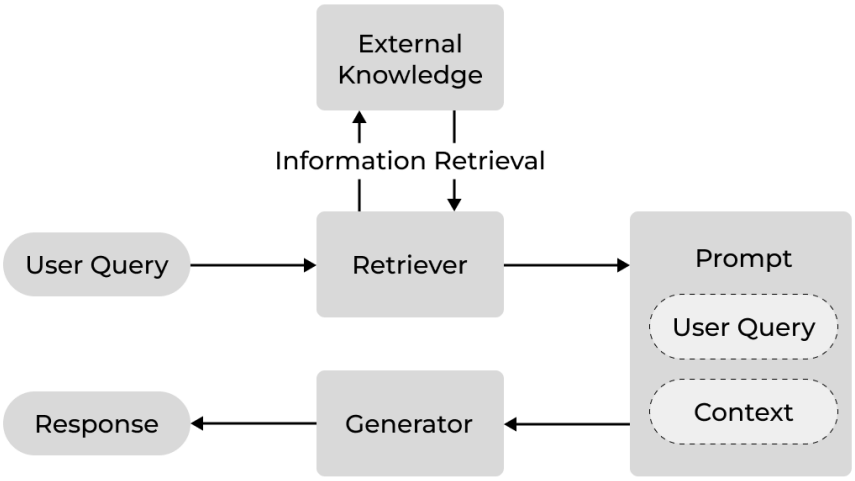
Většina jich plyne z faktu, že LLM uchovávají svoji paměť v parametrech, na kterých byly natrénovány, tedy v **parametrizované paměti**. Jelikož jsou trénovány na velice rozsáhlém množství dat, dokáží pochytit velké množství znalostí bez využití jakékoliv externí paměti. Jsou též skvělé v generování lidsky vypadajícího textu. Fungují však na základě pravděpodobnosti slovních sekvencí, nikoliv ověřených faktů. Jsou též omezené na data, která byla přítomna ve fázi tréninku. [22] [17] To vede k určitým limitům:

- **Halucinace**: LLM můžou produkovat odpověď, která sice působí věrohodně, je však 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 můžou scházet informace ke specifickým oblastem.
- **Citace**: LLM poskytují své odpovědi bez uvedení zdrojů, to snižuje důveryhodnost 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 přidaním vyhledávacího modulu, který staví na neparametrické paměti [22]. 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 můžou pokrývat specifickou doménu nebo privatní data.

3.2 Architektura RAG

Vstupem systému je uživatelův dotaz. Dále RAG kombinuje 2 klíčové komponenty. První je Retriever, je to mechanismus vyhledávání infromací v externím úložišti znalostí. Na základě dotazu vyhledává nejrelevantnější informace, které potenciálně obsahují odpoveď. 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: Základní diagram komponent

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 (2.5) dat 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 [7].
- **BM25** (*Best Match 25*) je 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 full-textové vyhledávání [36].

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 full-textové metody, jsou však výpočetně náročnější a vyžadují embedding modely. Modelem této kategorie pak je **DPR** (*Dense Passage Retriever*) [20], který používá dva nezávislé transformátorové enkodéry BERT [12] (2.3) pro mapování dotazu a dokumentu po částech do hustého vektorového prostoru, kde probíhá sémantické vyhledávání porovnáváním dotazu s pasážemi pomocí skalárního součinu.

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. [27]. Tento modul je však obvykle implementován pomocí velkého jazykového modelu (LLM). Nejčastěji se využívají transformerové modely jako BART [21], T5 [33] nebo GPT [43], které dosahují nejlepších výsledků v generování přirozeného jazyka.

Vylepšení

Zhao P. et al. ve své práci [46] navrhují metody k optimalizaci výsledků pro jednotlivé části RAG, které můžou 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.
    - **Rekurzivní vyhledávání** – vícenásobné vyhledávání pro zpřesnění nalezeného kontextu.

- **Optimalizace chunkování** – optimalizace velikosti chunků (menších částí dokumentu určených) pro efektivnější vyhledávání.
- **Doladění retrieveru** – retriever spoléhá na embedding model, který vytváří vektorové reprezentace dat. Kvalita 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 metadaty** – 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 produkujícího 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. Navíc vyhledávání 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 [44].

