VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

URČOVÁNÍ PODOBNOSTI PŘÍBĚHŮ

IDENTIFYING NARRATIVE SIMILARITY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. FRANTIŠEK SABOL**
AUTHOR

VEDOUCÍ PRÁCE **doc. RNDr. PAVEL SMRŽ, Ph.D.**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

164610

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Sabol František, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Zpracování zvuku, řeči a přirozeného jazyka
Název:	**Určování podobnosti příběhů**
Kategorie:	Umělá inteligence
Akademický rok:	2024/25

Zadání:

1. Seznamte se s pokročilými metodami strojového učení používanými pro automatickou tvorbu shrnutí (summarizaci) textů, reprezentaci významu a určování podobnosti.
2. Prostudujte rozhraní dostupných nástrojů pro analýzu textů, identifikaci postav a jejich důležitých atributů.
3. Na základě získaných znalostí navrhněte a realizujte systém pro určování podobnosti a odlišnosti příběhů a reprezentaci výsledků na základě extrahovaných klíčových slov, týkajících se děje.
4. Vytvořte testovací sadu pro vyhodnocení přesnosti a úplnosti hledání a vyhodnoťte vytvořený systém pomocí těchto standardních metrik.
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- dle doporučení vedoucího

Při obhajobě semestrální části projektu je požadováno:

- funkční prototyp řešení

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

Vedoucí práce:	**Smrž Pavel, doc. RNDr., Ph.D.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2024
Termín pro odevzdání:	21.5.2025
Datum schválení:	12.11.2024

Abstrakt

Porovnávanie podobnosti príbehov predstavuje komplexnú výzvu. Táto diplomová práca rieši tento problém návrhom a implementáciou systému využívajúceho veľké jazykové modely pre aspektovo orientovanú analýzu príbehov – zameranú na dej, postavy, prostredie a témy – s cieľom umožniť granulárne a interpretovateľné porovnávania. Významným prínosom práce je webové používateľské rozhranie, ktoré umožňuje exploratívne vyhľadávanie, vizualizáciu podobností naprieč aspektmi a detailné skúmanie analýz. Systém taktiež integruje agenta založeného na princípoch Retrieval-Augmented Generation (RAG) pre objavovanie príbehov. Pre evaluáciu systému a prínosu analýz veľkých jazykových modelov bola vytvorená nová dátová sada určená na evaluáciu vyhľadávania podobných príbehov. Experimentálne výsledky potvrdzujú, že dáta z viac-aspektovej analýzy, integrované do vyhľadávacieho procesu, majú pozitívny vplyv na výsledne pozorované metriky. Práca tak demonštruje funkčný systém a potvrdzuje prínos granulárnej, aspektovej analýzy, prezentovanej prostredníctvom používateľsky orientovaného rozhrania, pre exploráciu podobnosti medzi príbehmi.

Abstract

Comparing story similarity presents a complex challenge. This diploma thesis addresses this problem by designing and implementing a system that utilizes large language models for aspect-oriented story analysis – focusing on plot, characters, setting, and themes – to enable granular and interpretable comparisons. A significant contribution of the thesis is a web user interface that allows for exploratory search, visualization of similarities across aspects, and detailed examination of analyses. The system also integrates an agent based on Retrieval-Augmented Generation (RAG) principles for story discovery. For the evaluation of the system and the contribution of large language model analyses, a new dataset was created for evaluating the retrieval of similar stories. Experimental results confirm that data from the multi-aspect analysis, integrated into the search process, have a positive impact on the observed metrics. The thesis thus demonstrates a functional system and confirms the benefit of granular, aspect-based analysis, presented through a user-oriented interface, for exploring similarity between stories.

Kľúčové slová

veľké jazykové modely, naratológia, analýza príbehov, porovnávanie podobnosti príbehov, sémantické vyhľadávanie, RAG, Vespa, vektorová reprezentácia textu, Adalflow

Keywords

large language models, narratology, aspect-oriented narrative analysis, narrative similarity, semantic retrieval, RAG, Vespa, text embeddings, Adalflow

Citácia

SABOL, František. *Určování podobnosti příběhů*. Brno, 2025. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. RNDr. Pavel Smrž, Ph.D.

Určování podobnosti příběhů

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána doc. RNDr. Pavla Smrža, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
František Sabol
27. mája 2025

Poďakovanie

Predovšetkým by som sa chcel poďakovať pánovi doc. RNDr. Pavlovi Smržovi, Ph.D., za jeho cenné rady, konštruktívne pripomienky a trpezlivosť počas celého procesu tvorby tejto práce.

V neposlednom rade by som chcel vyjadriť vďaku mojej rodine a blízkym priateľom za ich podporu, povzbudenie, pochopenie a trpezlivosť, ktoré mi poskytovali nielen počas písania diplomovej práce, ale počas celého môjho štúdia. Bez ich opory by bolo dokončenie tejto práce omnoho náročnejšie.

Obsah

Kapitola 1

Úvod

Porovnávanie podobnosti príbehov predstavuje náročnú úlohu, ktorá si vyžaduje nielen pochopenie samotného jazyka, ale aj analýzu príbehových štruktúr, tematiky a častokrát aj pochopenie zámeru autora. S príchodom veľkých jazykových modelov (LLM) sa táto úloha stáva realizovateľnejšou, keďže umožňujú analyzovať príbeh na rôznych úrovniach abstrakcie a z rôznych aspektov, vďaka ich schopnosti spracovať vstup spolu s definíciou úlohy a inštrukciami od používateľa.

Napriek ich pokrokom v širokej škále úloh však LLM čelia viacerým obmedzeniam. Príbehy často obsahujú komplexné vzťahy postáv, skryté významy a kultúrne špecifiká, ktoré môžu byť pre tieto modely náročné na interpretáciu, čo vedie k nespoľahlivej analýze. Výsledky súčasných štúdií ukazujú, že hĺbkové porozumenie príbehov je stále náročnou úlohou aj pre tie najlepšie modely. Súčasné modely dokážu generovať dôveryhodné sumarizácie iba pre približne 50% testovaných príbehov [52], a ich schopnosť porozumieť komplexným vzťahom medzi postavami sa tiež ukazuje byť výrazne obmedzenou [65]. Neberúc do úvahy vplyv samotnej štrukturálnej, tematickej a obsahovej zložitosti príbehu, kvalita výstupu závisí aj na vhodne definovanom dotaze pre konkrétnu verziu modelu, dĺžke analyzovaného textu, požadovanej dĺžke výstupného textu a požadovanom formátovaní výstupu, ktoré je nevyhnutné pre automatické spracovanie výsledkov. Aj napriek týmto výzvam má ich aplikácia svoje opodstatnenie v automatizovanej analýze naratívov ako dokumentujú viaceré štúdie [1, 10].

Cieľom tejto práce je navrhnúť a implementovať systém na porovnávanie a vyhľadávanie príbehov, ktorý využíva veľké jazykové modely na viac-aspektovú dekompozíciu príbehov, s cieľom umožniť vyhľadávanie a určovanie podobnosti na nižšej úrovni granularity. Systém sa zameriava na prekonanie obmedzení tradičných prístupov tým, že umožňuje porovnanie príbehov na úrovni kľúčových naratívnych elementov – deja, postáv, prostredia a tém – a poskytuje používateľom interpretovateľné výsledky. Na dosiahnutie tohto cieľa práca predstavuje metodiku štruktúrovanej analýzy príbehov pomocou LLM, využitie sémantických vektorových reprezentácií pre tieto extrahované aspekty a ich integráciu do vyhľadávacieho systému postaveného na vektorovej databáze. Súčasťou práce je aj vývoj interaktívnej webovej aplikácie a agenta založeného na princípoch Retrieval-Augmented Generation (RAG) pre objavovanie príbehov.

Práca je rozdelená do niekoľkých kapitol. Kapitola *Veľké jazykové modely* (kap. 2) systematicky rozoberá teoretické východiská a technologický pokrok, ktorý viedol k vývoju súčasných transformer architektúr. V kapitole *Určovanie podobnosti príbehov* (kap. 3) sú predstavené rôznorodé prístupy k porovnávaniu príbehov, inšpirované naratológiou, s dôrazom na ich silné a slabé stránky. Kapitoly *Návrh riešenia* (kap. 4) a *Implementácia* (kap. 5) detailne

opisujú architektúru navrhovaného systému. To zahŕňa metodiku viac-aspektovej analýzy príbehov pomocou LLM, návrh reprezentácie príbehov, implementáciu sémantického vyhľadávania s využitím vektorovej databázy Vespa, vývoj webovej aplikácie pre interakciu s používateľom a integráciu RAG agenta. Rozoberané sú aj použité dátové zdroje (kap. 6). Experimentálna časť (kap. 7) prezentuje výsledky hodnotenia kľúčových komponentov systému, vrátane výberu embedding modelov a efektivity navrhnutých profilov relevancie. Záver práce (kap. 8) sumarizuje dosiahnuté výsledky a načrtáva možné smery pre zlepšenie systému.

Kapitola 2

Veľké jazykové modely

Veľké jazykové modely (LLM, z angl. Large Language Models) sú trieda transformer modelov charakterizovaných masívnou škálou parametrov (desiatky až stovky miliárd) a trénovaných na extrémne rozsiahlych textových korpusoch. Táto kapitola systematicky rozoberá ich teoretické základy a technologický vývoj.

Úvodná časť kapitoly sa venuje reprezentácii textu, kde je kľúčová tokenizácia — proces rozkladu textu na menšie jednotky (tokeny) pomocou metód ako Byte-Pair Encoding (BPE) alebo SentencePiece. Nasledujúca časť mapuje vývoj jazykových modelov pred transformermi — od štatistických n-gramov po neurónové siete (RNN, LSTM), ktoré napriek pokroku narážali na obmedzenia v modelovaní dlhých závislostí a paralelizácii. Sekcia 2.3 detailne rozoberá transformer architektúru, jej štruktúru (kódovacie a dekódovacie bloky), kľúčové varianty (encoder-decoder, encoder-only, decoder-only) a trénovacie stratégie, ktoré viedli k vzniku predtrénovaných modelov.

Kapitola sa ďalej rozoberá samotné veľké jazykové modely, pričom objasňuje ich autoregresívnu povahu, schopnosť učenia z kontextu (in-context learning), ktorá z nich robí univerzálny nástroj na riešenie rôznych úloh, ako aj výzvy spojené s obmedzeným kontextom. V nadväznosti na to sekcia 2.5 predstavuje princípy sémantického vyhľadávania. Zaoberá sa transformáciou textu na vektorové reprezentácie a efektívnymi metódami vyhľadávania podobných dokumentov, vrátane približných metód ako Hierarchical Navigable Small World (HNSW), ktoré sú dôležité pre prácu s rozsiahlymi dátovými kolekciami. Na tieto koncepty nadväzuje sekcia 2.6, ktorá popisuje techniky generovania podporovaného vyhľadávaním (Retrieval-Augmented Generation, RAG). Táto metóda efektívne kombinuje generatívne schopnosti LLM s možnosťou prístupu k externým znalostným bázam, čím zvyšuje faktickú presnosť a relevantnosť generovaných výstupov.

Záverečná časť kapitoly, konkrétne sekcia 2.7, sa venuje optimalizácii komplexných systémov založených na LLM. Zameriava sa na techniky úpravy vstupných textov, známe ako prompt inžinierstvo, a predstavuje pokročilé automatizované prístupy, akým je napríklad textová automatická diferenciácia.

2.1 Reprezentácia textu

Pri spracovaní prirodzeného jazyka modernými modelmi, akými sú transformer modely, je nevyhnutné, aby bol text vhodne prevedený do numerickej formy. Táto reprezentácia musí čo najlepšie zachytiť sémantiku, syntax a poradie tokenov v sekvencii. Proces reprezentácie

textu sa zvyčajne skladá z dvoch krokov: tokenizácie a následnej transformácie na vektorovú reprezentáciu.

Tokenizácia

Tokenizácia je metóda rozdeľovania vstupného textu do menších jednotiek, nazývaných *tokeny*. V praxi môže byť jeden token celé slovo, podslovo, prípadne znak. V transformer architektúrach, ktoré často pracujú s veľkým slovníkom, sa najčastejšie používajú tzv. *subword* tokenizačné techniky.

Hlavnou motiváciou subword tokenizácie je efektívne spracovanie aj zriedkavo sa vyskytujúcich alebo nových slov. Bez ohľadu na to, aké nezvyčajné je dané slovo, dokáže sa rozložiť na tokeny, ktoré sú modelom už známe.

Výstup tokenizácie. Výsledkom tokenizácie je sekvencia tokenov, ktoré reprezentujú pôvodný text. Každému tokenu je priradený index zo slovníka (*vocabulary*), čo vytvára podklad pre vstup do embedding vrstvy. V praxi tak pre vstupný text: $t_1\ t_2\ \ldots\ t_n$, získame sekvenciu tokenov $\boldsymbol{T} = (T_1, T_2, \ldots, T_m)$, pričom $m \geq n$ (pretože jedno slovo môže byť rozložené na viacero podslov). V praxi však výstup z tokenizátora nie je sekvencia tokenov ale ich indexov, na základe ktorých vieme nájsť samotný token vo vytvorenom slovníku (de-tokenizácia).

Byte-Pair Encoding (BPE)

Metóda, pri ktorej sa začína tokenizovať po znakoch a najčastejšie dvojice znakov sa rekurzívne spájajú, pokiaľ to maximalizuje pokrytie textu v trénovacom korpuse [48]. Táto technika pomáha ošetriť problém neznámych slov tak, že slovo sa rozdelí na menšie segmenty znakov (napr. dvojice znakov).

WordPiece a SentencePiece

WordPiece a SentencePiece sú metódy subword tokenizácie, ktoré rozdeľujú slová na menšie jednotky pomocou štatistických algoritmov (napr. BPE). Kým WordPiece sa používa v modeloch ako BERT [12], SentencePiece [30] je preferovaný algoritmus pre univerzálne viacjazykové modely (napr. T5 [45]) hlavne kvôli tomu, že nevyžaduje konkrétne predspracovanie textu, ktoré môže byť závislé na konkrétnom jazyku (napr. v japončine nie sú slová explicitne oddelené medzerou). Podporuje dva algoritmy — BPE a Unigram Language Model [29] — a poskytuje možnosť vybrať si fixnú veľkosť slovnej zásoby (napr. 32k tokenov).

Špeciálne tokeny

V mnohých NLP úlohách sa využívajú špeciálne tokeny, ktoré sú pridávané do slovníkov, napríklad:

- `[CLS]` alebo `<s>`: začiatočný token sekvencie (Class/Start).
- `[SEP]` alebo `</s>`: koncový (separačný) token, prípadne oddeľuje dve rôzne sekvencie.
- `[PAD]`: token na *padding*, ktorý slúži na zarovnanie dĺžok sekvencií v skupine vstupov (batch) počas tréningu.

Tieto tokeny zabezpečujú, že model je schopný jednotne spracovať rôzne dĺžky vstupov a že pre špecifické úlohy, sa dá dotrénovaním zabezpečiť to, aby vektorová reprezentácia tokenu `[CLS]` zohľadňovala sémantické, lexikálne a ďalšie vlastnosti celého vstupu.

2.2 Jazykové modely pred transformermi

V ére pred hlbokým učením boli štatistické jazykové modely pokusom o formalizáciu jazyka ako stochastického procesu. Ich pravdepodobnostný formalizmus predznamenal autoregresívnu povahu veľkých jazykových modelov, kde generovanie textu stále vychádza z podmienených pravdepodobností $P(w_t|w_{<t})$. Jedným z tejto triedy modelov je napríklad N-gramový jazykový model [26]. Pre sekvenciu slov $w_1, \ldots, w_n$ definoval pravdepodobnostný model:

$$P(w_0, \ldots, w_n) = \prod_{i=1}^{n} P(w_i|w_{1:i-1}). \tag{2.1}$$

Na základe Markovovho predpokladu, že slovo závisí iba na predošlom kontexte:

$$P(w_i|w_{1:i-1}) \approx P(w_i|w_{i-N+1:i--1}), \tag{2.2}$$

kde N (typicky 2 pre bigram) obmedzoval kontextové okno. Pravdepodobnosť bigramu „číta knihu“ sa odvodzovala priamo z korpusových dát:

$$P(\text{knihu}|\text{číta}) = \frac{C(\text{číta knihu}) + \alpha}{C(\text{číta}) + \alpha V}, \tag{2.3}$$

kde $C(\cdot)$ označuje počet výskytov n-gramu v korpuse, α predstavuje Laplaceovo vyhladzovanie (spôsob ošetrenia neznámeho kontextu) a V veľkosť slovnej zásoby.

Tento prístup, hoci zdanlivo primitívny, uviedol podstatnú myšlienku: jazyk je predvídateľný sekvenčný proces, kde je každé slovo funkciou svojho bezprostredného okolia.

Neurónové siete

Príchod neurónových sietí do spracovania prirodzeného jazyka (NLP) znamenal kvalitatívny zlom v modelovaní jazyka. Oproti štatistickým modelom, ktoré explicitne manipulovali s frekvenciami n-gramov, neurónové siete pracovali s kontinuálnymi reprezentáciami slov spolu s ich kontextom.

Od symbolov k vektorom. Základným princípom sa stalo embedding mapovanie: každé slovo w zo slovníka $\mathcal{V}$ bolo reprezentované ako d-rozmerný vektor $\boldsymbol{e}_w \in \mathbb{R}^d$, kde podobné slová mali podobné vektorové reprezentácie v naučenom priestore. Tento priestor sa učil automaticky počas trénovania pomocou backpropagácie, čo umožňovalo modelom zachytiť sémantické vzťahy (napr. $\boldsymbol{e}_{\text{kráľ}} - \boldsymbol{e}_{\text{muž}} + \boldsymbol{e}_{\text{žena}} \approx \boldsymbol{e}_{\text{kráľovná}}$), generalizovať pomocou distribuovaných vlastností vektorovej reprezentácie na zriedkavo sa vyskytujúce slová a modelovať jazyk pomocou kompozície lineárnych vrstiev a nelineárnych aktivácií.