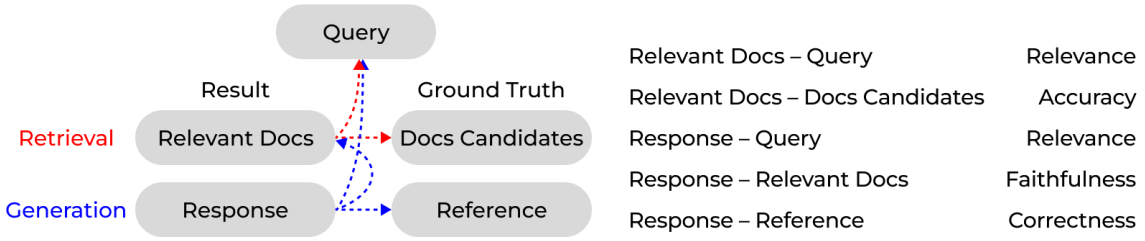
Jednotný proces hodnocení

V průzkumu nástrojů a benchmarků na evaluaci RAG systémů Yu et. al. [44] navrhli jednotný proces hodnocení RAG (*Auepora – A Unified Evaluation Process of RAG*), který pomáhá pochopit složitost srovnávacího hodnocení (*benchmarkování*) RAG a vychází z něho tato podkapitola. Auepora je zaměřena na zodpovězení shrnutí 3 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 [44]

1. Vyhledávání
   - **Relevance** – Určuje, jak dobře vyhledané dokumenty odpovídají vstupnímu dotazu. Cílem je zjistit, zdali systém navrací dokumenty obsahující potřebné informace.
   - **Přesnost** – Porovnává vyhledané dokumenty se sbírkou určených kandidátních dokumentů a tím měří schopnost systému identifikovat relevantní dokumenty včetně jejich priotitizace před méně relevantími.
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** – Rychlost vyhledávání a produkce odpovědi, která je nezbytná pro uživatele komunikujícího se systémem.
   - Zbylé požadavky vychází z benchmarku RGB [9]:
   - **Odolnost proti šumu** – Šum představují dokumenty relevantní otázce, avšak neobsahující správnou odpověď.
   - **Odmítnutí** – Aby se předešlo halucinacím LLM, RAG by měl odmítnout odpovědět, pakliže obdržený kontext není dostatečný k poskytnutí odpovědi.
   - **Odolnosti vůči protichůdným informacím** – Generátor by měl identifikovat protichůdné 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 využívané specifické datasety, obecně je však vhodné použít např. index Wikipedie díky její rozmanitosti témat a rozsáhlosti článků. Ověřeným přístupem též je využití sady zpráv o aktualitách. Taková data představují informace, na kterých jazykové modely nebyly trénovány a zodpovědnost tak je plně na vyhledávacím modulu [44].

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* [16, 23], kde jazykový model je instruován k ohodnocení relevance, věrnosti či kvality na bodové škále. Tento přístup využívá mimojiné framework popsaný v následující podkapitole, který byl zároveň využit k evaluaci implementovaného RAG systému této práce.

RAGAs: Automatizovaná evaluace RAG systémů

Systém RAGAs (*Retrieval Augmented Generation Assessment*) [13] 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, vyčísleno od 0 do 1. Níže uvedené metriky vychází z cílů hodnocení uvedených zde 3.3. Es et. al. pak v [13] pomocí experimentů dokazují, 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

$$F = \frac{|V|}{|S|}, \tag{3.1}$$

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

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

kde $sim(q, q_i)$ 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 zodpovětení dotazu. Výsledné skóre je vypočteno jako poměr počtu extrahovaných vět a celkovému počtu vět v kontextu.

3.4 Aplikace

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 ve formě chatbota, který odpovídá na otázky pomocí externího zdroje informací. Zdroj informací zde představují textové dokumenty propojené se systémem. Jedná se tak o efektivní vyhledávání nad těmito dokumenty, přičemž otázka je podána přirozeným jazykem a odpověď je uživatelsky přívětivá, podložená fakty s odkazy na zdroje infromací. Využití systému není nijak limitováno, záleží pouze, jaká data na textové bázi má systém k dispozici.

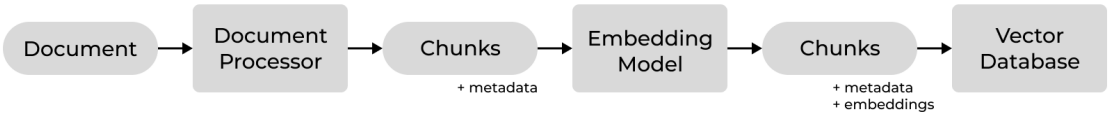
Systém je navrhnut tak, aby optimalizoval přesnost vyhledávání, která je nutná pro formulaci správné odpovědi generátorem. Vyhledávání dokumentů je cílové použití, jedná se však až o druhou fázi. První fází systému je zpracování dokumentů a jejich uložení do databáze.

4.1 Zpracování dokumentů

První 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 ukládájí do vektorové databáze.



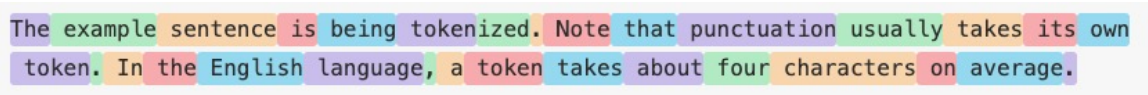
Obrázek 4.1: Diagram zpracování dokumentu až po uložení jeho reprezentace do databáze

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, nadpisy, věty) a dodržuje limit velikosti chunku.