Feedforward siete. Prvé neurónové jazykové modely (napr. [4]) používali dopredné siete s jednou skrytou vrstvou, ktoré pre N-gramový kontext $(w_{t-N+1}, \ldots, w_{t-1})$ predpovedali w_t pomocou:

$$\boldsymbol{h} = \sigma\left(\boldsymbol{W}_{\text{emb}} \cdot [\boldsymbol{e}_{w_{t-N+1}}; \ldots; \boldsymbol{e}_{w_{t-1}}] + \boldsymbol{b}\right), \quad P(w_t) = \text{softmax}(\boldsymbol{U}\boldsymbol{h}), \tag{2.4}$$

kde:

- $\boldsymbol{h} \in \mathbb{R}^{d_h}$: vektor skrytej vrstvy dimenzie d_h
- σ: nelineárna aktivačná funkcia (typicky sigmoid/tanh)
- $\boldsymbol{W}_{\text{emb}} \in \mathbb{R}^{d_h \times (N-1)d_e}$: matica pre transformáciu konkatenovaných embeddingov
- $\boldsymbol{e}_w \in \mathbb{R}^{d_e}$: embedding slova w s dimenziou d_e
- $\boldsymbol{U} \in \mathbb{R}^{|\mathcal{V}| \times d_h}$: matica pre mapovanie skrytej vrstvy na výstupnú distribúciu
- $\boldsymbol{b} \in \mathbb{R}^{d_h}$: bias vektor

Pričom $\boldsymbol{W}_{\text{emb}}$ a $\boldsymbol{U}$ boli trénovateľné parametre. Tieto modely síce zvládali dlhšie kontexty ako n-gramy, ale stále obmedzovali kontext na fixné N, čo motivovalo hľadanie architektúr s dynamickým kontextom.

Sekvenčný posun k RNN

Pred vznikom transformer architektúry dominovali v modelovaní sekvencií pre spracovanie prirodzeného jazyka rekurentné neurónové siete (RNN) a siete Long Short-Term Memory (LSTM). Tieto architektúry odstránili obmedzenie fixného okna zavedením skrytého stavu, ktorý inkrementálne včleňoval informáciu z celého prefixu $w_{1:t}$. Napriek úspechom v oblastiach, ako je strojový preklad a rozpoznávanie reči, RNN a LSTM čelili zásadným problémom [66]:

- problém miznúcich a explodujúcich gradientov: RNN využívali spätné šírenie chýb v čase, čo viedlo k tomu, že pri dlhých sekvenciách gradienty buď výrazne klesali (mizli), alebo naopak nadmerne rástli (explodovali) [41].
- obmedzenie sekvenčného spracovania: každý token musel byť spracovaný jeden po druhom, čo znemožňovalo paralelizáciu medzi krokmi sekvencie. To spôsobovalo, že trénovanie na rozsiahlych textových korpusoch bolo výrazne pomalé.

Tieto obmedzenia motivovali hľadanie architektúr, ktoré by dokázali spracovať dlhé sekvencie efektívnejšie a paralelne, čo nakoniec viedlo k vzniku transformer architektúry.

2.3 Transformer architektúra

Transformery, predstavené v práci *Attention is All You Need* [55], zásadne prekonali obmedzenia predchádzajúcich architektúr tým, že nahradili sekvenčné spracovanie paralelným výpočtom globálnych závislostí. Kľúčovou inováciou je mechanizmus pozornosti (angl. self-attention), ktorý explicitne modeluje interakcie medzi všetkými pármi tokenov v sekvencii bez ohľadu na ich vzdialenosť.

Základnými stavebnými jednotkami transformer architektúry sú kódovací blok a dekódovací blok. Ich úlohy sú komplementárne: zatiaľ čo kodér analyzuje a kontextualizuje vstupnú sekvenciu, dekodér ju transformuje (generuje) na výstupnú sekvenciu s využitím získaných reprezentácií. Obe časti pozostávajú z viacerých identických vrstiev, ktoré kombinujú mechanizmus pozornosti a transformácie cez dopredné siete, no líšia sa v spôsobe prístupu k maskovaniu tokenov pri výpočte pozornosti a spracovaniu kontextu.

Kódovací blok

Kódovací blok (encoder) je zodpovedný za transformáciu vstupnej sekvencie do reprezentácie, ktorá zvýrazňuje vzájomné vzťahy medzi tokenmi. V pôvodnom návrhu [55] sa skladá z viacerých identických vrstiev. Každá vrstva má dve podvsrtvy:

1. *Multi-Head Self-Attention* (MHA): Táto časť využíva tzv. *self-attention*, teda každý token sa „pozerá“ na všetky ostatné tokeny v sekvencii a váhovo agreguje ich reprezentácie. Pretože jediná hlava (tzv. head) môže byť príliš obmedzená, využíva sa koncept viachlavej pozornosti, kde sa paralelne aplikujú viaceré hlavy a ich výstupy sa následne spoja.

2. *Plne prepojené dopredné siete* (FFN): Za každým self-attention mechanizmom nasleduje plne pripojená dopredná neurónová sieť (typicky s dvoma lineárnymi transformáciami a nelineárnou aktiváciou).

Okrem týchto komponentov kódovací blok využíva *reziduálne spojenia* (skip connection) a *vrstvovú normalizáciu* (LayerNorm). Reziduálne spojenia napomáhajú prúdeniu gradientov a LayerNorm stabilizuje distribúciu aktivácií v priebehu trénovania [32].

Dekódovací blok

Dekódovací blok (decoder) má podobnú štruktúru ako kódovací blok, avšak jednotlivé vrstvy sú tvorené troma podvrstvami:

1. *Masked Multi-Head Self-Attention*: Na rozdiel od kódovacej časti je mechanizmus pozornosti v dekódovacom bloku *maskovaný*, aby dekodér pri predpovedaní tokenu w_t nevidel budúce tokeny $w_{t+1}, w_{t+2} \ldots$. Tento maskovací mechanizmus zabezpečuje autoregresívny charakter generovania sekvencie.

2. *Multi-Head Cross-Attention*: Spája dekodér s výstupmi kodéra. Dotazy (queries) pochádzajú z aktuálneho stavu dekodéra, kým kľúče (keys) a hodnoty (values) sú prevzaté z kodéra.

3. *Plne prepojené dopredné siete* (FFN): identické s kodérom.

Aj dekódovací blok obsahuje reziduálne spojenia s normalizáciou. Výstupná sekvencia sa generuje autoregresívne[1]: v každom kroku sa vyberie token s najvyššou pravdepodobnosťou, ktorý sa pridá do výstupu a stane sa vstupom pre nasledujúci krok. Tento proces pokračuje, kým nie je vygenerovaný špeciálny ukončovací token (napr. `</s>`) alebo kým sa nedosiahne maximálna dĺžka výstupu. Výber tokenov sa riadi dekódovacími stratégiami [23]:

- **Greedy decoding**: vyberie token s najvyššou pravdepodobnosťou,
- **Beam search**: udržuje k najlepších čiastkových sekvencií,
- **Nucleus sampling**: náhodný výber z top-p pravdepodobných tokenov.

[1] Vo fáze trénovania dekodér negeneruje výstup autoregresívne, keďže sa využíva technika teacher forcing, pri ktorej sa nevyužíva vygenerovaný token ale referenčný token z trénovacích dát: https://discuss.huggingface.co/t/what-to-use-for-the-target-input-in-the-decoder-for-autoregressive-usage/10037.

Varianty tranformer modelov

V praxi sa transformer modely líšia v tom, či využívajú obidve časti (encoder aj decoder), alebo len jednu. Toto rozdelenie nie je ľubovoľné – každá konfigurácia optimalizuje architektúru pre špecifický typ úloh a spôsob interakcie so sekvenciami.

Encoder-Decoder

Toto je pôvodný návrh z práce *Attention is All You Need* [55]. Tento prístup sa dodnes používa v modeloch ako T5 [45] alebo BART [33], ktoré rozširujú pôvodný koncept o rôzne stratégie tréningu.

Encoder

V tejto konfigurácii zostáva iba kódovací blok a model sa využíva najmä na úlohy, kde je výstup využitý ako klasifikačný vektor alebo iná fixná reprezentácia. Príkladom je BERT [12], ktorý spracúva text vcelku (nemá generatívnu časť) a výstup využíva napr. na klasifikáciu, extrakciu entít, či iné diskriminačné úlohy.

Encoder-only architektúra sa v súčasnosti významne uplatňuje v sémantickom vyhľadávaní, kde vektorové reprezentácie slov, viet alebo celých dokumentov umožňujú efektívne vyhľadávanie na základe významovej podobnosti. V praxi sa ako agregovaná reprezentácia dokumentu často používa embedding špeciálneho tokenu `[CLS]` alebo vážený priemer všetkých výstupných vektorov sekvencie [46]. Na vyhodnotenie podobnosti medzi vektormi sa využívajú rôzne metódy, príkladom môže byť: kosínová podobnosť, euklidovská vzdialenosť a ďalšie [63].

Decoder

Tento typ transformera neobsahuje kodér (teda ani cross-attention vrstva nie je súčasťou dekodéra) a spolieha sa výhradne na autoregresívne generovanie a mechanizmus maskovanej pozornosti, kde každý token vidí iba predchádzajúce tokeny. Modely ako GPT [44] generujú text postupne: v každom kroku vypočítajú distribúciu pravdepodobností $P(w_t \mid w_{<t}; X)$ (X je vstupná sekvencia) nad celým slovníkom a vyberajú token optimalizujúci celkovú pravdepodobnosť výstupnej sekvencie.

Prístup k trénovaniu

Paralelné spracovanie dlhých sekvencií a škálovateľnosť transformer modelov umožnili trénovanie na masívnych korpusoch textu, čím vznikli predtrénované jazykové modely (PLM). Tieto modely, napríklad BERT [12], využívajú univerzálne jazykové reprezentácie naučené počas fázy *predtrénovania* (pretraining), ktoré sa následne rýchlo adaptujú na špecifické úlohy (klasifikácia, preklad, generovanie) prostredníctvom *doladenia* (fine-tuning) na menšej sade dát pre danú úlohu.

2.4 Veľké jazykové modely

Veľké jazykové modely (LLM, z angl. Large Language Models) sú trieda modelov charakterizovaných masívnou škálou parametrov (desiatky, stovky miliárd až bilióny) a trénovaných na extrémne rozsiahlych textových korpusoch. Zväčša ide o modely vychádzajúce z

transformer architektúry, no dnes už sú známe aj iné typy ako napr. veľké difúzne jazykové modely [39]. Podľa Kaplana a kol. [27] sa výkon LLM riadi tzv. zákonmi škálovania, podľa ktorých zvyšovanie počtu parametrov, objemu dát a výpočtových zdrojov vedie k predvídateľnému zlepšeniu vo všetkých metrikách. Zaujímavým fenoménom je však vznik emergentných schopností — kvalitatívne nových vlastností (napr. tvorba analógií, dedukcia pravidiel), ktoré sa objavujú iba pri dosiahnutí kritickej veľkosti modelu a nie sú explicitne trénované [58].

Autoregresívne LLM

Jednou z vlastností autoregresívnych LLM je schopnosť riešiť úlohy prostredníctvom *učenia z kontextu* (in-context learning). Tento prístup nevyžaduje explicitné doladenie modelu na konkrétnu úlohu, ale spolieha sa na schopnosť modelu extrapolovať riešenie z krátkeho príkladu alebo inštrukcie vloženej priamo do vstupného textu (tzv. *prompt*).

Výhody a obmedzenia. Učenie z kontextu umožňuje rýchlu adaptáciu modelu na nové úlohy bez potreby aktualizácie parametrov, čo je obzvlášť cenné pri doménach s obmedzeným počtom dát a obmedzenými výpočtovými zdrojmi. Výkon však výrazne závisí od formulácie promptu: nekonzistentné formátovanie, nejednoznačné inštrukcie alebo neoptimálny výber príkladov môžu viesť k výraznému poklesu presnosti [38]. Okrem toho, LLM majú tendenciu generovať *halucinácie* — hodnoverné, ale fakticky nesprávne tvrdenia — najmä v doménach s nízkou zastúpenosťou v trénovacích dátach [22]. Voľnosť, ktorú umožňuje slovný opis riešenia úlohy a inštrukcií v prompte, dáva priestor pre vznik celej rady techník, ktoré zlepšujú výsledky [47]. V praxi často využívané sú napr.:

- **Zero-shot learning**: model dostane iba textový popis úlohy bez príkladov,
- **Few-shot learning**: spolu s textovým popisom úlohy vstup obsahuje aj niekoľko vstupno-výstupných párov, ktoré modelu ilustrujú formát a logiku úlohy,
- **Chain-of-Thought (CoT)** [59]: špeciálny typ few-shot promptu, kde príklady zahŕňajú aj explicitnú logickú úvahu vedúcu k riešeniu. Táto technika zvyšuje presnosť v úlohách vyžadujúcich uvažovanie.

Obmedzený kontext

Autoregresívne veľké jazykové modely (LLM) čelia dvom výzvam: obmedzenému prístupu k aktuálnym/externým znalostiam (model pracuje výhradne s informáciami absorbovanými počas trénovania) a maximálnej dĺžke vstupnej sekvencie, ktorá limituje schopnosť pracovať s rozsiahlymi kontextmi (napr. určovanie podobnosti medzi skupinou príbehov vo forme kníh). V súčasnosti sú dostupné modely s obrovským kontextom, no aj napriek tomu stále nedokážu efektívne využívať informácie z celého dostupného kontextu [21].

Ďalším z možných prístupov k integrácií externých znalostí je Retrieval-Augmented Generation (RAG) [13]. Tento prístup spája generovanie textu (LLM) s vyhľadávaním v externých databázach. Pri spracovaní vstupu systém najprv identifikuje kľúčové koncepty, vyhľadá relevantné dokumenty a následne generuje výstup kombinujúci trénované znalosti s externými zdrojmi.

2.5 Sémantické vyhľadávanie

Cieľom vyhľadávania je na základe dopytu, ktorý môže mať podobu otázky resp. dokumentu efektívne identifikovať a získať relevantné resp. podobné dokumenty z korpusu.

Motiváciou pre sémantické vyhľadávanie sú problémy lexikálnych metód s vyhľadávaním významovo podobných, no lexikálne odlišných dokumentov a problémy s disambiguáciou, kedy je lexikálna zhoda výrazná, no kontext mení význam daného slova.

Základom sémantického vyhľadávania je projekcia textu – či už ide o dopyt alebo dokumenty v korpuse – do vysokodimenzionálneho vektorového priestoru, v angličtine označovaného ako *embedding space*. Cieľom tejto projekcie je, aby vzniknutá vektorová reprezentácia zachytávala sémantický význam textu, a to tak, že sémanticky podobné texty sú v tomto vektorovom priestore reprezentované geometricky blízkymi vektormi. Potom je možné určovať podobnosť medzi dvoma vetami, časťami dokumentov alebo celými dokumentmi na základe vzdialenosti medzi ich vektormi v priestore. K určeniu vzdialenosti sa používajú metriky vzdialenosti [63].

Medzi najpoužívanejšie modely určené na generovanie vektorových reprezentácií sú varianty vychádzajúce z modelu BERT [12], ktoré boli ďalej doladené na úlohách z domény sémantickej reprezentácie textu, tak, aby priestor vektorových reprezentácií mal požadované vlastnosti [46].

Ďalším podstatným problémom vyhľadávania je to, ako budeme vyhľadávať najbližších susedov. Naivné porovnávanie každého uloženého vektora s každým je neefektívne. Časová zložitosť tohto prehľadávania, kde sa dopyt porovnáva s každým dokumentom v databáze, rastie lineárne s počtom dokumentov, čo je pre rozsiahle databázy neprijateľné.

Vyhľadávanie najbližšieho suseda

Formálne, pre danú množinu N bodov $P = \{p_1, p_2, \ldots, p_N\}$ v metrickom priestore X (v našom prípade $\mathbb{R}^d$) a dopytovací bod $q \in X$, cieľom je nájsť bod $p_{true} \in P$, ktorý je najbližšie k q podľa zvolenej metriky vzdialenosti $d(p, q)$, t.j. $p_{true} = \arg\min_{p \in P} d(p, q)$ [2]. Tento základný problém sa rozširuje na nájdenie K najbližších susedov (k-Nearest Neighbors – k-NN), kde sa hľadá množina K bodov z P s najmenšími vzdialenosťami k q.

Ako už bolo naznačené, najjednoduchší prístup k riešeniu je úplné vyhľadávanie. Pri tomto prístupe sa pre daný dopyt q vypočíta jeho vzdialenosť od každého bodu $p_i \in P$ a následne sa vyberie bod s najmenšou vzdialenosťou. Časová zložitosť takéhoto vyhľadávania je typicky $O(N \cdot d)$, kde N je počet bodov v databáze a d je dimenzia vektorov (výpočet vzdialenosti má zložitosť $O(d)$). Pre rozsiahle databázy a vysokodimenzionálne vektory, typické pre sémantické reprezentácie, je tento prístup výpočtovo náročný a pomalý. Pre zrýchlenie sa využívajú dátové štruktúry, často označované ako indexy. Tieto indexy sú vybudované nad množinou dátových bodov P počas fázy predspracovania a ich úlohou je efektívne organizovať dáta tak, aby sa pri dopyte nemusela prehľadávať celá množina P, ale len jej malá, relevantná podmnožina. Efektivita, aj napriek predspracovaniu a redukcií porovnaní, je pri veľkom objeme dát a vysokodimenzionálnom priestore stále nepostačujúca [24]. Nájsť najbližšieho suseda vo vysokodimenzionálnom priestore efektívne si preto vyžaduje určité kompromisy.

Približné vyhľadávanie najbližších susedov

Vzhľadom na vysokú výpočtovú náročnosť exaktného vyhľadávania najbližších susedov vo vysokodimenzionálnych priestoroch, ktorá ho robí nepoužiteľným pre mnohé praktické ap-

likácie s požiadavkami na nízku latenciu, sa pozornosť presunula k metódam približného vyhľadávania najbližších susedov (k-Approximate Nearest Neighbor Search – k-ANN).