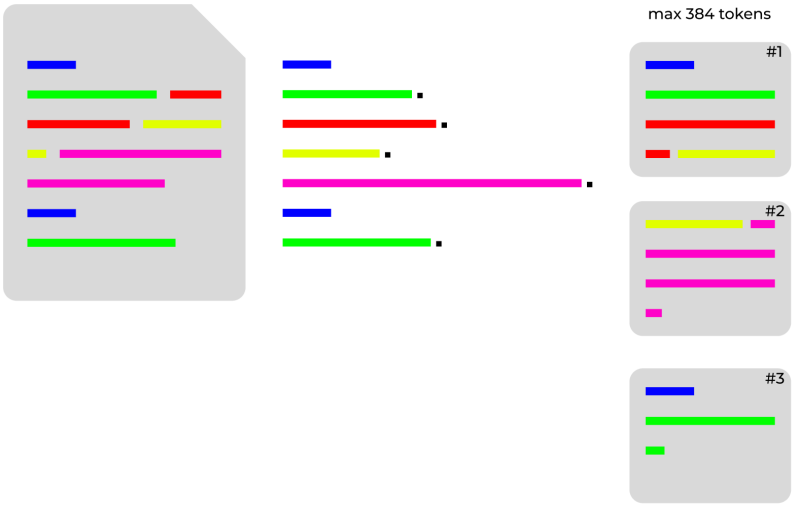
Metoda, jež systém implementuje, se snaží co nejvíce zachovat kontext a strukturu textu, přičemž velikost chunku omezuje především vstupní limit embedding modelu. Ten je počítán v tokenech, a proto i text je tokenizérem příslušného embedding modelu přepočítáván na tokeny.



Obrázek 4.2: Vizualizace tokenizace. [1]

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.

Nadpisy vždy tvoří počátek nového chunku, protože označují začátek nového logického celku.

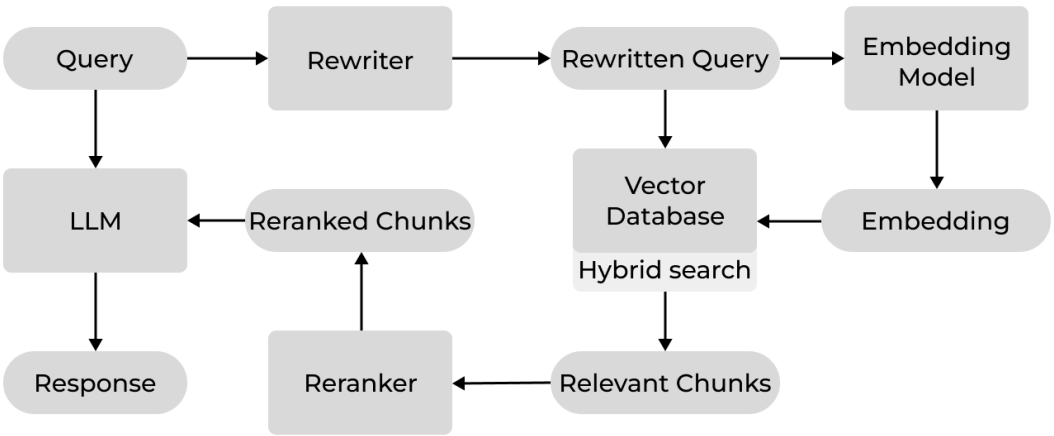


Obrázek 4.3: chunking

4.2 Vyhledávání dokumentů

Jakmile vektorová databáze obsahuje data, která slouží jako báze znalostí, nastává druhá fáze systému. Vyhledávání dokumentů pro vytvoření kontextu a poskytnutí formulované odpovědi jazykovým modelem je hlavním případem užití systému.

Dotaz uživatele je přeformulován do podoby optimalizovanější pro vyhledávání. Přeformulovaný dotaz je použit pro hybridní vyhledávání, které navrací relevantní chunky. Ačkoliv mají své skóre vyhledávání, jejich pořadí a relevance je upřesněna pomocí modulu `Reranker`. Tyto chunky pak slouží jako kontext pro jazykový model, který pomocí něj zodpovídá dotaz uživatele.



Obrázek 4.4: Základní diagram komponent

Přeformulování dotazu

Modul `Rewriter` slouží k optimalizaci dotazů pro vyhledávání. Uživatelé často formulují své dotazy neefektivně tím, že používají nadbytečná nebo vycpávková slova. Tento modul využívá jazykový model a vhodně ho instruuje k přeformulování dotazu do stručnější a informativnější podoby, aniž by změnil jeho původní význam.

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

Již 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.