Základnou myšlienkou k-ANN je obetovať garanciu nájdenia skutočne najbližších susedov výmenou za zrýchlenie procesu vyhľadávania alebo zníženie pamäťových nárokov. Hlavným cieľom k-ANN je teda poskytnúť veľmi rýchle odpovede na dopyty, pričom vrátené výsledky sú s vysokou pravdepodobnosťou blízke optimálnym. Pre praktické aplikácie sú podstatné tieto aspekty:

- rýchlosť dopytu – ako rýchlo dostaneme odpoveď,
- kvalita približného vyhľadávania – pri vyhľadávaní K najbližších susedov sa určuje ako pomer nájdených skutočných najbližších susedov a K nájdených výsledkov,
- čas budovania indexu – ako dlho trvá príprava dátovej štruktúry,
- pamäťovú náročnosť – koľko pamäte index zaberie,
- schopnosť pracovať s dynamickými zmenami v dátach (pridávanie/mazanie bodov).

Existuje široké spektrum ANN algoritmov, ktoré sa líšia v princípoch, na ktorých sú založené (napr. metódy založené na randomizovaných kd-stromoch [3], hašovaní [24] alebo grafoch [36]), a v kompromisoch, ktoré ponúkajú medzi vyššie uvedenými aspektmi. Výber konkrétnej metódy závisí od špecifických požiadaviek danej aplikácie.

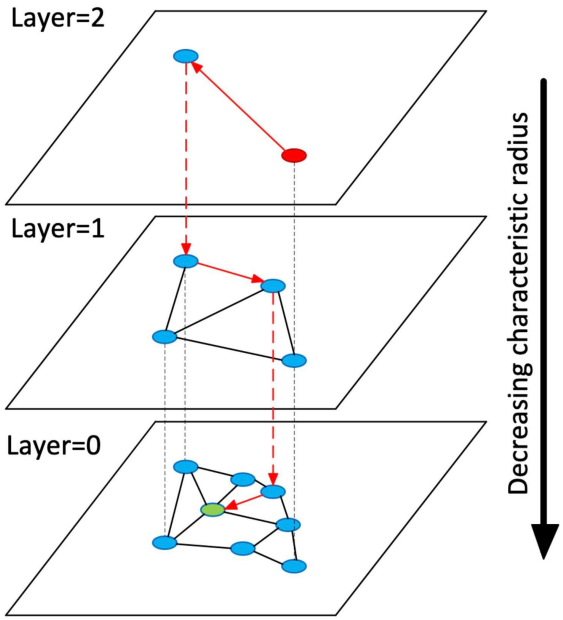
Hierarchický navigovateľný malý svet (Hierarchical Navigable Small World)

Jedným z často implementovaných algoritmov, resp. dátových štruktúr, pre približné vyhľadávanie najbližších susedov, je Hierarchical Navigable Small World (HNSW) [36]. Tento algoritmus patrí do kategórie grafových metód a je známy svojou vysokou efektivitou a presnosťou pri vyhľadávaní vo vysokodimenzionálnych priestoroch, ako aj schopnosťou efektívne zvládať budovanie indexu a dynamické vkladanie prvkov.

Základnou myšlienkou HNSW je konštrukcia viacvrstvovej grafovej štruktúry. Každá vrstva v tejto hierarchii predstavuje podmnožinu dátových bodov (vektorov), pričom najvyššia vrstva obsahuje najmenší počet bodov a najnižšia vrstva obsahuje všetky body z dátovej sady. Hrany v grafe spájajú navzájom blízke body. Vrcholy v horných vrstvách slúžia ako vstupné body a umožňujú rýchle preskakovanie dlhých vzdialenosti v priestore, zatiaľ čo vrcholy v nižších vrstvách umožňujú jemnejšie a presnejšie doladenie vyhľadávania v lokálnom okolí.

Vyhľadávanie podobného vektora k danému dopytu v HNSW prebieha nasledovne: začína sa vo vstupnom bode najvyššej vrstvy grafu. V každej vrstve sa iteratívne prechádza k susedom, ktorí sú bližšie k dopytovanému vektoru (greedy search), až kým sa nenájde lokálne minimum (bod, ktorého všetci relevantní susedia sú od dopytu ďalej). Následne sa proces opakuje v nasledujúcej, nižšej vrstve, pričom ako vstupný bod slúži nájdené lokálne minimum z predchádzajúcej vrstvy. Tento proces pokračuje až do najnižšej vrstvy (vrstva 0), kde sa vykoná finálne doladenie a nájdu sa kandidáti na najbližších susedov. Veľkosť dynamického zoznamu kandidátov počas vyhľadávania je parametrom, ktorý slúži na vyváženie presnosti a rýchlosti, pričom vyššia hodnota vedie k presnejšiemu, ale pomalšiemu vyhľadávaniu.

Proces vkladania nového vektora do HNSW štruktúry začína v najvyššej vrstve. Algoritmus náhodne určí maximálnu vrstvu, do ktorej bude nový bod vložený, pričom pravdepodobnosť pridelenia do vyššej vrstvy exponenciálne klesá. Následne sa v každej vrstve (od



Obr. 2.1: Znázornenie hierarchickej štruktúry vrstiev v algoritme HNSW. Červené šípky znázorňujú priechod grafom od vstupu (červený uzol) až k dotazu (zelený uzol). Prevzané z [36].

najvyššej pridelenej smerom nadol) vyhľadajú najbližší susedia k vkladanému bodu a vytvoria sa s nimi spojenia (hrany). Počet susedov, s ktorými sa bod spojí v každej vrstve, a veľkosť dynamického zoznamu kandidátov počas konštrukcie sú dôležité parametre ovplyvňujúce kvalitu grafu a rýchlosť jeho budovania. Vďaka tomuto inkrementálnemu prístupu HNSW podporuje dynamické vkladanie nových prvkov do už existujúceho indexu. Čo sa týka vymazávania prvkov, táto operácia je v HNSW a podobných grafových štruktúrach komplexnejšia. Častým riešením je zneplatnenie prvku, kedy je prvok označený ako neplatný a pri vyhľadávaní ignorovaný, ale fyzicky z grafu odstránený nie je okamžite, aby sa nenarušila konektivita. Dlhodobé hromadenie takto označených prvkov však môže viesť k degradácii výkonu a presnosti, preto niektoré implementácie ponúkajú stratégie pre periodickú rekonštrukciu alebo čistenie indexu.

2.6 Generovanie podporované vyhľadávaním

Generovanie podporované vyhľadávaním (retrieval-augmented generation, RAG [13]) je technika, ktorá kombinuje generatívne schopnosti jazykových modelov s vyhľadávaním relevantných textových častí v externej databáze. Pri RAG prístupe model nie je odkázaný iba na *parametrickú* pamäť (t. j. naučené váhy modelu), ale dokáže si kontext vyhľadať v *neparametrickej* pamäti (t. j. v indexovanej kolekcii dokumentov, prípadne častí textu). Na tieto časti je potom možné sa odkazovať v generovanom výstupe, čo umožňuje overenie výsledkov používateľom a prispieva tak k zvýšeniu dôveryhodnosti systému.

Naivný RAG

Naivný RAG predstavuje najjednoduchší variant, pozostávajúci z reťazca:

1. **indexovanie**: dokumenty sú predspracované, rozdelené na menšie bloky (tzv. *chunks*), ich vektorové reprezentácie (angl. embeddings) sa uložia do vektorovej databázy.
2. **vyhľadávanie**: na základe používateľskej otázky sa vyberú najrelevantnejšie bloky podľa podobnosti embeddingov.
3. **generovanie**: tieto bloky sa spolu s pôvodnou otázkou vložia do LLM a model generuje finálnu odpoveď.

Toto riešenie je častokrát postačujúce, no v praxi sa môžu vyskytnúť problémy s nepresným vyhľadávaním a nadbytočnými informáciami, ktoré zaberajú kontext.

Pokročilé RAG systémy

Pokročilý RAG systém rozvíja naivný prístup a pridáva **pre-retrieval** a **post-retrieval** fázy (viď stredná časť na obr. 2.2).

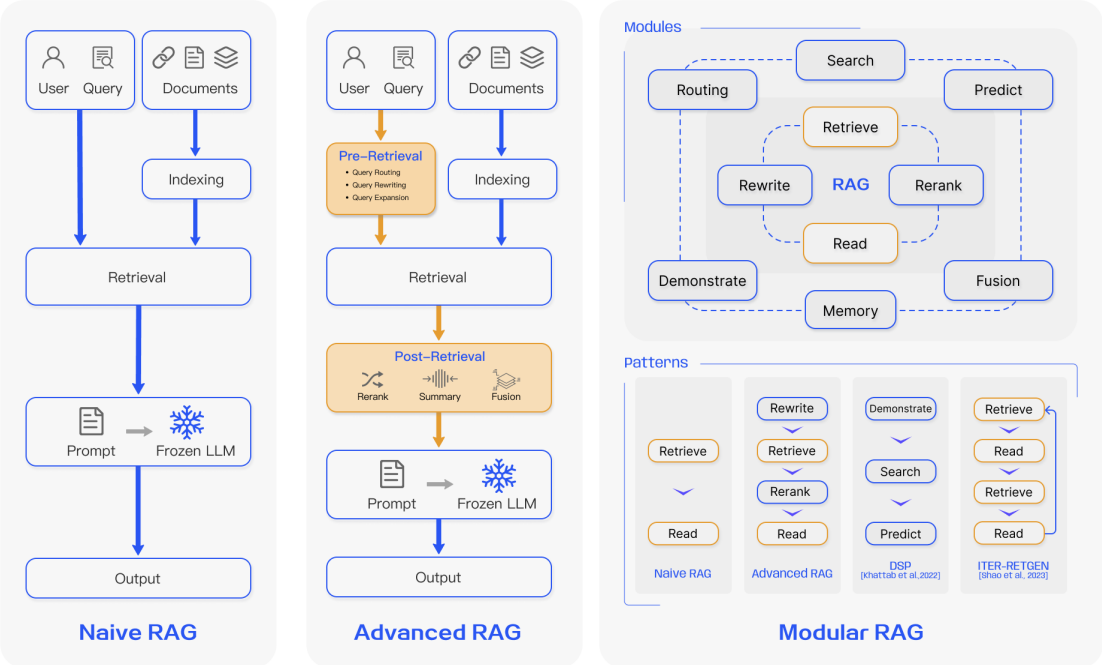
Pre-retrieval optimalizácie. Pomáhajú zlepšiť kvalitu indexovania a úpravu pôvodnej otázky. Môže ísť napr.o úpravu dotazu (tzv.*query rewriting*), doplnenie kľúčových slov (*query expansion*) alebo využitie metadát zdokumentov (napr. časové údaje). Lepšie definovaný dotaz pomáha získať presnejšie a konzistentnejšie výsledky v nasledujúcom kroku.

Post-retrieval spracovanie. V tejto fáze sa spracováva už vybraný kontext, typicky formou preusporiadania blokov podľa relevancie (*reranking*), prípadne *kompresie obsahu* (výber len najdôležitejších častí), aby sa pri generovaní predišlo zahlteniu modelu menej podstatnými informáciami. Takto zredukovaný a zostavený kontext sa následne použije vo finálnom promptovaní LLM.

Modulárne RAG systémy

Modulárne RAG systémy ponúkajú väčšiu flexibilitu tým, že sa systémy skladajú z viacerých špecializovaných modulov (pravý blok na obr. 2.2):

- **Search** (*vyhľadávanie*): umožňuje priamu interakciu so špecifickými databázami či internetovými službami.
- **Routing** (*triedenie dotazov*): rozhoduje, či je otázka smerovaná do klasického vyhľadávania, do doplnkovej úlohy sumarizácie, alebo do iného modulu.
- **Memory** (*pamäť*): slúži na iteratívne obohacovanie modelu o informácie, ktoré sa využijú pri ďalšom vyhľadávaní.
- **Predict** (*predikcia/generovanie*): generuje časti kontextu priamo cez LLM, čím minimalizuje potrebu neustáleho vyhľadávania.
- **Fusion** (*fúzia obsahu*): kombinuje viacero častí kontextu, aby vznikla jednotná a konzistentná odpoveď.



Obr. 2.2: Existuje mnoho techník ako zlepšiť RAG a zvýšiť tak presnosť celého systému. Obrázok prezentuje tri základné kategórie RAG systémov (prevzané z [13]).

2.7 Optimalizácia LLM systému na základe textu

Napriek pokročilým schopnostiam LLM, najmä v oblasti učenia z kontextu, ich výkonnosť a spoľahlivosť zostávajú citlivé na spôsob, akým sú inštruované. Kvalita výstupu LLM závisí nielen od samotného modelu a dát, ale aj od presnosti a zrozumiteľnosti vstupného textu, teda promptu.

Prompt inžinierstvo

Prompt inžinierstvo (z angl. Prompt Engineering) [47] je proces navrhovania, formulovania a iteratívneho zlepšovania promptov s cieľom maximalizovať presnosť a relevanciu výstupov generovaných LLM pre danú úlohu. Ako bolo spomenuté pri zero-shot a few-shot učení, aj malé zmeny vo formulácii inštrukcií, štruktúre príkladov či poradí informácií v prompte môžu dramaticky ovplyvniť správanie modelu [34, 38].

Manuálne prompt inžinierstvo, hoci intuitívne, naráža na viaceré obmedzenia, obzvlášť pri zložitejších aplikáciách [61, 62]:

- **časová náročnosť:** návrh a testovanie optimálnych promptov je často založené na metóde pokus-omyl a vyžaduje značné úsilie, najmä pri nových úlohách alebo modeloch,
- **citlivosť na formuláciu:** jemné variácie v slovách alebo štruktúre promptu môžu viesť k nepredvídateľným zmenám vo výstupoch, čo sťažuje systematické zlepšovanie,
- **zložité závislosti:** v komplexných systémoch (napr. multi-hop RAG alebo agentné systémy), ktoré kombinujú viacero volaní LLM s externými nástrojmi (ako retrievery

alebo formátovače dát), sa manuálne ladenie promptov pre každý modul stáva neprehľadným a náchylným na chyby. Závislosti medzi komponentmi môžu byť zložité a ich optimalizácia vyžaduje holistický prístup.

Tieto výzvy motivovali výskum v oblasti automatizácie procesu tvorby a optimalizácie promptov, známej ako *Automatické prompt inžinierstvo* (APE).

Textová automatická diferenciácia

Jedným z prístupov k APE, inšpirovaným z numerického gradientného zostupu (angl. gradient descent) a algoritmu spätnej propagácie (angl. backpropagation), je *textová automatická diferenciácia*. Rámce ako **TextGrad** [62] a **LLM-AutoDiff** [61] prenášajú princípy optimalizácie založenej na gradientoch do textovej domény. Využívajú LLM na generovanie a propagáciu textovej spätnej väzby – textových gradientov – skrz komplexnými LLM systémami.

Formálna definícia problému. Cieľom APE je nájsť optimálnu sadu promptov $\{\mathcal{P}_i\}^*$ pre všetky relevantné LLM komponenty M_i v systéme, modelovanom ako orientovaný graf $\mathcal{G} = (\mathcal{N}, \mathcal{E})$. Optimalizácia sa riadi minimalizáciou celkovej textovej stratovej funkcie $\mathcal{L}$, ktorá môže zahŕňať viacero čiastkových cieľov $\mathcal{L}_{T_j}$ pre rôzne podúlohy T_j (napr. kvalita vyhľadávania a kvalita finálnej odpovede v RAG systéme) [61],

$$\{\mathcal{P}_i\}^* = \arg\min_{\{\mathcal{P}_i\}} \bigcup_j \mathcal{L}_{T_j}\left(\mathcal{P}_{\mathcal{N}_{T_j}}\right) \tag{2.5}$$

kde $\mathcal{P}_{\mathcal{N}_{T_j}}$ je podmnožina promptov ovplyvňujúcich podúlohu T_j.

Výpočtový graf a dopredný priechod. Základom je modelovanie systému ako výpočtového grafu $\mathcal{G}$. Uzly $\mathcal{N}$ predstavujú komponenty systému – môžu to byť volania LLM, funkčné moduly (retriever, formátovač dát, deduplikátor) alebo špeciálne uzly pre výpočet straty [61]. Hrany $\mathcal{E}$ definujú tok dát a závislosti medzi komponentmi, pričom graf môže obsahovať aj cykly (napr. pri iteratívnych agentoch alebo multi-hop RAG). Textové vstupy, najmä prompty $\mathcal{P}_i$, sú považované za *trénovateľné parametre*.

Počas *dopredného priechodu* (forward pass) sa graf vykonáva v topologickom poradí (alebo sa cykly rozvinú). Každý uzol v vypočíta svoju hodnotu na základe hodnôt svojich predchodcov: $v = f_v(\text{PredecessorsOf}(v))$. Zároveň sa dynamicky buduje *graf parametrov* $\mathcal{G}_p$, ktorý sleduje presné závislosti medzi všetkými medzivýsledkami a parametrami [61]. Na konci sa vypočíta hodnota textovej straty $\mathcal{L}$ (napr. pomocou LLM evaluátora, ak nie je dostupný referenčný výstup alebo štandardnej metriky).

Spätné šírenie textových gradientov (Backward Pass). Po doprednom priechode nasleduje *spätný priechod* (backward pass), ktorý propaguje textovú spätnú väzbu (text. gradienty) od uzla straty $\mathcal{L}$ smerom k vstupným parametrom $\mathcal{P}_i$. Textový gradient pre ľubovoľnú premennú v v grafe $\mathcal{G}_p$ sa vypočíta rekurzívne agregovaním ($\cup$) spätnej väzby od všetkých jej priamych nasledovníkov $w \in \text{SuccessorsOf}(v)$.

$$\frac{\partial \mathcal{L}}{\partial v} = \bigcup_{w \in \text{SuccessorsOf}(v)} \nabla_f(v, w, \frac{\partial \mathcal{L}}{\partial w}) \tag{2.6}$$

kde ∇_f je operátor textového gradientu špecifický pre funkciu f, ktorá vytvorila w z v. Tento operátor využíva hodnoty v, w a už vypočítaný gradient pre w ($\partial\mathcal{L}/\partial w$) na generovanie spätnej väzby pre v.

Implementácia ∇_f závisí od typu funkcie f:

- **LLM volania (f = LLM):** operátor ∇_{LLM} využíva pomocný, *spätnoväzbový LLM* ($\text{LLM}_{\text{backward}}$) [61, 62]. Tento LLM dostane prompt obsahujúci kontext (napr. vstup v, výstup w) a gradient nasledovníka ($\partial\mathcal{L}/\partial w$) a jeho úlohou je vygenerovať textovú kritiku alebo návrhy, ako upraviť v, aby sa zlepšil výsledok vzhľadom na $\mathcal{L}$.
- **funkčné uzly (napr. retriever, formátovač):** tieto uzly nemajú vlastné textové parametre na optimalizáciu. Spätná väzba nimi často len „preteká“ (*pass-through gradients*). Ak má funkčný uzol w len jedného predchodcu v, tak $\partial\mathcal{L}/\partial v = \partial\mathcal{L}/\partial w$. Ak má viac predchodcov, môže byť potrebný LLM na atribúciu chyby, alebo sa gradient jednoducho prenesie na všetkých relevantných predchodcov.
- **cyklické volania:** aby sa zachovala kauzalita, LLM-AutoDiff používa *časovo-sekvenčné gradienty*, kde každé volanie uzla v cykle dostane gradient s časovou značkou t, a tieto sa agregujú usporiadane.
- **štruktúrované prompty (peers):** rôzne časti promptu (inštrukcia, príklady, formát) môžu byť definované ako samostatné peer parametre. Spätnoväzbový LLM potom môže generovať cielenú spätnú väzbu pre všetkých naraz, pre každý peer zvlášť, čím sa predchádza ich zmiešaniu a zvyšuje presnosť optimalizácie[2].

Aktualizácia parametrov (Textual Gradient Descent – TGD). Po vypočítaní agregovaných textových gradientov $\partial\mathcal{L}/\partial\mathcal{P}_i$ pre všetky optimalizovateľné prompty $\mathcal{P}_i$ nasleduje krok aktualizácie. Využíva sa *optimalizačný LLM* (LLM_{opt}), ktorý funguje ako analógia gradientného zostupu (TGD) [62]:

$$\mathcal{P}_{i,\text{new}} = \text{TGD.step}(\mathcal{P}_i, \frac{\partial\mathcal{L}}{\partial\mathcal{P}_i}) \equiv \text{LLM}_{\text{opt}}(\mathcal{P}_i, \text{GradientContext}(\mathcal{P}_i), \frac{\partial\mathcal{L}}{\partial\mathcal{P}_i}) \tag{2.7}$$

Optimalizačný LLM dostane aktuálny prompt $\mathcal{P}_i$, jeho textový gradient $\partial\mathcal{L}/\partial\mathcal{P}_i$ a ďalší kontext (napr. popis role, história predchádzajúcich návrhov, informácie o systéme[3].) a jeho úlohou je navrhnúť novú, vylepšenú verziu promptu $\mathcal{P}_{i,\text{new}}$. Tento proces sa iteratívne opakuje, často v spojení s validačnou množinou dát na rozhodnutie, či sa nový prompt prijme.

[2] Autori [61] na základe ich experimentov upozorňujú na lost-in-the-middle problém LLM s veľkým kontextom, ktoré sú schopné spracovať veľa príkladov naraz, no kvalita spätnej väzby klesá pri zaplnení kontextu, z čoho vyplýva potreba rozdeliť a spracovať prompt po častiach.

[3] Bližšie špecifikované v sekcii 3.5 v práci [61]

Kapitola 3

Určovanie podobnosti príbehov

Určovanie podobnosti príbehov je zložitý proces, ktorý zahŕňa nielen sémantické a lexikálne podobnosti textov, ale aj hlbšie aspekty, ako sú štruktúra príbehu, charakterové vzťahy, vývoj deja či emocionálny dopad na čitateľa. Jedným z kľúčových problémov v tejto oblasti je pochopiť, ktoré prvky príbehu ľudia intuitívne považujú za relevantné pri posudzovaní podobnosti príbehov. Táto otázka si vyžaduje interdisciplinárny prístup, ktorý zahŕňa poznatky z naratológie, psychológie a spracovania prirodzeného jazyka (NLP).

Aspektov, na základe ktorých je možné určovať podobnosť príbehov, je nepredstaviteľne veľa [50]. Pri analýze príbehu sa NLP metódy zameriavajú aj na praktické využitie existujúcich konceptov z naratológie, ktorá poskytuje určité definície toho, čo všetko tvorí príbeh alebo skôr naratív – pojem označujúci obsah (príbeh), ako aj to, akým spôsobom je obsah komunikovaný.

V nasledujúcich sekciách kapitoly sú predstavené pohľady naratológie na to, čo sú podstatné aspekty naratívu a taktiež sú načrtnuté možnosti ako pristúpiť k určovaniu podobnosti príbehov v NLP.

3.1 Príbeh z pohľadu naratológie

Naratológia skúma to, čo všetky naratívy, majú spoločné, ako aj to, čo im umožňuje odlišovať sa od seba, a snaží sa charakterizovať relevantný súbor pravidiel a noriem riadiacich tvorbu a spracovanie naratívov. Hlavným prínosom využitia existujúcich teoretických rámcov je redukcia a zameranie sa na konečný počet vlastností príbehu, ktoré podľa slov ich autorov poskytujú postačujúci systém pre pochopenie, interpretáciu a hľadanie spoločných čŕt príbehov. Teoretické modely, na ktorých stoja mnohé prístupy v porozumení naratívov, sú rôznorodé.

Podľa štrukturálnej naratológie, každý naratív pozostáva z dvoch častí: príbehu (histoire), teda obsahu alebo reťazca udalostí (činy, udalosti deja) a takzvaných existencií (postavy, prvky prostredia); a z diskurzu (discours), čo je vyjadrenie, prostriedok, ktorým sa obsah komunikuje. Zjednodušene povedané, príbeh je to „čo“ je v naratíve zobrazené, diskurz je to „ako“ [6]. Genettov model [14], založený na perspektívnej podstate naratívu (t. j. nevyhnutnosti rozprávača), túto štruktúru rozširuje o naráciu (narrating), ktorá explicitne zdôrazňuje rolu rozprávača pri formovaní informácií. Genette navyše definuje tri kľúčové vzťahy medzi troma základnými prvkami: *čas* (tense), ktorý analyzuje usporiadanie udalostí (chronológia, retrospektíva, časové posuny); *modus* (mood), ktorý skúma rovnováhu

medzi dejovosťou a popisom; a *hlas* (voice), ktorý určuje perspektívu (fokalizáciu) a spôsob interakcie postáv.

Ďalším používaným rámcom je Proppova morfológia ľudových rozprávok [15], vytvorená na základe analýzy ruských rozprávok. Propp identifikoval univerzálne charakterové roly a ich typické funkcie, ktoré sa v príbehoch opakujú. Táto reprezentácia príbehov našla praktické využitie hlavne pri generovaní naratívov [20], hoci je do určitej miery obmedzená na pevne stanovený súbor funkcií a stereotypov postáv. Prístup dekompozície príbehu na opakujúce sa elementy príbehu (t.j. morálne posolstvá, role postáv a opakujúce sa scény) je veľmi populárnym prístupom k analýze príbehov a existujú komunitné fóra, ktoré poskytujú takúto analýzu a sú bohatým zdrojom dát (viď sekcia 6.4).

Post-klasická naratológia kladie väčší dôraz na spoločenský a kultúrny rozmer príbehov. Herman [19] označuje tento kontext ako *situovanosť*, ktorá môže výrazne formovať tvorbu i vnímanie naratívov. Príbehy tak často nadobúdajú význam vďaka hodnotám, normám alebo emocionálnej reakcii publika. Tento aspekt zdôrazňujú aj Shen a kol. [49], podľa ktorých môžu byť dva príbehy považované za podobné predovšetkým vtedy, ak rezonujú s čitateľom (publikom) na rovnakej morálnej či emocionálnej úrovni, hoci sa z hľadiska štrukturálnej analýzy vôbec nemusia podobať.

Táto rôznorodosť rámcov a prístupov ukazuje, že naratológia nespočíva v jedinom univerzálnom spôsobe pre analýzu príbehov. Naopak, ide o komplexnú sústavu konceptov, v ktorých sa prelínajú štrukturálne prvky (dej, postavy, prostredie, perspektíva rozprávača) s kontextuálnymi a emocionálnymi dimenziami (kultúra, spoločenské normy, morálne hodnoty či emocionálna rezonancia). Tento fakt sa podpisuje aj na nejednotnej definícii úloh v oblasti počítačového porozumenia naratívov, kde každá metóda využíva vlastný pohľad na naratív [43].

V minulosti mohlo byť náročné a pracné prepojiť všetky tieto koncepty do jedného celistvého systému, no pokrok v technológiách spracovania prirodzeného jazyka — najmä vo vývoji veľkých jazykových modelov — umožňuje takéto komplexné riešenia prakticky aplikovať. Ako ukazujú Chun [10] a ďalší autori [42], vďaka LLM je dnes možné použiť generatívny prístup a úspešne zvládať komplexnú štrukturálnu analýzu príbehov vychádzajúcu z teoretických konceptov naratológie. Aplikovanie automatizovaných metód implementujúcich tieto teoretické rámce na obsiahlych korpusoch naratívov, tiež umožňuje vyhodnotiť ich relevanciu a validitu [42].

3.2 Smery v oblasti porovnávania príbehov

Praktické metódy porovnávania príbehov v spracovaní prirodzeného jazyka vychádzajú z naratologických konceptov (pozri sekcia 3.1) a zohľadňujú štrukturálne, sémantické, emotívne aj spoločenské aspekty textu. V nasledujúcich podsekciách uvediem niekoľko základných smerov v oblasti určovania podobnosti príbehov.

Metódy zamerané na vyhľadávanie

Cieľom týchto metód je na základe referenčného príbehu efektívne vyhľadávať v kolekcii textov také naratívy, ktoré s ním vykazujú najvyššiu mieru podobnosti — či už z hľadiska štruktúry deja, tém, posolstiev alebo lexikálnej zhody. V súčasnosti sa pri týchto úlohách využívajú modely určené pre vytváranie vektorových reprezentácií dokumentov, ktoré transformujú vstupný text do vektorov (tzv. embeddings) so schopnosťou zachytiť sémantickú hĺbku, kontext a aj lexikálne vlastnosti.

Hatzel a Biemann [17], sa zamerali na význam deja a snažia sa potlačiť vplyv špecifických detailov príbehu (mená postáv, prostredie a pod.). Ich cieľom je hľadať moderné adaptácie známych príbehov a využiť pri tom vektorovú reprezentáciu textu z veľkého jazykového modelu. Ich metóda je založená na kontrastívnom učení. Pri tvorbe datasetu navrhujú použiť pseudorandomizáciu štrukturálnych vlastností príbehov. Takto adaptovanú reprezentáciu textu nazývajú – *narrative embeddings*. Ich výsledky ukazujú, že ich prístup k tvorbe príbehových vektorových reprezentácií výrazne pomáha v úlohách typu „vyhľadaj príbeh s podobným dejom“, no nevyhodnocujú ďalšie podobnosti medzi príbehmi, jedná sa teda o holistické určenie podobnosti.

Shen a kol. [49] definujú koncept *empatickej podobnosti* (empathic similarity) a vysvetľujú, že pri vnímaní podobných príbehov nejde len o podobnosť medzi štrukturálnymi prvkami príbehov, ale hlavne o zhodu emočnej reakcie. Zdôrazňujú, že príbehy môžu byť veľmi podobné najmä vtedy, keď vzbudzujú rovnakú emocionálnu odozvu publika či podobné morálne ponaučenie/posolstvo. Tieto tvrdenia vyhodnotili na nimi vytvorenom datasete EmpathicStories, ktorý poskytuje anotácie štrukturálnych vlastností a emocionálnej rezonancie medzi pármi príbehov. Túto sadu napokon využili na dotrénovanie modelu pre vyhľadávanie dokumentov, čo výrazne zvýšilo koreláciu medzi výstupmi modelu a ľudským vnímaním v porovnaní so základnou verziou modelu.

Tématické klastrovanie

V literárnej teórii je žáner bežne vnímaný ako skupina textov zjednotených podľa spoločných tematických čŕt (napr. detektívka, fantasy, sci-fi, román). Keďže definícia žánrov nie je celkom jednoznačná (text môže patriť do viacerých žánrov, prípadne obsahovať inovatívne/prechodné prvky), ich automatická identifikácia zostáva náročnou úlohou.

V nedávnej štúdii Sobchuk a Šeľa [50] predstavujú systematické porovnávanie textov na základe tématickej analýzy. K tematickej analýze pristupujú ako k úlohe bez učiteľa a používajú *klastrovanie*, ktoré zaraďuje texty do skupín na základe ich podobnosti bez priameho použitia trénovacích dát. Výsledkom ich práce je systém, ktorý dokáže rozdeliť kolekciu príbehov do skupín, ktoré zodpovedajú žánrom. Zaujímavé je zistenie autorov, že *bag-of-words* prístup (v kombinácii s primeraným predspracovaním) k vytvoreniu vektorovej reprezentácie príbehov dosiahol veľmi slušné výsledky a niekedy prekonal zložitejšie algoritmy (doc2vec [31] a Latent Dirichlet Allocation (LDA) [5]). V štúdii však chýba porovnanie so súčasnými metódami ako Sentence-BERT [46] alebo inými veľkými jazykovými modelmi.

Analýza na úrovni základných elementov príbehu

Nové metódy aplikujú komplexnejšiu analýzu príbehu s cieľom porovnať príbehy pomocou viacerých elementov, pričom integrujú teoretické základy naratológie. Tomuto smeru napomáha zlepšenie veľkých jazykových modelov, ktoré dokážu nielen extrahovať základné informácie o deji a postavách, ale aj generovať a hodnotiť celé príbehy, a to na základe „promptingu“.

Prompting je metóda, pri ktorej sú jazykové modely usmerňované prostredníctvom špecifických vstupných pokynov alebo otázok, aby dosiahli požadované výstupy, ako napríklad identifikáciu tém, postáv, udalostí či holistickú analýzu štruktúry príbehu.

Chun [10] napríklad prezentuje metódu porovnávania príbehov (AIStorySimilarity) na základe štyroch elementov naratívu: postáv, deja, prostredia a témy, ktoré v rámci seba zahŕňajú ešte 31 aspektov. Metóda je postavená na LLM, ktorý pri porovnávaní párov

príbehov extrahuje ich tematické a naratívne prvky a následne porovnáva ich špecifické podobnosti na úrovniach jednotlivých elementov, ale aj celkovej podobnosti páru príbehov.

Podobne metóda FaNS (Facet-based Narrative Similarity) [1] stavia práve na takomto detailnom rozčlenení príbehu pomocou veľkých jazykových modelov. Inšpiráciu možno vidieť vo všeobecne známom modelovaní udalostí cez otázky *5W1H*: *Kto* (Who), *Čo* (What), *Kedy* (When), *Kde* (Where), *Prečo* (Why) a *Ako* (How). Najprv sa z textu vyextrahujú kľúčové aspekty (t.j. postavy, doba, miesto, dôvody konania, spôsoby realizácie), a až potom sa porovnáva, do akej miery sú tieto komponenty zhodné. Výsledkom je metrika podobnosti, ktorá preukázala výrazne lepšiu zhodu s ľudským posúdením ako tradičné prístupy na základe sémantickej podobnosti textu.

Detekcia naratívnych trópov

V literárnej praxi *trópy* predstavujú ustálené, často opakované prvky alebo motívy, vďaka ktorým možno príbehy rýchlo zaradiť či porovnať. Patria medzi ne známe situácie (napr. detektív rieši záhadu vraždy), postavové archetypy (*mentor*, *nechcený hrdina*) či literárne prostriedky (napr.*flashback*).

Tieto trópy môžu fungovať ako „skratky“ pre čitateľa i autora — vopred vytvárajú očakávania ohľadom typu deja, atmosféry či vývoja postáv. Zároveň tak tvoria *bázu* pre porovnávanie príbehov, pretože ak dve diela obsahujú podobné sady tropov (napr. *zakázaná láska*, *magický artefakt*, *záchrana sveta*), čitatelia ich budú vnímať ako žánrovo či tematicky príbuzné.

Detekcií tropov na základe webovej stránky TVTropes (viď. sekcia 6.4) sa venuje práca Chaudhary a Jhala [8], ktorí ukazujú, že hoci komunitné definície tróp v odlišných médiách môžu byť nejasné, aj základné metódy (*bag-of-words*, *doc2vec*) dokážu predikovať ich výskyt v scenároch filmov.

Kapitola 4

Návrh riešenia

Predchádzajúca kapitola detailne preskúmala mnohorozmernú povahu podobnosti príbehov a odhalila, že jej určovanie je komplexná a inherentne subjektívna úloha. Ukázalo sa, že neexistuje jediná univerzálna metrika ani jednotný teoretický rámec, ktorý by dokázal obsiahnuť všetky relevantné dimenzie. Práve táto absencia jednoduchej, kvantifikovateľnej metriky a rozmanitosť prístupov k analýze, ktoré sa nesnažia o jedinú definitívnu odpoveď, viedla k zameraniu sa na návrh systému, ktorý umožňuje exploratívne hľadať a skúmať príbehy, pričom indikátorom podobnosti je sémantická podobnosť medzi významnými prvkami príbehu.