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 $\alpha \in [0, 1]$, kde 0 se přiklání fulltextovému vyhledávání a 1 vektorovému. Výsledné skóre je pak vypočteno jako vážený průměr normalizovaných hodnot skóre jednotlivých vyhledávání:

$$s_{\text{hybrid}} = \alpha \cdot \frac{s_{\text{vector}} - \min(s_{\text{vector}})}{\max(s_{\text{vector}}) - \min(s_{\text{vector}})} + (1 - \alpha) \cdot \frac{s_{\text{fulltext}} - \min(s_{\text{fulltext}})}{\max(s_{\text{fulltext}}) - \min(s_{\text{fulltext}})}, \quad (4.1)$$

kde s_{fulltext} je skóre z fulltextového vyhledávání (např. BM25) a s_{vector} je skóre vektorového vyhledávání, které může být založeno např. na kosinové podobnosti nebo Euklidovské vzdálenosti.

Reranking

Proces vyhledávání lze považovat za dvoufázový, protože důležitým krokem po hybridním vyhledávání je tzv. *reranking*. Jedná se o proces přeuspořádání či zpřesnění výsledků původního vyhledávání podle míry relevance k dotazu uživatele.

Samotné vyhledávání často upřednostňuje rychlost před přesností, jelikož probíhá nad rozsáhlým množstvím dat (ANNS 2.5). V této fázi se typicky využívá architektura *Bi-encoder*, kde jsou dotaz a dokument zakódovány odděleně a jejich relevance je určena pomocí

podobnostní funkce (např. kosinové podobnosti). Tento přístup je rychlý, ale může u něho docházet ke ztrátě sémantických detailů. Výsledkem je pevně daný počet nejrelevantnějších k 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 infromace s nedostatečnou mírou relevance z kontextu odstraní. Vzhledem k vyšší výpočetní náročnosti se reranking aplikuje pouze na omezený počet top-k výsledků z předchozí fáze.

Výsledkem je vyšší kvalita odpovědi generované jazykovým modelem, protože do promptu se dostávají pouze skutečně relevantní informace a LLM tak není zahlceno přebytečnými informacemi [2].

Prompt Engineering

Posledním krokem k optimalizaci odpovědi před jejím generováním je návrh promptu, který slouží jako vstup jazykovému modelu a obsahuje instrukce společně s kontextem a dotazem. Cílem prompt engineeringu

Konkrétní implementace promptu včetně formátu vstupu je uvedena zde **??**.

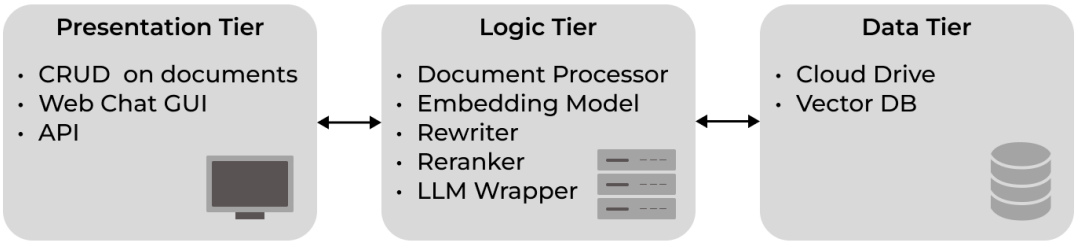
4.3 Integrace komponent

Předchozí podkapitoly uvedly princip dvou hlavních fází systému:

1. **Zpracování** – vstupem je textový dokument, který je zpracován a uložen do vektorové databáze.

2. **Vyhledávání** - 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 4.5.



Obrázek 4.5: Třívrstvá architektura systému

Synchronizace s cloudovým úložištěm

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

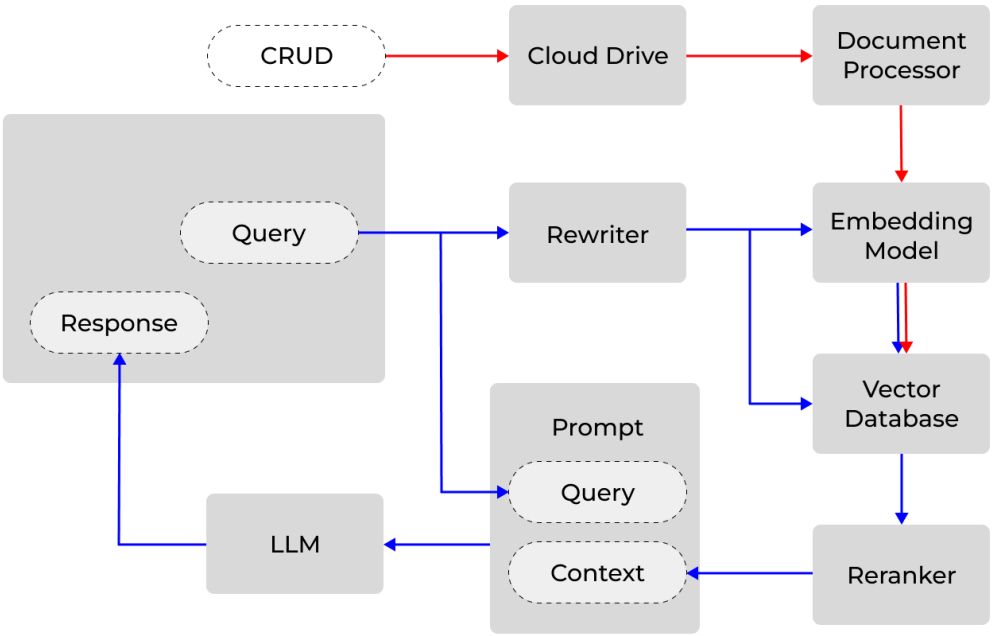
Tato složka je systémem monitorována. Jakmile se její stav změní, systém je o tom prostřednictvím API cloudové služby notifikován a zahajuje synchronizační rutinu, která obnáší detekci změn a zajištění jejich promítnutí do databáze.