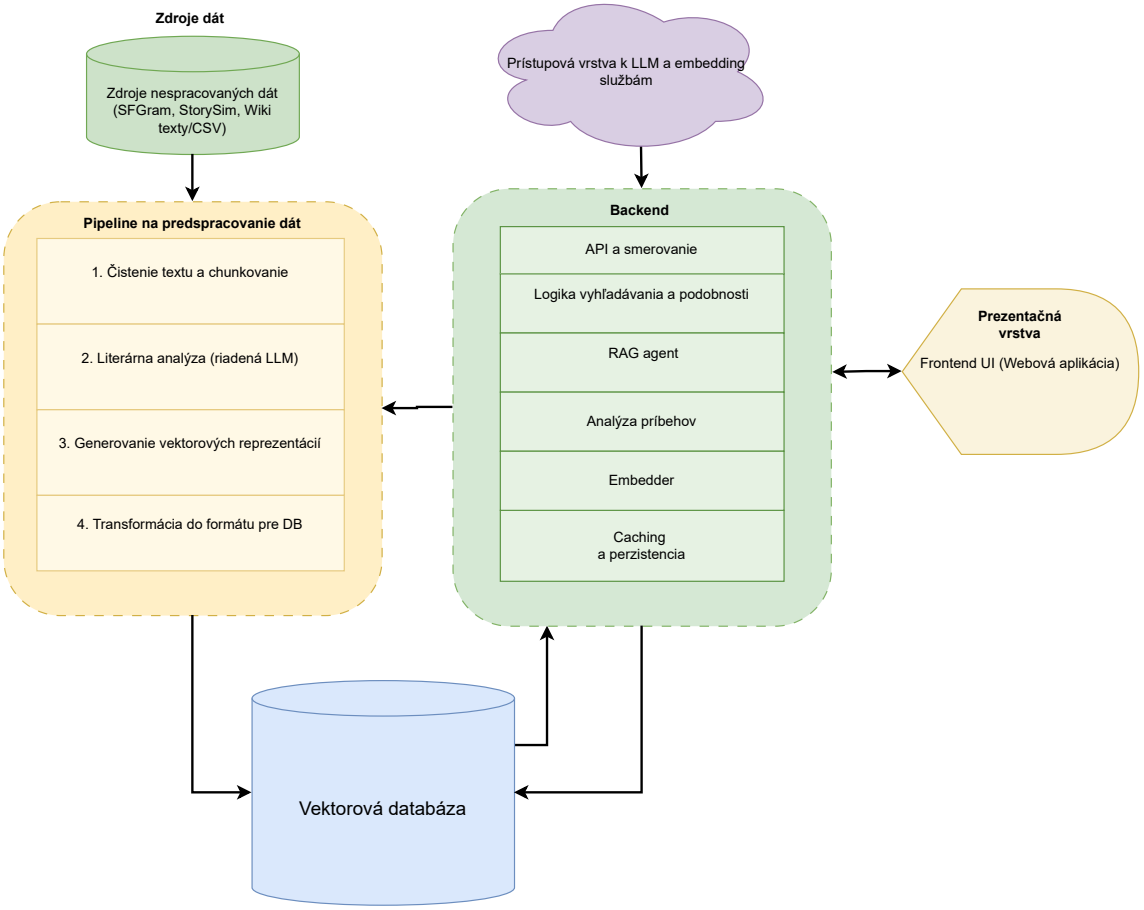
Cieľom práce je teda vytvoriť systém na vyhľadávanie podobných príbehov a s tým spojené určovanie podobnosti príbehov s podobnými charakteristikami pomocou sémantickej podobnosti v korpuse analyzovaných textov príbehov. Sekcia 4.3 popisuje analýzu a reprezentáciu príbehov v systéme. Aspektovo orientovaná analýza príbehov, vo forme sumarizácie podstatných udalostí, rieši problém hľadania paralel v navonok odlišných príbehoch. Navrhnutá reprezentácia umožňuje porovnávať a vyhľadávať príbehy na základe deja, postáv, prostredia a motívov príbehu. Ďalším dôležitým prvkom systému je databáza s podporou vektorového vyhľadávania, čo bližšie popisuje sekcia 4.4. Posledná sekcia popisuje východiska návrhu a samotné navrhnuté používateľského rozhranie systému.

4.1 Systém pre vyhľadávanie a porovnávanie príbehov

Primárnym cieľom navrhovaného systému je umožniť používateľom efektívne vyhľadávať príbehy, ktoré sú podobné zadanému dopytu.

Kľúčovým bodom v procese návrhu takéhoto systému je návrh vhodnej reprezentácie príbehov, ktorá by umožnila predovšetkým zmysluplné porovnávanie a vyhľadávanie. Problémom je to, ako definovať a merať podobnosť medzi príbehmi, ktoré môžu byť komplexné a viacvrstvové. Pre tento účel bola ako základná metóda zvolená analýza a extrakcia významných fragmentov príbehu a ich transformácia do vektorovej reprezentácie. Vzdialenosť medzi týmito vektormi, vypočítaná pomocou zvolenej metriky (napr. kosínusová podobnosť), potom slúži ako kvantitatívna miera sémantickej podobnosti.

Prípravu dát do tejto vektorovej formy, vrátane samotnej analýzy a extrakcie relevantných informácií z pôvodných textov príbehov, zabezpečujú moduly predspracovania. Tieto moduly sú zodpovedné za transformáciu textov na štruktúrované dáta, ktoré následne slúžia ako vstup pre generovanie sémantických vektorov.



Obr. 4.1: Diagram znázorňujúci podstatné časti systému a tok dát.

Pri počiatkoch návrhu systému pre určovanie podobnosti bol hlavným komponentom RAG systém, ktorého úlohou bolo určiť podobnosť metódou odpovedania na otázky v doméne rozsiahlych naratívnych textov (kníh, esejí), kde by používateľ špecifikoval hľadaný aspekt podobnosti a systém by mu poskytol súhrn odcitovaných podobností a paralel zo zdrojových textov. Po otestovaní výpočtovej náročnosti a efektivity tohto prístupu bol pôvodný návrh prepracovaný a RAG systém je jedným modulom systému a jeho úlohou je využívať sémantické vyhľadávanie v korpuse analyzovaných príbehov a poskytnúť používateľovi sumarizovaný prehľad výsledkov vzhľadom na zadaný dopyt a ním definované podobnosti.

Ďalšou časťou systému je vektorová databáza a s ňou spojené klientské moduly, ktoré zabezpečujú funkcionalitu ukladania a vyhľadávania. Poslednou časťou systému je prezentačná vrstva, ktorá zabezpečuje interaktívne používateľské rozhranie formou webovej aplikácie.

4.2 Zdroje dát

Podstatným rozhodnutím pre návrh systému je povaha textovej reprezentácie príbehov a voľba ich zdrojov. Ako potenciálne zdroje prichádzajú do úvahy napríklad knihy, odborné či spravodajské články, reportáže, príspevky na komunitných fórach alebo popisky titulov z filmových či literárnych databáz. Pre identifikáciu relevantných a netriviálnych podob-

ností je dôležité, aby bol vstupný korpus čo najrozsiahlejší a pokrýval príbehy rozmanitých žánrov.

Nakoľko cieľom systému je analýza a porovnávanie príbehov na nižšej úrovni – teda vyžaduje sa zachytenie podstatných tém, udalostí alebo postáv – stručné anotácie, ako sú napríklad popisy titulov v databáze IMDb[1], či iné veľmi krátke zhrnutia, neposkytujú dostatok informácií pre potreby zamýšľanej analýzy. Na druhej strane, spracovanie kompletných rozsiahlych diel, akými sú celé knihy, by bolo výpočtovo mimoriadne náročné a nákladné. Tvorba diverzifikovaného korpusu kníh by bola výrazne sťažená aj licenčnými podmienkami. Výber by sa totiž musel obmedziť prevažne na diela s verejnou licenciou.

Z týchto dôvodov sa táto práca primárne zameriava na využitie dejových opisov (synopsií) príbehov, ktoré sú vo veľkej miere dostupné na platforme Wikipédia[2]. Tieto texty predstavujú optimálny kompromis: ponúkajú podstatne viac detailov a kontextu než veľmi krátke anotácie, čím umožňujú hlbšiu analýzu, avšak sú stále výrazne stručnejšie a menej náročné na spracovanie než kompletné diela. Zároveň sa predpokladá, že opisy dejov z Wikipédie sú spracované tak, aby neobsahovali nadbytočné množstvo irelevantných detailov. Tento fakt by mohol prispieť k tomu, že jazykový model bude schopný efektívnejšie identifikovať nosné témy a zmysluplné tematické celky príbehu. Kapitola 6 ďalej rozoberá podrobnú špecifikáciu dátových sad a proces ich tvorby.

4.3 Reprezentácia a určovanie podobnosti príbehov

Pri návrhu reprezentácie príbehov, bolo preskúmaných niekoľko stratégií transformácie dlhších dokumentov do vektorovej reprezentácie. Pri transformácii textu do vektorových reprezentácií sa dnes do veľkej miery využívajú aj autoregresívne LLM [54]. Obvykle ide však o modely s väčším počtom parametrov (7 miliárd a viac), ktoré majú vyššie požiadavky na zdroje a keďže systém má poskytnúť interaktívne vyhľadávanie, s čím súvisí potreba rýchlej transformácie dopytu pri interakcii so systémom, sa táto práca venuje použitiu modelov s menším počtom parametrov (modely vychádzajúce z pôvodnej architektúry BERT [12]), ktoré aj napriek tomu, stále stoja na popredných priečkach v rebríčku porovnania Massive Text Embedding Benchmark – MTEB[3], ktorá na širokej škále datasetov vyhodnocuje kvalitu vektorových reprezentácií.

Jedným z hlavných problémo je obmedzená dĺžka vstupnej sekvencie pre embedding modely. To komplikuje spracovanie zdrojových textov príbehov, ktoré sú častokrát dlhšie ako toto kontextové okno. Dlhší príbeh sa preto musí segmentovať a spracovať segmenty izolovane. V súčasnosti rastie veľkosť kontextu, no s dlhším kontextom rastú kvôli mechanizmu pozornosti kvadraticky aj požiadavky na zdroje, preto je stále výhodné pracovať s kratšími segmentami a dokument reprezentovať sekvenciou vektorov. Ďalším dôvodom, prečo segmentovať na menšie časti, je granularita vyhľadávania informácií.

Segmentácia textu a kontext

Segmentáciou textu na kratšie, častokrát prekrývajúce sa, segmenty, dochádza k strate kontextu a teda k zmene sémantického významu daného segmentu. Zachovanie kontextu a dekompozícia textu do kratších ucelených sémantických jednotiek je netriviálny problém.

[1] Databáza filmov IMDb: https://www.imdb.com/.

[2] Manuál pre tvorbu synopsie na Wikipédii: https://en.wikipedia.org/wiki/Wikipedia:How_to_write_a_plot_summary.

[3] Massive Text Embedding Benchmark: https://huggingface.co/blog/mteb.

Riešením môže byť využitie štruktúry dokumentu (kapitoly, sekcie a pod.) k deleniu na sémanticky súdržné sekcie, no nedá sa však aplikovať na neštrukturované zdroje a stále chýba globálny kontext, ktorý môže byť dôležitý vzhľadom na určitý dopyt. Ďalším smerom riešenia je predspracovanie segmentov pomocou LLM sumarizácie celého dokumentu do krátkej formy a pripojiť tento kontext ku každému segmentu.

Úroveň granularity vyhľadávania

Ďalším problémom pri vytváraní vektorových reprezentácií je úroveň granularity, ktorú chceme zachovať pre vyhľadávanie. Zjednodušene možno konštatovať, že použitie dlhších segmentov vedie k nižšej granularite vyhľadávania, kým kratšie segmenty poskytujú možnosť vyhľadávania s vyššou granularitou. Toto je dôležite pre porovnávanie príbehov na zamýšľanej nižšej úrovni podobnosti ako len celkovej podobnosti príbehov.

Metóda neskorej interakcie

Zaujímavým riešením je použitie techniky neskorej interakcie [28]. Základnou myšlienkou tejto techniky je reprezentovať text, nie jedným vektorom, ale vektormi pre každý token zo vstupnej sekvencie, čo vytvára priestor pre porovnávanie na nižšej úrovni granularity. Neskorá interakcia znamená, že proces agregovania vektorov (angl. pooling) je ponechaný až na fázu výpočtu relevancie medzi sekvenciou vektorov reprezentujúcich dopyt E_q a sekvenciou reprezentujúcou dokument E_d.

Relevantnosť sa potom počíta tak, že pre každý vektor tokenu dopytu $E_{q_i} \in E_q$ sa nájde maximálna podobnosť (pomocou zvolenej metriky vzdialenosti) s niektorým vektorom tokenu dokumentu $E_{d_j} \in E_d$. Súčet týchto maximálnych podobností tvorí celkové skóre $S_{q,d}$:

$$S_{q,d} = \sum_{i \in |E_q|} \max_{j \in |E_d|} E_{q_i} \cdot E_{d_j}^T$$

Nevýhodou tohto prístupu je jeho pamäťová náročnosť, keďže je potrebné uchovávať N-dimenzionálny vektor pre každý token dokumentu.

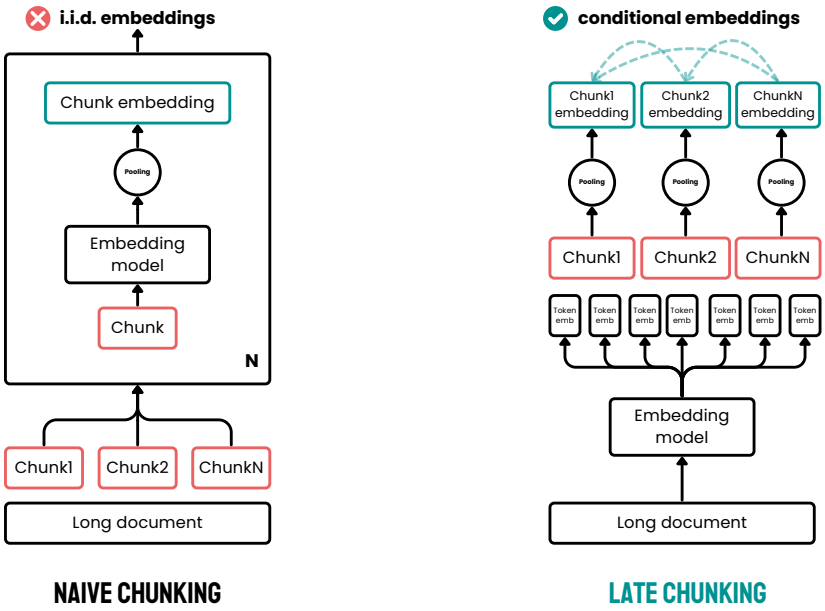
Neskorá segmentácia dokumentu

Ďalším riešením môže byť nesegmentovať text pred vytvorením vektorovej reprezentácie, ale segmentovať až výstupnú sekvenciu. Tento spôsob si vyžaduje model s veľkým kontextom a trénovanie s použitím stratégie span pooling, na čo poukazujú autori v ich práci [16]. Podľa výsledkov prezentovaných v práci by mal tento spôsob zachovávať kontext celého dokumentu v každom segmente.

Návrh reprezentácie príbehu

V nadväznosti na prehľad súčasných prístupov, ktoré riešia reprezentáciu dokumentov, bola identifikovaná potreba rozdeliť príbeh na krátke, no dostatočne informatívne a významné segmenty z hľadiska určovania podobnosti príbehov. Zároveň sa dnes metódy pre určovanie podobnosti príbehov posúvajú od sémantickej podobnosti celého príbehu a snažia sa o hlbšiu, viacúrovňovú analýzu, pričom demonštrujú potenciál LLM.

Z týchto dôvod bola navrhnutá štrukturovaná reprezentácia príbehov, založená na analýze, pri ktorej ide o extrakciu významných bodov príbehu s cieľom izolovať ich, a umožniť tak nájsť podobnosti na nižšej úrovni granularity. Konkrétne, je analýza zameraná na štyri



Obr. 4.2: Schéma neskorej segmentácie dokumentu. Dlhý dokument je najprv spracovaný embedding modelom a až následne sú vektorové reprezentácie tokenov segmentované a agregované do jednej vektorovej reprezentácie pre celý segment. Prevzané z [16].

aspekty príbehu, ktoré sú považované za fundamentálne pre pochopenie a porovnanie príbehov: dej, postavy, prostredie a motív. Tento prístup je priamo inšpirovaný metódami Facet-based Narrative Similarity [1] a AIStorySimilarity [10].

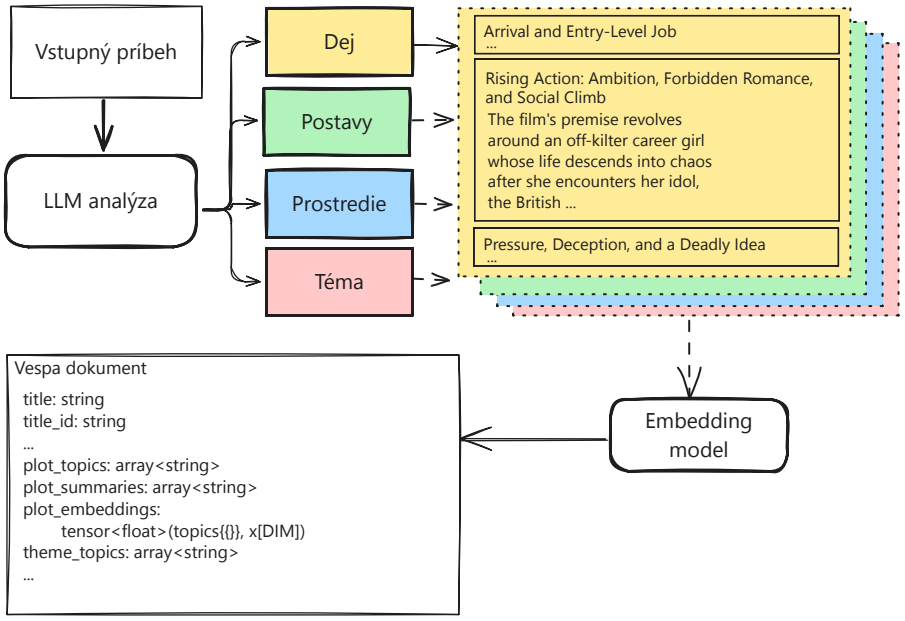
Na rozdiel od priameho vytvorenia vektorovej reprezentácie na základe pôvodného textu, navrhovaný systém najprv extrahuje a sumarizuje informácie relevantné pre každý zo štyroch aspektov. Relevantnými informáciami môžu byť kľúčové udalosti v rámci deja, analýzy konkrétnych postáv, popisy dôležitých miest či časových rámcov v rámci prostredia, alebo hlavné myšlienky a motívy. Každej takto identifikovanej téme je priradený jednak krátky, výstižný identifikátor alebo názov, a jednak podrobnejší textový sumár alebo popis, ktorý ju vysvetľuje alebo analyzuje.

Následne, zhrnutia extrahovaných podtém príbehov naprieč všetkými štyrmi aspektmi sú transformované do sémantickej vektorovej reprezentácie (angl. embedding) prostredníctvom embedding modelu. Týmto spôsobom nevzniká jediný vektor pre celý príbeh, ale sada vektorov, kde každý vektor zachytáva sémantický význam textového popisu konkrétnej sémanticky uceleného fragmentu príbehu. Tieto výsledky analýzy – ich textové popisy a k nim prislúchajúce sémantické vektorové reprezentácie – sú uložené, v štruktúrovanej podobe, vo vektorovej databáze.

Štrukturovaná podoba umožňuje flexibilné sémantické vyhľadávanie, kde používateľ môže hľadať príbehy podobné referenčnému na úrovni špecifických aspektov porovnávaním príslušných sád vektorov. Zároveň táto štruktúra podporuje cielené preusporiadanie výsledkov podľa zamerania sa na konkrétny aspekt príbehu alebo holistické porovnanie, pri ktorom systém môže kombinovať skóre podobnosti z viacerých aspektov. Rôzne stratégie hodnotenia relevancie tak môžu dynamicky priradiť rôznu váhu jednotlivým aspektom podľa špecifík konkrétneho dotazu a zámeru – hľadáme konkrétnu zápletku, prostredie, či určité morálne posolstvo?

Formálna definícia

Formálne by sa teda reprezentácia daného príbehu S mohla definovať ako usporiadaná štvorica množín $R_S = (I_{plot}, I_{character}, I_{setting}, I_{theme})$. Každá množina I_a by obsahovala naratívne položky p_k prináležiace danému aspektu a. Každá položka p_k by bola reprezentovaná ako trojica $(topic_k, text_k, \overrightarrow{v_k})$, kde $topic_k$ je jej krátky identifikátor, $text_k$ je jej textový sumár a $\overrightarrow{v_k}$ je sémantická vektorová reprezentácia pre $text_k$.



Obr. 4.3: Navrhovaná reprezentácia príbehov a schéma Vespa dokumentu pre túto dátovú štruktúru.

4.4 Efektívne vyhľadávanie a ukladanie

Uloženie a efektívne vyhľadávanie štruktúrovaných reprezentácií príbehov, ktoré pozostávajú zo sád textových popisov a ich zodpovedajúcich vektorových reprezentácií pre jednotlivé aspekty, si vyžaduje vhodný výber úložiska, ktoré by podporovalo rýchle vektorové vyhľadávanie. Tu vstupujú do hry vektorové úložiská, ktoré sú navrhnuté na správu, indexovanie a rýchle vyhľadávanie na základe výpočtov podobnosti vektorov reprezentujúcich konkrétny dokument. Dokument sa v tomto kontexte chápe ako dátový model alebo konkrétna inštancia uložená vo vektorovej databáze, ktorá je cieľom vyhľadávania a tvorí ju množina atribútov.

Vektorová databáza a platforma Vespa

Pre potreby efektívneho ukladania a vyhľadávania v navrhovanej štruktúrovanej reprezentácii príbehov sa javí ako vhodná voľba Vespa[4], ktorá podporuje viacvektorové vyhľadávanie a reprezentáciu dokumentov. Navyše, umožňuje definovať vlastné funkcie na výpočet

[4]Vespa: https://vespa.ai/

relevancie (angl. ranking expressions[5]), ktoré poskytujú základ pre implementáciu porovnávania príbehov podľa špecifických aspektov a vytváranie rôznych stratégií hodnotenia relevantných príbehov (angl. ranking profiles[6]).

Tenzorové polia pre štruktúrované dáta

Vo Vespe je tenzor[7] dátová štruktúra, ktorá zovšeobecňuje skaláry, vektory a matice na ľubovoľný počet dimenzií. Tenzory sa skladajú z buniek (angl. cells), pričom každá bunka obsahuje hodnotu a je identifikovaná unikátnou adresou pozostávajúcou z párov menovka-dimenzia. Dimenzie môžu byť *mapované* (riedke, s reťazcovými menovkami, napr. `topic{}`) alebo *indexované* (husté, s číselnými indexami od 0, napr. `embedding_dim[N]`).

Pre potreby tohto projektu je kľúčová schopnosť Vespy ukladať viacero vektorov pre jeden dokument pomocou tenzorového poľa s jednou mapovanou a jednou indexovanou dimenziou. Príklad takejto definície v schéme dokumentu:

```
field topic_vectors type tensor<float>(topic{},embedding_dim[1024]) {
    indexing: attribute | HNSW(distance-metric: angular)
    attribute:fast-search
}
```

V tomto príklade `topic{}` je mapovaná dimenzia, ktorá slúži ako identifikátor konkrétneho súhrnu v rámci deja, popisu postavy, špecifikácie prostredia alebo témy. Dimenzia `embedding_dim[1024]` je indexovaná a reprezentuje samotný vektor embeddingu. Toto umožňuje tzv. viacvektorové indexovanie (angl. multi-vector indexing), kde jeden dokument (príbeh) obsahuje sadu pomenovaných vektorov. Táto štruktúra priamo zodpovedá potrebe ukladať a vyhľadávať na základe sady vektorov pre každý príbeh, ako je definované v sekcii 4.3. Ďalej je v príklade definovaný druh indexu, ktorý sa má použiť pre pole `topic_vectors`. Definícia `HNSW(distance-metric: angular)` definuje dve dôležité veci: metriku pre určenie vzdialenosti medzi vektormi a typ indexu.

Približné vyhľadávanie najbližších susedov s HNSW

Na efektívne vyhľadávanie sémanticky podobných prvkov v rozsiahlych kolekciách vektorov sa využíva približné vyhľadávanie najbližších susedov (ANN [25]). Namiesto garantovaného nájdenia absolútne najbližších susedov (čo je výpočtovo extrémne náročné vo vysokodimenzionálnych priestoroch), ANN algoritmy poskytujú najbližších susedov s výrazne nižšou latenciou, no s určitou pravdepodobnosťou.

ANN je vo Vespa databáze implementovaný pomocou grafového indexu HNSW (bližšie popísané v sekcií 2.5). Implementácie HNSW vo Vespe ďalej umožňuje:

- **integráciu s filtrovaním:** ANN vyhľadávanie (cez operátor `nearestNeighbor` v dotazovacom jazyku YQL) je možné kombinovať s tradičnými filtrami na iných atribútoch dokumentu.
- **podpora pre CRUD operácie v reálnom čase:** vektory v HNSW indexe je možné pridávať, aktualizovať a mazať bez potreby kompletného prebudovania indexu.

[5] Výpočet relevancie dokumentov: https://docs.vespa.ai/en/ranking-expressions-features.html
[6] Profily relevancie: https://docs.vespa.ai/en/ranking.html
[7] Používateľský manuál k tenzorom: https://docs.vespa.ai/en/tensor-user-guide.html

- **viacvektorové HNSW indexovanie:** ako už bolo spomenuté, HNSW index je možné budovať nad tenzorovými poľami obsahujúcimi viacero vektorov na dokument. Dokument je potom nájdený, ak ktorýkoľvek z jeho indexovaných vektorov je sémanticky blízky vektoru dopytu.

Profily relevancie a tenzorové funkcie

Samotné nájdenie podobných vektorov je len časťou riešenia. Vespa poskytuje nástroj na definovanie logiky viacúrovňového výpočtu relevancie prostredníctvom výrazov pre vyhodnotenie relevancie (ranking expressions). Profil relevancie špecifikuje, ako sa má vypočítať skóre pre každý dokument vzhľadom na dopyt. Stavebnými blokmi sú tenzorové funkcie.

Tenzorové funkcie sú matematické operácie na tenzoroch, ktoré môžu pochádzať z dopytu (napr. `query(query_vector)`), z dokumentu (napr. `attribute(topic_vectors)`) alebo môžu byť definované ako konštanty. Pre navrhovaný systém to znamená, že priamo vo výraze pre relevanciu je možné:

- vypočítať skóre podobnosti (napr. kosínusovú podobnosť) medzi dopytovacím vektorom a každým individuálnym vektorom súhrnu v rámci príbehu. Príklad výpočtu kosínusovej podobnosti pre každý vektor v poli `topic_vectors` (skalárny súčin):

```
sum(
    query(query_vector) *
    attribute(topic_vectors)
    embedding_dim
),
```

- agregovať tieto individuálne skóre na úrovni aspektu alebo celého príbehu pomocou redukčných tenzorových funkcií, ako je napríklad `reduce(tensor_of_scores, aggregator, dimension)`. Napríklad, na nájdenie maximálnej podobnosti spomedzi všetkých súhrnov príbehu (`per_topic_similarity`) pozdĺž dimenzie `topics` je možné použiť:

```
reduce(per_topic_similarity, max, topics),
```

- kombinovať tieto agregované skóre z rôznych aspektov (dej, postavy, prostredie, témy) s rôznymi váhami, čím sa implementuje skutočne viac-aspektové porovnávanie. Napríklad: `0.6 * plot_score + 0.4 * character_score`.

- využívať preddefinované funkcie ako napr. `closeness(field, topic_vectors)`, ktorá vráti normalizované skóre podobnosti najlepšie zodpovedajúceho vektora v tenzorovom poli, alebo `closest(topic_vectors)`, ktorá identifikuje menovku (label) z mapovanej dimenzie (`topic`) najrelevantnejšieho vektora.

Profily relevancie tak umožňujú definovať viacero stratégií hodnotenia – od zamerania sa na konkrétny aspekt príbehu až po holistické porovnanie, kde sa skóre zo všetkých aspektov kombinujú. Ďalšou schopnosťou profilov je možnosť viacfázového vyhľadávania a zoradenie výsledkov, kde prvá fáza je zameraná na rýchle vyhľadávanie kandidátnych dokumentov (ANN) a nasledujúce fázy zoraďujú výsledky podľa výpočtovo náročnejších funkcií pre výpočet relevantnosti dokumentov alebo s využitím jazykových modelov.

Návrh profilov relevancie

Predošlé sekcie načrtli možnosti, ako je možné definovať relevanciu, čo možno označiť aj ako podobnosť medzi príbehmi, a to pomocou profilov relevancie a tenzorových funkcií pracujúcich nad vektorovými reprezentáciami analyzovaných príbehov. Návrh týchto profilov vychádza z niekoľkých princípov, ktorých cieľom je poskytnúť diverzitu výsledkov a pokryť rôzne typy dotazov, na ktoré sú používatelia zvyknutý.

Pre systém boli navrhnuté tri skupiny profilov relevancie, ktoré vychádzajú z detailnej, viac-aspektovej reprezentácie príbehov a potrieb používateľov.

Profily zamerané na aspekt. Aspektovo-špecifické profily priamo reflektujú viac-aspektovú reprezentáciu príbehu. Ich hlavným účelom je umožniť používateľovi alebo systému zamerať proces vyhľadávania a hodnotenia podobnosti na jeden konkrétny naratívny aspekt. Ak je napríklad kľúčová podobnosť v dejovej štruktúre, použije sa profil optimalizovaný pre dej; ak sú dôležité charakteristiky postáv, zvolí sa profil zameraný na postavy. Keďže sa v rámci jedného aspektu nejedná o jeden jediný vektor, ale o sadu vektorov reprezentujúcich rôzne prejavy daného aspektu (napr. rôzne dejové zvraty alebo analýzy viacerých postáv), treba zvoliť spôsob, akým túto viacvektorovú štruktúru agregovať a premietnuť do výsledného skóre relevancie pre daný aspekt. Keďže text reprezentujúci jeden bod príbehu má charakter celistvej informácie, bol pre tieto profily zvolený spôsob redukcie na základe maximálnej podobnosti. To znamená, že výsledná hodnota relevancie dokumentu sa rovná skóre podobnosti pre najbližší prvok poľa daného aspektu vzhľadom na dopyt.

Holistické profily. Na rozdiel od predchádzajúcej kategórie, holistické profily kombinujú podobnosť zo všetkých aspektov, aby poskytli jedno celkové skóre podobnosti. Tieto profily sú vhodné pre všeobecné odporúčania, pre menej špecifické dopyty alebo pre situácie, keď používateľ hľadá príbehy s čo najväčším počtom zhôd k referenčnému príbehu (dopytu). V tomto zameraní na celkovú podobnosť môže hrať dôležitú úlohu aj to, koľko podtém je podobných k dopytu, preto v tomto prípade dáva zmysel premietnuť skóre sémantickej podobnosti každej témy (vzhľadom na dopyt) do výsledného skóre relevancie dokumentu. Skóre podobnosti sa najprv počíta na úrovni každého aspektu samostatne, kde je výstupom vektor skóre podobnosti medzi témou a dopytom pre každú tému aspektu. Premietnutie všetkých tém do výsledného skóre relevancie preto vieme vykonať spriemerovaním tohto vektora. Druhou možnosťou je vybrať maximálnu hodnotu. Výsledné hodnoty pre každý aspekt sa spriemerujú do finálneho skóre relevancie dokumentu, pričom je možné použiť dynamický vážený priemer, podľa hodnôt definovaných dopytom.

Lexikálne profily. Motiváciou pre zavedenie lexikálnych profilov je častý scenár použitia vyhľadávačov, kedy používatelia formulujú dopyty pomocou kľúčových slov. Pre tento prípad použitia je primárne relevantné, či a ako často sa hľadané kľúčové slová vyskytujú v textových atribútoch dokumentu, bez ohľadu na ich sémantickú kategorizáciu do aspektov. Navrhnutý systém preto v rámci lexikálnych profilov určuje zhodu na základe všetkých relevantných textových atribútov príbehu, ako sú názov, autor, jednotlivé textové súhrny analyzovaných aspektov, či dokonca plný text príbehu, ak je dostupný.

Hybridné profily. Kombináciou lexikálnych a sémantických profilov je možné využiť silné stránky oboch a poskytnúť tak najuniverzálnejší profil.

4.5 Integrovanie LLM do systému

Navrhnutý systém implementuje kľúčové funkcie pomocou generatívneho LLM a to najmä pri analýze príbehov a pri RAG vyhľadávaní. Navrhnutá štrukturovaná reprezentácia príbehu a realizácia RAG systému si vyžaduje, aby výstupy boli generované v štandardizovanom a štruktúrovanom formáte.

Štrukturovaný výstup

Tento prístup znižuje nejednoznačnosť a umožňuje automatizované spracovanie a validáciu. V súčasnosti je štandardom podpora štrukturovaného výstupu a to buď formou inštrukcie v prompte alebo gramatikou obmedzeným dekódovaním.

Na základe prieskumu sa pre tento systém ako primárna metóda volí priame generovanie v požadovanom formáte. Tento prístup je zvolený s ohľadom na zistenia, že menej reštriktívne metódy, ako je priame inštruovanie modelu, majú tendenciu lepšie zachovávať pôvodné schopnosti modelu v porovnaní s úpravou priestoru tokenov pri dekódovaní obmedzenom gramatikou [53]. Hoci generovanie podľa komplexných schém prostredníctvom promptu môže byť pre menšie modely náročné a menej spoľahlivé v garantovaní validného výstupu v porovnaní s gramatikou obmedzeným dekódovaním [35].

Priame generovanie v požadovanom formáte

LLM je inštruovaný prostredníctvom promptu, to si vyžaduje, aby bol doladený (fine-tuned) na dátových sadách obsahujúcich príklady výstupu v cieľovom formáte (napr. `JSON`, `YAML`, `XML`). Model sa tak učí generovať text, ktorý zodpovedá syntaxi a štruktúre daného formátu.

Výhodou tohto prístupu je, že implementácia môže byť jednoduchšia pri použití modelov s adekvátnymi schopnosťami porozumenia a replikácie formátu.

Nevýhodou je však možnosť generovania syntakticky nesprávneho alebo neúplného výstupu, napríklad chýbajúce zátvorky v `JSON` alebo nesprávne odsadenie v `YAML`. Preto je nutná následná validácia a potenciálne opakovanie požiadavky pri chybách, čo zvyšuje latenciu a operačné náklady. Kvalita výstupu je navyše často ovplyvnená konkrétnym LLM a komplexnosťou požadovanej štruktúry.

Obmedzenie dekódovania gramatikou (angl. constrained decoding)

Proces generovania tokenov modelom je aktívne riadený tak, aby výsledný výstup vždy zodpovedal vopred definovanej formálnej gramatike (napr. gramatike pre `JSON` alebo špecifickej schéme). V každom kroku dekódovania sú povolené iba tie tokeny, ktoré vedú k syntakticky správnemu pokračovaniu podľa danej gramatiky.

Hlavnými výhodami sú zabezpečenie syntaktickej správnosti výstupu a eliminácia chýb formátovania, čo znižuje potrebu opakovania požiadaviek a zjednodušuje následné spracovanie. Táto metóda tiež umožňuje definovať komplexnejšie výstupné štruktúry, ktoré zvyšujú riziko chýb v prípade priameho generovania.

Na druhej strane, implementácia môže byť náročnejšia a vyžaduje nástroje alebo knižnice podporujúce obmedzené dekódovanie[8]. Obmedzené dekódovanie je implementované zásobníkovým automatom, čo zvyšuje výpočtovú náročnosť dekódovania a vedie k pomalšiemu generovaniu výstupu [40]. Striktné gramatické obmedzenia môžu v niektorých prípadoch ovplyvniť sémantický obsah alebo variabilitu výstupu [53].

[8] Knižnica Outlines: https://dottxt-ai.github.io/outlines.

Analýza príbehu

Transformácia vstupného korpusu príbehov do navrhnutej reprezentácie je úlohou veľkého jazykového modela. Samotnej analýze predchádza spracovanie vstupu do vhodného formátu. Vstupom tejto analýzy je text príbehu, ktorého hlavným zdrojom je opis deja z Wikipédie. S cieľom zvýšiť dôveryhodnosť systému a poskytnúť možnosť rýchleho overenia na základe odkazov na zdrojový text, musí byť súčasťou predspracovania textu segmentácia a značkovanie segmentov, aby bolo možné sa na ne odkazovať. Vstupný text príbehu je rozdelený na segmenty s prekryvom. Prekryv má zmierniť problém chýbajúceho kontextu. Pre samotnú analýzu nemá vplyv, keďže dĺžka opisov dejov z Wikipédie je v rozsahu veľkého množstva modelov. Pôvodný návrh však pracoval priamo so zdrojovým textom vrámci RAG modulu, kde bolo nutné riešiť izolované spracovanie segmentov. Hlavnou požiadavkou na prompt je poskytnúť inštrukcie s cieľom zabezpečiť požadovaný formát výstupu, ktorý je špecifikovaný v sekcií 4.3.

RAG modul

Navrhovaný RAG generuje odpovede iteratívne a využíva nástroje, pričom tento spôsob vychádza z princípov práce ReAct [60].