Při detekci nového dokumentu je tento dokument stažen a prochází fází zpracování (4.1). 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 synchornizací.

Tok dat v systému

Na obrázku 4.6 je znázorněna celková architektura systému a vzájemné propojení komponent.



Obrázek 4.6: Základní diagram komponent

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.

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.

Základ systému je implementován v jazyce Python. Webová aplikace (uživatelské rozhraní) využívá framework Vue.js. 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:

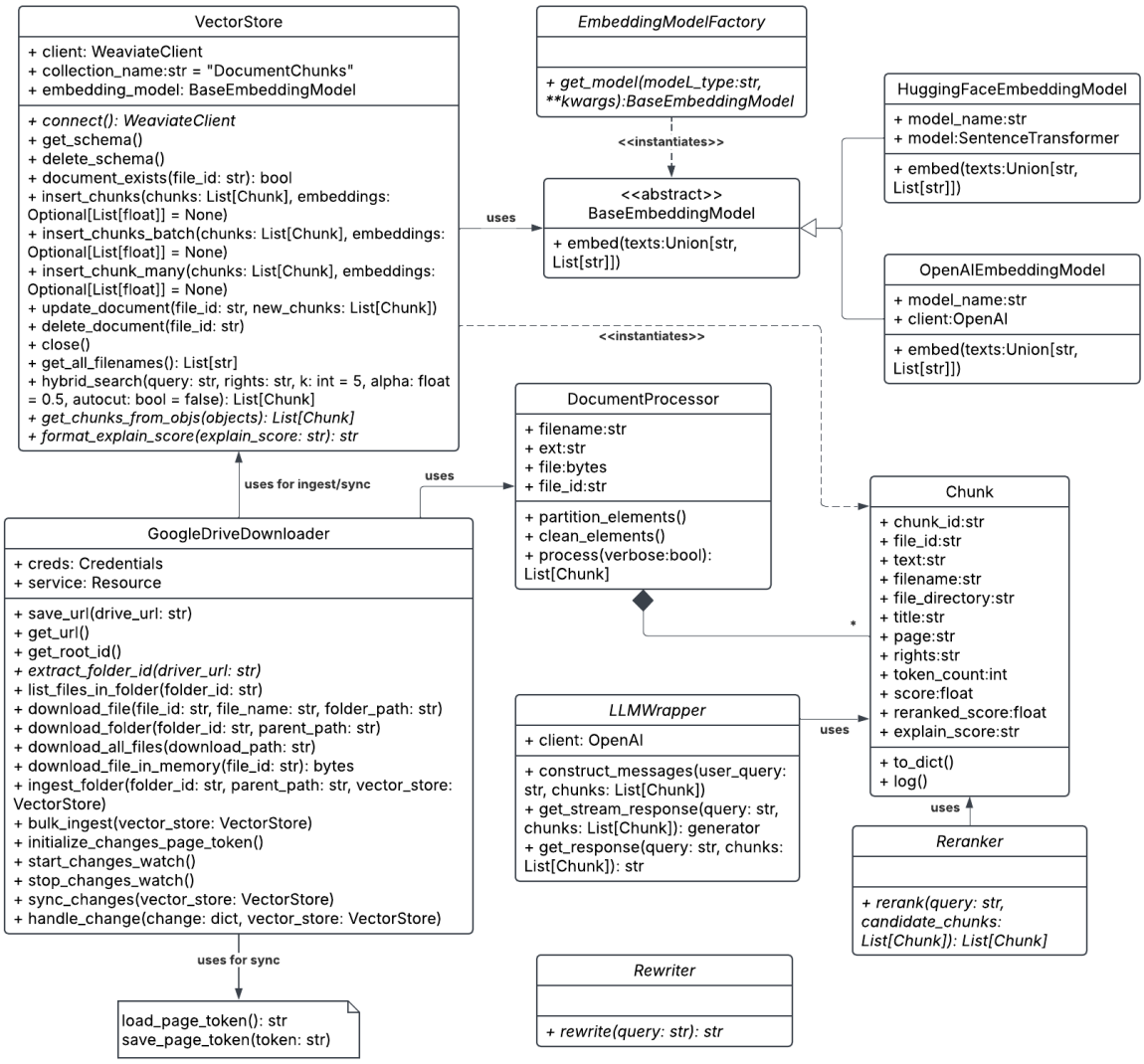
- FastAPI - webový framework k propojení backendu a frontendu skrze API.
- Unstructured - knihovna poskytující rozhraní pro zpracování nestrukturovaných dat.
- NLTK (Natural Language Toolkit) - sada knihoven pro zpracování přirozeného jazyka.
- Transformers (HuggingFace) - knihovna pro práci s předtrénovanými jazykovými modely, využita pro tokenizér.
- Sentence Transformers (HuggingFace) - nadstavba s modely pro vytváření vektorových reprezentací.
- Weaviate - open-source vektorová databáze s podporou hybridního vyhledávání.
- OpenAI API - přístup ke generativním modelům GPT.
- Google Drive API - rozhraní pro sledování změn a synchronizaci se složkou na Google Drive cloudovém úložišti.
- NGROK

- RAGAs - evaluační framework využit při testování a hodnocení systému (viz 6)

Konkrétní použití těchto technologií je rozebráno v následujících sekci.

5.2 Moduly systému

Systém byl navržen modulárně ...



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

`DocumentProcessor`

Třída `DocumentProcessor` slouží ke zpracování dokumentů. Při inicializaci obdrží cestu k lokálně 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` [41]. Ta poskytuje metody `partition()` pro různé typy souborů, které mají všechny výstupem seznam elementů reprezentujících logické celky dokumentu. Každý element obsahuje extrahovaný text, metadata a kategorii popisující účel v dokumentu (např. záhlaví, narativní text, nadpis apod. - možné kategorie elementu se odvíjí od typu dokumentu). Je nutné dodat, že `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 a odrážek v seznamech. Konfigurovatelným krokem k předpovídatelné struktuře datasetu je pak detekce nezaznamenaných nebo naopak odstranění přebytečných nadpisů, což vede k lepšímu dělení textu.

3. **Chunkování** je implementováno podle principu popsaného v kap. 4.1. Text jednotlivých elementů je dělen na věty pomocí `nltk.tokenize.sent_tokenize(text)` [5].

   Věty jsou analyzovány na počet tokenů. O tokenizaci se stará `AutoTokenizer` [14], jehož architektura je odvozena od použitého embedding modelu.

   Embedding model `all-mpnet-base-v2`, pro který jsou chunky připravovány, má vstupní limit 384 tokenů 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 též detekce nadpisů. Ta využívá kategorii elementů `Title`.

`EmbeddingModel`

Pro možnost výběru 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 je navržen pro obecné použití a z dostupných modelů poskytuje nejlepší kvalitu [38].

Třída `OpenAIEmbeddingModel` využívá rozhraní OpenAI API k získání embeddingů [29]. Tento způsob byl zkoumán jako alternativní, nicméně tato služba je zploplatněná, a tak bylo preferováno využití lokálního modelu pro finální verzi systému.

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

`VectorStore`

Pro implementaci externího úložiště znalostí s hybridním vyhledáváním byla využita open-source vektorová databáze Weaviate [42], která poskytuje jednoduché řešení pro lokální

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 tak využívá HNSW (viz 2.5) k indexaci a kosinovu podobnost jako podobností funkci vektorovému vyhledávání. Hybridní vyhledávání tak je realizováno sématnickým vyhledáváním společně s BM25. Strategie vyhledávání byla zvolena následovně:

```
response = collection.query.hybrid(
    query=query,
    vector=embedding_model.embed(query)[0],
    alpha=0.5,
    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 kap. 4.2. Parametr `auto-limit` představuje tzv. *autocut*, který nevrací pevný počet vyhledaných záznamů, ale k prvních shluků záznamů dle skóre. Dále je aplikován filtr dle přístupových práv uživatele k ilustraci metadata filtrování.

`Rewriter`

Modul `Rewriter` zaobaluje jediný úkol. Statická metoda `rewrite()` využívá OpenAI API [29] pro volání modelu `gpt-4o` k přeformulování dotazu uživatele do podoby optimalizovanější pro vyhledávání. Použitý prompt vypadá následovně:

```
system: Rewrite the query to optimize for retrieval.
        If a query is real nonsense, just return the original query
user:   {query}
```

`Reranker`

Modul `Reranker` mění pořadí a upřesňuje míru relevnace chunků, které byly vyhledáváním označeny za relevantní. Využívá model `cross-encoder/ms-marco-MiniLM-L-6-v2` [37]. Pomocí metody `rank(query, candidates)` pak určuje skóre relevance, dle kterého jsou chunky nově řazeny pro formulaci kontextu. Skóre se typicky pohybuje přibližně v intervalu $[-10, 10]$, kde chunky se záporným skóre jsou označeny jako nerelevantní dotazu a z kontextu vyřazeny.

`LLMWrapper`

[29]

`GoogleDriveDownloader`

[11]

5.3 API

[34]

5.4 Uživatelské rozhraní

Kapitola 6

Testování a evaluace

[10]

Kapitola 7

Závěr

Literatura

[1] Dostupné z: https://gpt-tokenizer.dev/.

[2] Ahmed, S. Dostupné z: https://medium.com/@sahin.samia/what-is-reranking-in-retrieval-augmented-generation-rag-ee3dd93540ee.

[3] Alammar, J. *The Illustrated Word2vec*. 2018. Dostupné z: https://jalammar.github.io/illustrated-word2vec/.

[4] Bernhardsson, E. *Annoy: Approximate Nearest Neighbors Oh Yeah* https://github.com/spotify/annoy. GitHub, 2015.

[5] Bird, S.; Loper, E. a Klein, E. *NLTK API Documentation: nltk.tokenize.sent_tokenize* https://www.nltk.org/api/nltk.tokenize.sent_tokenize.html. 2024. Accessed: 2025-03-25.

[6] Blumberg, R. a Atre, S. The Problem with Unstructured Data. *DM Review*, 2003, sv. 13, č. 2, s. 42–49. Dostupné z: https://www.soquelgroup.com/wp-content/uploads/2010/01/dmreview_0203_problem.pdf.

[7] Büttcher, S.; Clarke, C. L. A. a Cormack, G. V. *Information Retrieval: Implementing and Evaluating Search Engines*. Cambridge, Massachusetts, London, England: The MIT Press, 2010. ISBN 978-0-262-02651-2.

[8] Camacho-Collados, J. a Pilehvar, M. T. From Word to Sense Embeddings: A Survey on Vector Representations of Meaning. *CoRR*, 2018, abs/1805.04032. Dostupné z: http://arxiv.org/abs/1805.04032.

[9] Chen, J.; Lin, H.; Han, X. a Sun, L. *Benchmarking Large Language Models in Retrieval-Augmented Generation*. 2023. Dostupné z: https://arxiv.org/abs/2309.01431.

[10] Contributors, R. *Ragas Documentation (Stable)* https://docs.ragas.io/en/stable/. 2024. Accessed: 2025-03-25.

[11] Developers, G. *Google Drive API Documentation* https://developers.google.com/drive. 2024. Accessed: 2025-03-25.

[12] Devlin, J.; Chang, M.; Lee, K. a Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*, 2018, abs/1810.04805. Dostupné z: http://arxiv.org/abs/1810.04805.

[13] Es, S.; James, J.; Espinosa Anke, L. a Schockaert, S. *RAGAS: Automated Evaluation of Retrieval Augmented Generation*. 2023. Dostupné z: https://arxiv.org/abs/2309.15217.

[14] Face, H. *Transformers Documentation: AutoTokenizer* https://huggingface.co/docs/transformers/v4.49.0/en/model_doc/auto#transformers.AutoTokenizer. 2024. Accessed: 2025-03-25.

[15] Google. *Google Trends: ai vyhledávání*. Dostupné z: https://trends.google.com/trends/explore?date=all&q=ai.

[16] Gu, J.; Jiang, X.; Shi, Z.; Tan, H.; Zhai, X. et al. *A Survey on LLM-as-a-Judge*. 2025. Dostupné z: https://arxiv.org/abs/2411.15594.

[17] Gupta, S.; Ranjan, R. a Singh, S. N. *A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions*. 2024. Dostupné z: https://arxiv.org/abs/2410.12837.

[18] Han, Y.; Liu, C. a Wang, P. *A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge*. 2023. Dostupné z: https://arxiv.org/abs/2310.11703.

[19] Jégou, H.; Douze, M. a Schmid, C. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Institute of Electrical and Electronics Engineers, 2011, sv. 33, č. 1, s. 117–128. Dostupné z: https://hal.inria.fr/inria-00514462v2.

[20] Karpukhin, V.; Oguz, B.; Min, S.; Wu, L.; Edunov, S. et al. Dense Passage Retrieval for Open-Domain Question Answering. *CoRR*, 2020, abs/2004.04906. Dostupné z: https://arxiv.org/abs/2004.04906.

[21] Lewis, M.; Liu, Y.; Goyal, N.; Ghazvininejad, M.; Mohamed, A. et al. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *ArXiv preprint arXiv:1910.13461*, 2019. Dostupné z: https://arxiv.org/abs/1910.13461.

[22] Lewis, P. S. H.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *CoRR*, 2020, abs/2005.11401. Dostupné z: https://arxiv.org/abs/2005.11401.