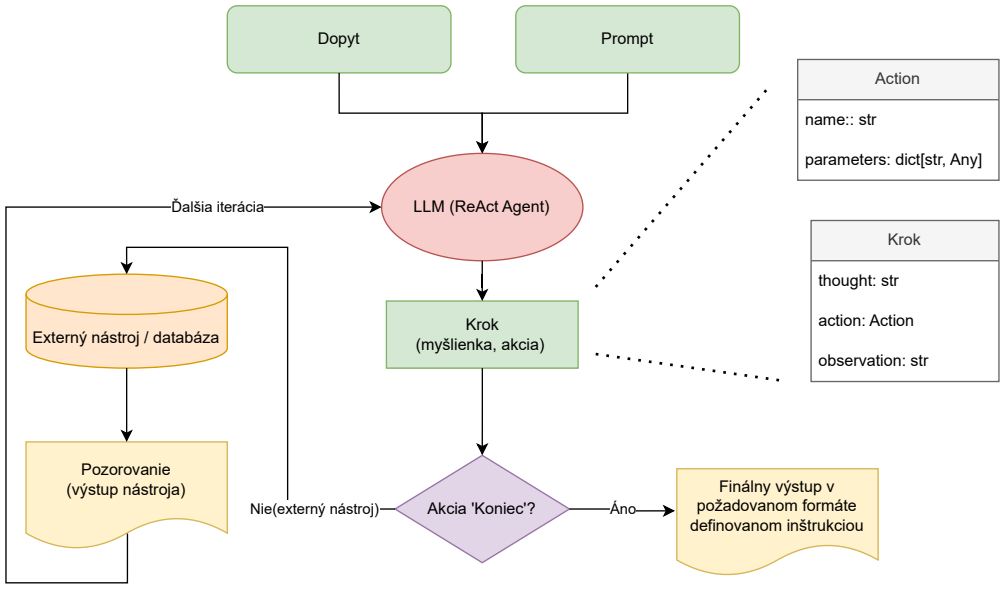
ReAct

ReAct predstavuje rámec, ktorý umožňuje LLM kombinovať plánovanie a vykonávanie akcií na riešenie komplexných úloh. Namiesto toho, aby model generoval iba finálnu odpoveď, alebo sa spoliehal výlučne na svoje interné znalosti, ReAct inštrukcie ho vedú k vytváraniu explicitných stôp uvažovania a k interakcii s externým prostredím prostredníctvom nástrojov.

Základný princíp fungovania ReAct spočíva v iteratívnom procese, ktorý sa skladá z nasledujúcich fáz:

- **uvažovanie:** LLM najprv vygeneruje myšlienkový pochod alebo stopu uvažovania. Táto myšlienka slúži na analýzu aktuálneho stavu problému, dekompozíciu úlohy na menšie kroky, formulovanie stratégie, aktualizáciu plánu na základe predchádzajúcich pozorovaní, alebo na identifikáciu potreby ďalších informácií. Myšlienky pomáhajú modelu sledovať postup a prispôsobovať sa dynamickým situáciám;
- **akcia:** na základe vygenerovanej myšlienky sa LLM rozhodne vykonať konkrétnu akciu. Akcia je zameraná na interakciu s externým prostredím, často prostredníctvom definovaných nástrojov, s cieľom získať nové informácie alebo ovplyvniť stav prostredia;
- **pozorovanie:** po vykonaní akcie systém získa pozorovanie, čo je výsledok alebo spätná väzba z externého prostredia (napr. výsledok vyhľadávania z databázy, odpoveď z API nástroja). Toto pozorovanie je následne poskytnuté LLM ako vstup pre ďalší cyklus uvažovania.

Tento cyklus uvažovania, akcie a pozorovania sa opakuje, pričom LLM dynamicky prispôsobuje svoj plán a postup, až kým nedosiahne riešenie úlohy alebo nevyčerpá definované maximum krokov.



Obr. 4.4: Architektúra ReAct agenta znázorňujúca iteratívny cyklus spracovania (myšlienka, akcia, pozorovanie) a súvisiace dátové štruktúry Krok a Akcia.

Nástroje. V kontexte ReAct sú nástroje externé funkcie alebo služby, ktoré LLM môže použiť prostredníctvom svojich akcií. Tieto nástroje rozširujú schopnosti LLM tým, že mu umožňujú prístup k informáciám z databázy, vykonávať špecifické výpočty, alebo interagovať s inými systémami, ktoré sú mimo jeho priameho dosahu. LLM plánuje, kedy a ako tieto nástroje použiť na základe úlohy a aktuálneho kontextu.

Prompt. Spôsob, akým LLM generuje stopy uvažovania a akcie v rámci ReAct, je podmienené navrhnutým promptom. Tento prompt zvyčajne obsahuje niekoľko príkladov demonštrujúcich požadovaný formát a postup riešenia úlohy, vrátane sekvencií myšlienok, akcií (s volaním nástrojov) a pozorovaní. Prompt tak usmerňuje model, ako má štruktúrovať svoje výstupy a interakcie.

Výsledkom je história krokov, ktorá reprezentuje proces riešenia úloh, kde je možné sledovať, ako model dospel k záveru, a identifikovať prípadné chyby v jeho uvažovaní alebo interakciách s nástrojmi.

Porovnávanie príbehov

V kontexte porovnávania príbehov funguje RAG modul založený na princípoch ReAct takto:

1. Najprv systém spracuje vstupný dopyt používateľa.
2. Následne prostredníctvom histórie krokov formuluje plán. Tento plán môže zahŕňať identifikáciu relevantných aspektov príbehu na porovnanie a špecifikáciu kritérií pre vyhľadávanie.
3. Potom systém vykoná akcie na získanie informácií z databázy analyzovaných príbehov. Napríklad vyhľadá príbehy s podobnými dejovými prvkami, charakteristikami postáv alebo témami.

4. Na základe získaných informácií aktualizuje svoje uvažovanie a iteratívne pokračuje v pláne, až kým nedospeje k finálnej odpovedi, ktorou je súhrnu podobností.

Návrh výstupného formátu odporúčaní agenta

Forma, akou agent podáva svoje výsledky, je veľmi dôležitá časť návrhu. Formát a jeho hlavné atribúty vychádzajú z niekoľkých praktických úvah o komplexnej povahe otázok, viacúrovňovej podobnosti príbehov a ako ich prezentovať v jednoducho pochopiteľnej podobe.

Prehľadné tematické zoskupovanie a prispôsobivosť. Keďže otázky používateľov sú často zložité a môžu pokrývať viacero tém, agent zoskupuje výsledky podľa tém alebo logických súvislostí. Používateľom to uľahčuje orientáciu v odpovediach, znižuje riziko zahltenia informáciami a umožňuje im sústrediť sa na najdôležitejšie časti. Takéto usporiadanie sa tiež dobre prispôsobuje rôznemu počtu nájdených príbehov, či už ich je málo alebo veľa.

Transparentnosť procesu a viacúrovňové vysvetlenia. Návrh kladie dôraz na jasné vysvetlenia na viacerých úrovniach: prečo je relevantná celá skupina príbehov vo vzťahu k pôvodnej otázke a prečo bol do nej zaradený každý konkrétny príbeh. Usporiadanie výsledkov zároveň čiastočne odkrýva, ako agent postupne pracoval, aké stratégie použil a ako dospel k záverom, čo zvyšuje celkovú transparentnosť jeho činnosti.

Flexibilita v zobrazení rôznych typov výsledkov. Nie všetky nájdené odporúčania majú rovnakú povahu alebo dôležitosť. Zoskupovanie umožňuje flexibilne prezentovať rôzne kategórie výsledkov – napríklad hlavné zhody, alternatívne návrhy, či doplnkové informácie – každú vo vlastnej, jasne označenej sekcii, čím sa zvyšuje zrozumiteľnosť pre používateľa.

Tieto úvahy viedli k návrhu výstupného formátu, ktorý tvoria tieto podstatné atribúty:

- história krokov agenta – prehľad jednotlivých fáz práce agenta, obsahujúci jeho úvahy, vykonané akcie (napríklad vyhľadávania) a získané pozorovania (výsledky akcií),
- celkové záverečné zhrnutie – súhrnné hodnotenie a konečný pohľad agenta na výsledky jeho pátrania a odporúčania,
- finálne odporúčania – hlavná časť výstupu, obsahujúca zoznam odporúčaných príbehov usporiadaných do tematických skupín,
- súhrn skupiny – krátky popis spoločných čŕt alebo hlavného zamerania príbehov v rámci jednotlivej tematickej skupiny,
- zdôvodnenie skupiny – vysvetlenie, prečo agent považuje jednotlivú tematickú skupinu príbehov ako celok za relevantnú vzhľadom na pôvodný dopyt používateľa,
- zoznam príbehov v skupine – konkrétne odporúčané príbehy v rámci jednotlivej tematickej skupiny, pričom každý príbeh je ďalej popísaný špecifickými údajmi,
- dôvod zaradenia do odporúčaní – konkrétne vysvetlenie od agenta, prečo tento špecifický príbeh odporúča v kontexte danej skupiny a používateľovho dopytu,
- skóre relevancie – číselná hodnota, ak bola poskytnutá vyhľadávacím nástrojom alebo iným hodnotiacim mechanizmom.

4.6 Návrh používateľského rozhrania

Pre vizualizáciu výsledkov a interakciu so systémom bola zvolená webová aplikácia. Cieľom návrhu bolo zaistiť splnenie dvoch základných funkcií, vyhľadávanie podobných príbehov a možnosť prezerať si výsledky analýzy príbehov. Za týmto účelom boli preskúmané vyhľadávacie rozhrania, ktoré poskytujú známe a používané databázy filmov, kníh a iného obsahu. Konkrétne bola preskúmaná funkcionalita a rozhranie stránok GoodReads[9], IMDb[10] a nástroja pre sémantické vyhľadávanie – ConnectedPapers [11].

Silnou stránkou sémantického vyhľadávania je nájdenie známeho, no čiastočne zabudnutého diela na základe jeho obsahu. Používatelia si môžu pamätať *„knihu o dievčati, ktoré cestovalo v čase“* alebo *„chlapcovi uviaznutom na lodi s tigrom“* to presne zodpovedá typu dopytov, v ktorých exceluje sémantické vyhľadávanie. Navrhnuté rozhranie by im preto malo poskytnúť možnosť, ako jednoducho a intuitívne zadať otázku, dokument alebo iný text v prirodzenom jazyku a následne poskytnúť prehľad čo najrelevantnejších výsledkov, s vizuálnym naznačením podobnosti medzi hľadaným a nájdenými príbehmi. Tento prípad použitia je hlavným zameraním používateľského rozhrania navrhovaného systému.

Poslednou funkcionalitou rozhrania je interakcia s agentným RAG systémom, ktorý na základe textovej inštrukcie alebo zdrojového príbehu iteratívne vyhľadáva a sumarizuje výsledky. Keďže ide o vyhľadávanie, je navrhnutá rovnako ako štandardné vyhľadávanie. Líši sa len formou toho ako sú prezentované výsledky, kde dôležitú úlohu hraje slovný opis podobností a prezentácia histórie krokov agenta.

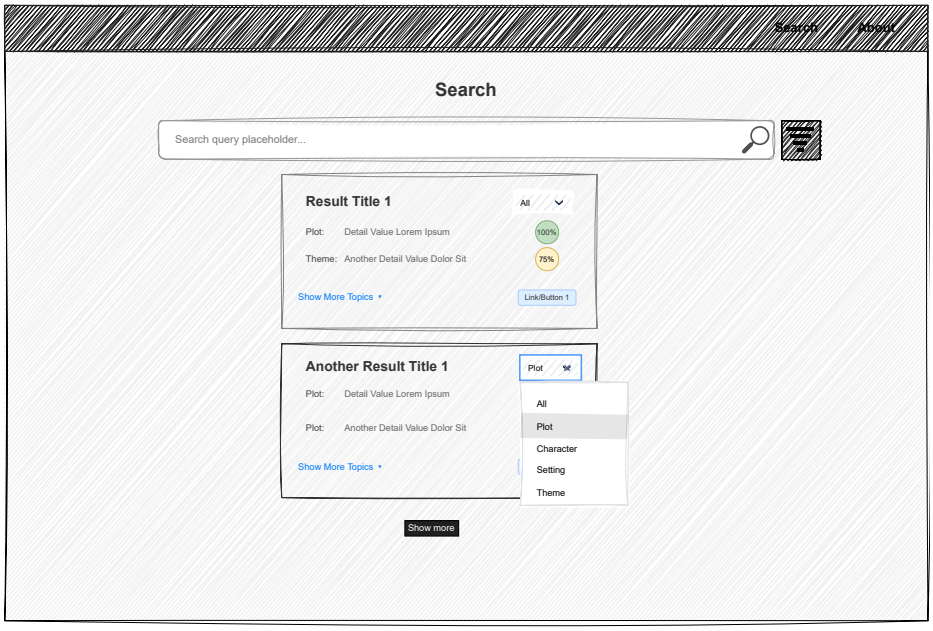
Vyhľadávanie a vizualizácia výsledkov

Interakcia používateľa so systémom pri vyhľadávaní sa riadi zaužívanými princípmi a prvkami rozhrania: zadanie dopytu do vstupného poľa a následné zobrazenie relevantných výsledkov. Tento dizajn je jadrom každého rozhrania pre vyhľadávanie. Ďalším dôležitým rozhodnutím je vybrať si, čo je používateľovi podstatné ukázať v zozname výsledkov. Karta príbehu v zozname výsledkov bola navrhnutá tak, aby bolo možné rýchlo nájsť a pochopiť, v čom spočíva podobnosť nájdeného príbehu so zadaným dopytom. Tento návrh bol zásadný aj pre výber databázy Vespa, kde flexibilita v definovaní výpočtov relevancie a exportovaní výsledkov umožňuje prezentačnej vrstve tieto dáta efektívne využiť. Výsledok sémantického vyhľadávania pozostáva nielen z celkového skóre relevancie pre každý príbeh, ale aj zo skóre pre každú extrahovanú tému či aspekt vzhľadom na pôvodný dopyt. Konkrétne, rozhranie vizualizuje normalizované skóre pre štyri hlavné naratívne aspekty a pre špecifické analyzované témy pomocou percentuálneho skóre, ktoré musí byť normalizované, aby uľahčilo interpretáciu, keďže produkovaná podobnosť môže mať rôzny rozsah hodnôt.

Detailný pohľad na príbeh

Táto sekcia rozhrania je určená na prezentáciu rozšírených informácií o príbehu, ktorý používateľ zvolil zo zoznamu výsledkov vyhľadávania. Ponúka súhrn výsledkov analýzy daného príbehu a doplňujúce detaily, ktoré nie sú obsiahnuté v prehľade výsledkov. Súčasťou tohto pohľadu je aj funkcionalita pre objavovanie podobných príbehov prostredníctvom dedikovaného zoznamu „*Podobné príbehy*“, čo reflektuje bežné konvencie vyhľadávacích aplikácií.

[9] Odporúčanie kníh – Goodreads: https://www.goodreads.com/
[10] Databáza filmov IMDb: https://www.imdb.com/
[11] Connected Papers: https://www.imdb.com/



Obr. 4.5: Náčrt používateľského rozhrania pre vyhľadávanie a zobrazenie výsledkov. Zobrazuje vstupné pole pre dopyt a príklad výsledku s vizualizáciou skóre relevancie pre jednotlivé aspekty príbehu.

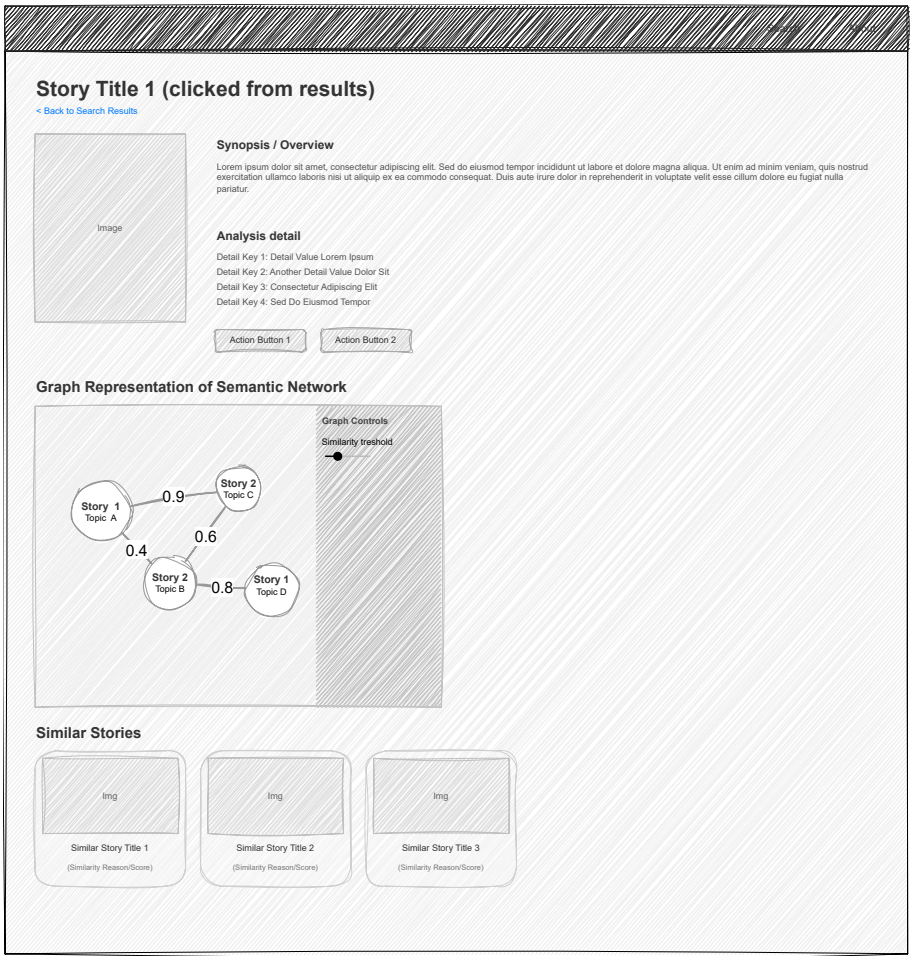
Vizualizácia sémantických vzťahov formou grafu

Vzhľadom na to, že navrhovaný dátový model umožňuje komparáciu príbehov na úrovni granulárnych naratívnych elementov (ako sú témy, udalosti, motívy) extrahovaných počas analytickej fázy, systém dokáže vizualizovať komplexnú sieť vzťahov medzi príbehmi. Pre tento účel bola zvolená grafová reprezentácia, kde uzly symbolizujú extrahované naratívne elementy a hrany kvantifikujú mieru ich sémantickej podobnosti. Používateľ má možnosť interaktívne manipulovať s vizualizáciou, napríklad úpravou prahovej hodnoty pre zobrazenie podobnosti, aplikovaním filtrov podľa názvu diela, alebo selektívnym zameraním na konkrétny naratívny aspekt.