[23] Li, H.; Dong, Q.; Chen, J.; Su, H.; Zhou, Y. et al. *LLMs-as-Judges: A Comprehensive Survey on LLM-based Evaluation Methods*. 2024. Dostupné z: https://arxiv.org/abs/2412.05579.

[24] Malkov, Y. A. a Yashunin, D. A. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *CoRR*, 2016, abs/1603.09320. Dostupné z: http://arxiv.org/abs/1603.09320.

[25] MathWorld Contributors. *Distance*. N.d. Dostupné z: https://mathworld.wolfram.com/Distance.html. Accessed: 2025-01-26.

[26] Michal Rost. *Co jsou vektorové databáze a proč jsou vhodné pro AI?* Dostupné z: https://www.michalrost.cz/co-jsou-vektorove-databaze-a-proc-jsou-vhodne-pro-ai.

[27] Murel, J. a Noble, J. What is an encoder-decoder model? *IBM*, 2024. https://www.ibm.com/think/topics/encoder-decoder-model.

[28] Musadiq Peerzada. *Vectors Unleashed: Navigating the Future with Vector Databases.* Dostupné z: https://blogs.musadiqpeerzada.com/vectors-unleashed-navigating-the-future-with-vector-databases.

[29] OpenAI. *OpenAI Platform Documentation Overview* https://platform.openai.com/docs/overview. 2024. Accessed: 2025-03-25.

[30] Pan, J. J.; Wang, J. a Li, G. *Survey of Vector Database Management Systems.* 2023. Dostupné z: https://arxiv.org/abs/2310.14021.

[31] Peng, D.; Gui, Z. a Wu, H. *Interpreting the Curse of Dimensionality from Distance Concentration and Manifold Effect.* 2024. Dostupné z: https://arxiv.org/abs/2401.00422.

[32] Pennington, J.; Socher, R. a Manning, C. Glove: Global Vectors for Word Representation. In:. Leden 2014, sv. 14, s. 1532–1543.

[33] Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S. et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 2020, sv. 21, č. 140, s. 1–67. Dostupné z: https://arxiv.org/abs/1910.10683.

[34] Ramírez, S. *FastAPI – Custom Response - StreamingResponse* https://fastapi.tiangolo.com/advanced/custom-response/#streamingresponse. 2024. Accessed: 2025-03-25.

[35] Reimers, N. a Gurevych, I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *CoRR*, 2019, abs/1908.10084. Dostupné z: http://arxiv.org/abs/1908.10084.

[36] Robertson, S. a Zaragoza, H. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval*, Leden 2009, sv. 3, s. 333–389.

[37] Sentence Transformers, U. *CrossEncoder Pretrained Models — Sentence-Transformers Documentation* https://sbert.net/docs/cross_encoder/pretrained_models.html. 2024. Accessed: 2025-03-25.

[38] Sentence Transformers, U. *Pretrained Models — Sentence-Transformers Documentation* https://sbert.net/docs/sentence_transformer/pretrained_models.html. 2024. Accessed: 2025-03-25.

[39] Shinde, R.; Goel, A.; Gupta, P. a Dutta, D. Similarity Search and Locality Sensitive Hashing using TCAMs. *CoRR*, 2010, abs/1006.3514. Dostupné z: http://arxiv.org/abs/1006.3514.

[40] 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. Dostupné z: http://dx.doi.org/10.1016/j.cogsys.2024.101216.

[41] Unstructured Technologies. *Open-Source Overview* https://docs.unstructured.io/open-source/introduction/overview. 2024. Accessed: 2025-03-25.

[42] Weaviate. *Weaviate Developer Documentation* https://weaviate.io/developers/weaviate. 2024. Accessed: 2025-03-25.

[43] Yenduri, G.; M, R.; G, C. S.; Y, S.; Srivastava, G. et al. *Generative Pre-trained Transformer: A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions*. 2023. Dostupné z: https://arxiv.org/abs/2305.10435.

[44] Yu, H.; Gan, A.; Zhang, K.; Tong, S.; Liu, Q. et al. Evaluation of Retrieval-Augmented Generation: A Survey. In: *Big Data*. Springer Nature Singapore, 2025, s. 102–120. ISBN 9789819610242. Dostupné z: http://dx.doi.org/10.1007/978-981-96-1024-2_8.

[45] Zhang, B.; Liu, X. a Lang, B. Fast Graph Similarity Search via Locality Sensitive Hashing. In: Ho, Y.-M.; Sang, J.; Ro, Y. M.; Kim, C.-S. a Wu, F., ed. *Advances in Multimedia Information Processing – PCM 2015*. Springer, Cham, 2015, sv. 9314, s. 352–361. Lecture Notes in Computer Science. Dostupné z: https://doi.org/10.1007/978-3-319-24075-6_60.

[46] Zhao, P.; Zhang, H.; Yu, Q.; Wang, Z.; Geng, Y. et al. *Retrieval-Augmented Generation for AI-Generated Content: A Survey*. 2024. Dostupné z: https://arxiv.org/abs/2402.19473.

[47] Zilliz. *Introduction to Unstructured Data*. 2022. Dostupné z: https://zilliz.com/learn/introduction-to-unstructured-data. Accessed: 2025-01-26.