Prezentácia výsledkov RAG vyhľadávania

Používateľské rozhranie pre prezentáciu výsledkov RAG vyhľadávania príbehov by malo poskytnúť prehľad o procese a zisteniach agenta. Cieľom je nielen zobraziť odporúčané príbehy, ale aj vysvetliť, ako k nim agent dospel a aká bola jeho stratégia.

Rozdielom oproti štandardnému vyhľadávaniu je dĺžka tohto procesu. Keďže sa jedná o dlhší proces, rozhranie by malo na to používateľa upozorniť. Prezentácia výsledkov je závislá na navrhnutom výstupe ReaAct agenta. Ďalším rozdielom oproti štandardnému vyhľadávaniu je nedeterministickosť odpovedí. Každý výstup vyhľadávania je jedinečný, preto je vhodné umožniť používateľovi prezerať si históriu takýchto vyhľadávaní.



Obr. 4.6: Náčrt používateľského rozhrania na prezentáciu analýzy a ďalších informácii o príbehu. Zobrazuje opis príbehu, identifikované témy príbehu, zoznam podobných príbehov a interaktívny graf, reprezentujúci sémantické podobnosti medzi analyzovanými príbehmi.

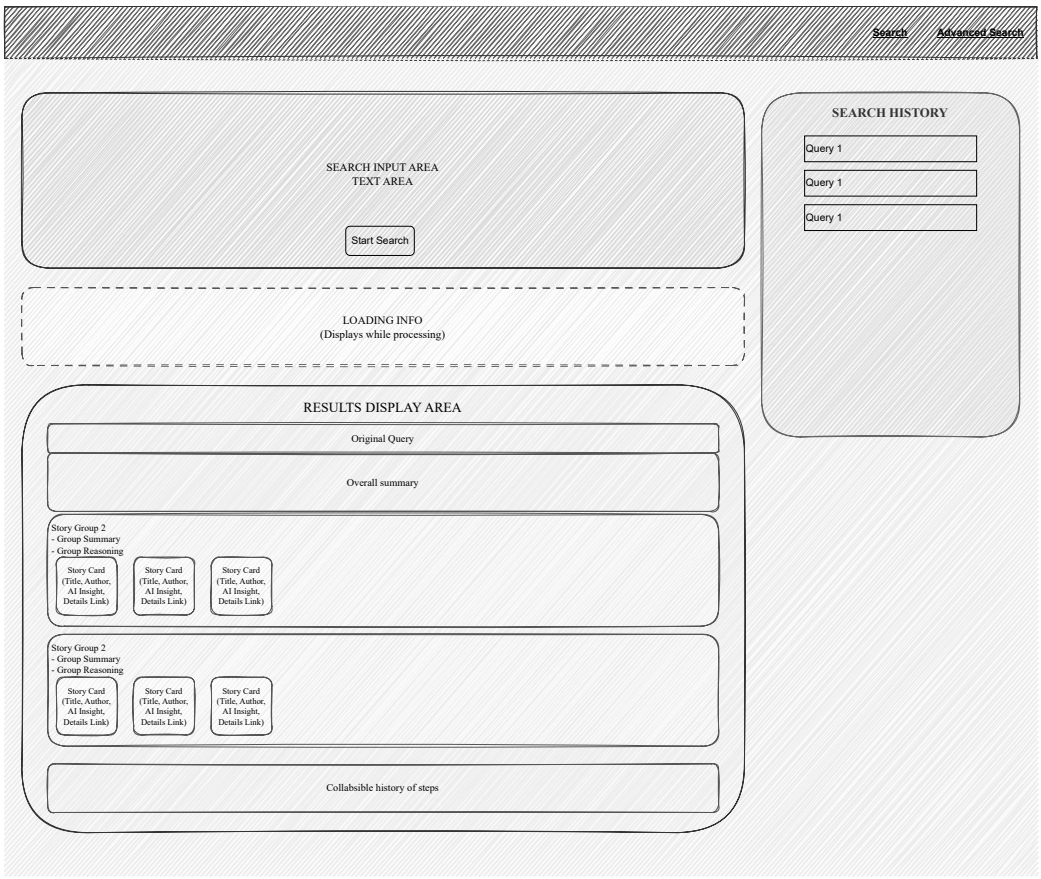
Zhrnutie dopytu a celkový prehľad

Na úvod sa používateľovi pripomenie jeho pôvodný dopyt. Ak agent vygeneroval celkové zhrnutie alebo stratégiu, táto informácia sa zobrazí ako prvá, poskytujúc kontext k nasledujúcim odporúčaniam. Táto sekcia je vizuálne odlíšená a zdôrazňuje hlavné myšlienky.

Agent môže zoskupiť nájdené príbehy do tematických kategórií. Každá skupina má vlastný súhrn a zdôvodnenie, prečo boli tieto príbehy vybrané alebo ako spolu súvisia.

V rámci každej skupiny sú príbehy prezentované formou kariet. Dôležitou súčasťou karty je „Postreh AI“, čo slúži pre krátke vysvetlenie, prečo bol konkrétny príbeh vybraný vo vzťahu k dopytu.

Poslednou navrhnutou sekciou je sekcia histórie krokov, prezentovaná ako rozbaliteľný prvok. Táto sekcia poskytuje možnosť nahliadnuť do iteratívneho procesu, ktorým agent dospel k výsledkom a zobrazuje jednotlivé kroky agenta.



Obr. 4.7: Wireframe návrh používateľského rozhrania RAG vyhľadávania.

Kapitola 5

Implementácia

V tejto kapitole je podrobne opísaná implementácia jednotlivých komponentov systému, použité technológie a dátové štruktúry. Projekt je štruktúrovaný do modulov zodpovedajúcich za logiku spracovania a analýzu dát (zahŕňajúcu taktiež adaptéry a ďalšiu potrebnú komunikáciu s poskytovateľmi LLM), prezentačnú vrstvu, dátové modely a experimenty vykonané pri vyhodnocovaní systému.

5.1 Použité technológie

Systém je implementovaný v programovacom jazyku Python vo verzii 3.12. Takmer všetky časti systému, webová aplikácia, práca s dátami a experimenty sú napísané v jazyku Python a k príprave prostredia stačí ľubovoľný systém na správu balíkov pre Python (v práci bol použitý nástroj `uv`[1]).

Rámec Adalflow

AdalFlow[2] je rámec navrhnutý pre implementáciu a optimalizáciu LLM systémov zložených z viacerých komponentov. Poskytuje základné triedy a rozhrania pre vytváranie optimalizovateľných LLM systémov. Filozofia dizajnu rámca je významne inšpirovaná princípmi knižnice PyTorch, napr. trieda `GradComponent` je analógiou `nn.Module`, takže všetky triedy, ktoré dedia túto triedu je možné optimalizovať a sú zaradené do výpočetného grafu. Automatická textová optimalizácia je bližšie vysvetlená v sekcii 2.7.

Jednotné rozhranie pre integráciu rôznych LLM

V práci používam open-source proxy-server LiteLLM[3], ktorý poskytuje jednotné programové rozhranie pre rôznych poskytovateľov veľkých jazykových modelov. Štandardizuje interakcie prostredníctvom formátu OpenAI API[4].

[1] Nástroj uv: https://github.com/astral-sh/uv
[2] Rámec AdalFlow: https://adalflow.sylph.ai/
[3] LiteLLM:https://www.litellm.ai/.
[4] OpenAI API: https://platform.openai.com/docs/overview.

Nástroje pre lokálne nasadenie LLM

Pre lokálne testovanie menších jazykových modelov bol využitý nástroj Ollama[5], ktorého API je zahrnuté v LiteLLM zozname podporovaných poskytovateľov. Lokálne poskytuje prístup k modelom cez REST/WebSocket API. Po spustení a inicializácii modelu (`ollama serve`) vytvorí HTTP server s endpointmi pre generovanie textu alebo vektorových reprezentácií (napr. `/api/generate`). Modely sa v Ollama sťahujú príkazom `ollama pull [názov]` z centralizovaného registra, kde sú uložené ako samostatné balíky obsahujúce váhy modelu, konfiguráciu a závislosti.

Vo fáze spracovania dát sa ukázali určité nedostatky pri vytváraní vektorových reprezentácií a to hlavne to, ako Ollama vyvažuje záťaž pri viacerých asynchronných požiadavkách, kedy nevyužíva celú dostupnú pamäť grafickej karty, čo malo negatívny vplyv na rýchlosť vytvárania vektorovej reprezentácie pre analyzované príbehy. Z tohto dôvodu bolo nutné implementovať metódy, ktoré využívajú knižnicu `sentence-transformers`.

Sentence-transformer[6] je python knižnica, ktorá poskytuje triedy s rozhraním prispôsobeným pre úlohy týkajúce sa sémantickej reprezentácie textu, vyhľadávania a určovania podobnosti dokumentov.

FastHTML

Webový rámec FastHTML[7], napísaný v jazyku Python, umožňuje implementáciu celej webovej aplikácie. FastHTML presadzuje hypermediálny prístup, kde server priamo generuje a zasiela HTML kód. Dôsledkom tohto architektonického rozhodnutia je eliminácia potreby vyvíjať samostatnú klientsku (frontend) aplikáciu. Týmto sa výrazne zjednodušuje celková štruktúra systému a interakcia medzi klientom a serverom, keďže komplexnosť spojená s oddeleným frontendom a backendom, vrátane náročnej synchronizácie dát a stavov, je takmer úplne odstránená. Server je implementovaný ako ASGI (Asynchronous Server Gateway Interface) prostredníctvom Uvicorn servera a Starlette webového rámca. ASGI poskytuje základ pre asynchrónnu komunikáciu na webe, čo FastHTML umožňuje efektívne spracovávať HTTP požiadavky a odpovede. Kľúčovým prvkom je tiež knižnica HTMX, ktorá rozširuje možnosti štandardného HTML a umožňuje implementovať interaktivitu bez nutnosti písania rozsiahleho a často komplexného JavaScript kódu. Vďaka HTMX môže akýkoľvek element na stránke iniciovať komunikáciu so serverom pomocou štandardných HTTP metód a udalostí. Server následne odpovedá HTML fragmentom, ktorý môže modifikovať ľubovoľnú časť existujúcej stránky bez potreby kompletného znovunačítania.

Ďalším dôvodom pre výber tohto rámca pre vývoj webovej aplikácie je to, že preberá syntax a spôsob definovania koncových bodov, logiku smerovania, autentifikáciu a ďalšie štandardné komponenty od známeho rámca FastAPI pre vývoj backend aplikácií v jazyku Python.

Vespa CLI

Vespa CLI[8] (verzia 8.423.30) je program určený na interakciu s databázou Vespa z príkazového riadka. V rámci tejto práce sa primárne využíval na vkladanie dát, a to konkrétne

[5] Ollama: https://ollama.com/.
[6] Knižnica Sentence-Transformers: https://sbert.net
[7] FastHTML: https://github.com/AnswerDotAI/fasthtml
[8] Vespa CLI: https://docs.vespa.ai/en/vespa-cli.html.

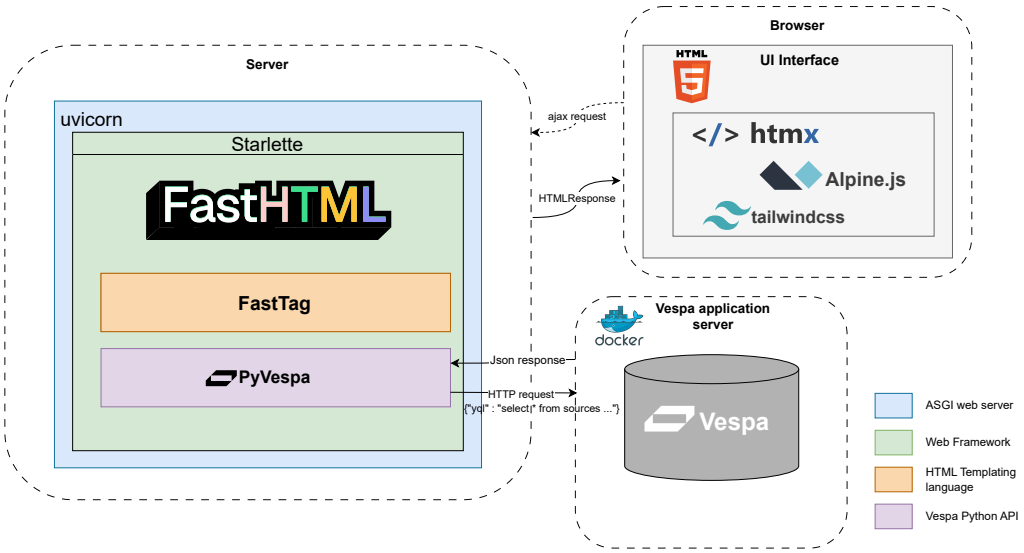
pomocou príkazu `vespa feed`. Tento spôsob bol zvolený pre jeho vysokú efektivitu a rýchlosť pri nahrávaní dát do databázy[9].

Docker

Pri lokálnom vývoji systému bol využívaný Docker pre vytvorenie prostredia pre Vespa databázu. Docker[10] izoluje aplikácie a ich závislosti do ľahko prenosných kontajnerov. Každý kontajner predstavuje autonómne prostredie obsahujúce všetky nevyhnutné prvky pre beh aplikácie, čo garantuje jej identické fungovanie nezávisle od hostiteľského systému.

5.2 Webová aplikácia

Architektúra aplikácie (obrázok 5.1) vychádza z princípov tvorby webových aplikácií na základe odpovedí len s HTML obsahom. Aplikácia je postavená na asynchrónnom webovom rámci FastHTML, ktorý stavia na princípoch server-side renderingu (sekcia 5.1).



Obr. 5.1: Architektúra webovej aplikácie.

Modul `main.py` predstavuje hlavný vstupný bod webovej aplikácie. Je zodpovedný za definovanie všetkých dostupných webových ciest, spracovanie HTTP požiadaviek od používateľa a generovanie dynamického HTML obsahu. To znamená, že väčšina logiky používateľského rozhrania a generovanie HTML kódu prebieha na strane servera. Tento prístup je doplnený knižnicou HTMX, ktorá umožňuje vykonávať AJAX požiadavky priamo z HTML elementov a dynamicky vymieňať len časti webovej stránky. Pre niektoré špecifické, čisto klientske interakcie, ako je ovládanie viditeľnosti bočného panela, sa využíva minimalistická JavaScript knižnica Alpine.js[11]. Pri tomto prístupe sa malé fragmenty JavaScript kódu vkladajú do generovaných HTML elementov.

[9] Experimentálne vyhodnotenie rýchlosti nahrávania dát: https://vespa-engine.github.io/pyvespa/examples/feed_performance.html.

[10] Platforma Docker: https://www.docker.com/

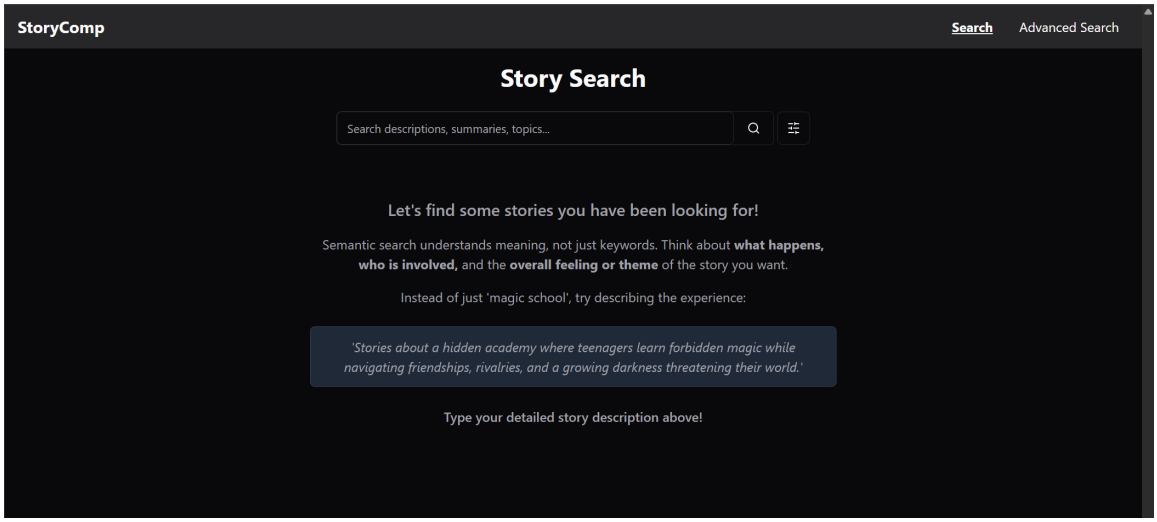
[11] https://alpinejs.dev/.

Dátová perzistencia pre uložené výsledky vyhľadávaní agenta je zabezpečená pomocou SQLite databázy, ku ktorej sa pristupuje prostredníctvom knižnice `fastlite`[12]. Na optimalizáciu výkonu a zníženie latencie pri opakovaných alebo časovo náročných operáciách systém implementuje niekoľko úrovní cachovania: `TTLCache` pre dočasné ukladanie stavov (napr. prebiehajúce výpočty) a súborovú cache pre predspracované dáta pre grafovú vizualizáciu vyhľadávania. Dátové štruktúry prenášané medzi komponentmi a pri komunikácii s frontendom sú validované a typované pomocou Pydantic modelov[13].

Navrhnuté funkcie a interakcie sú implementované prostredníctvom nasledujúcich skupín ciest, ktoré priamo reflektujú hlavné používateľské pohľady popísané v sekcii 4.6.

Spracovanie štandardného vyhľadávania a zobrazenie výsledkov

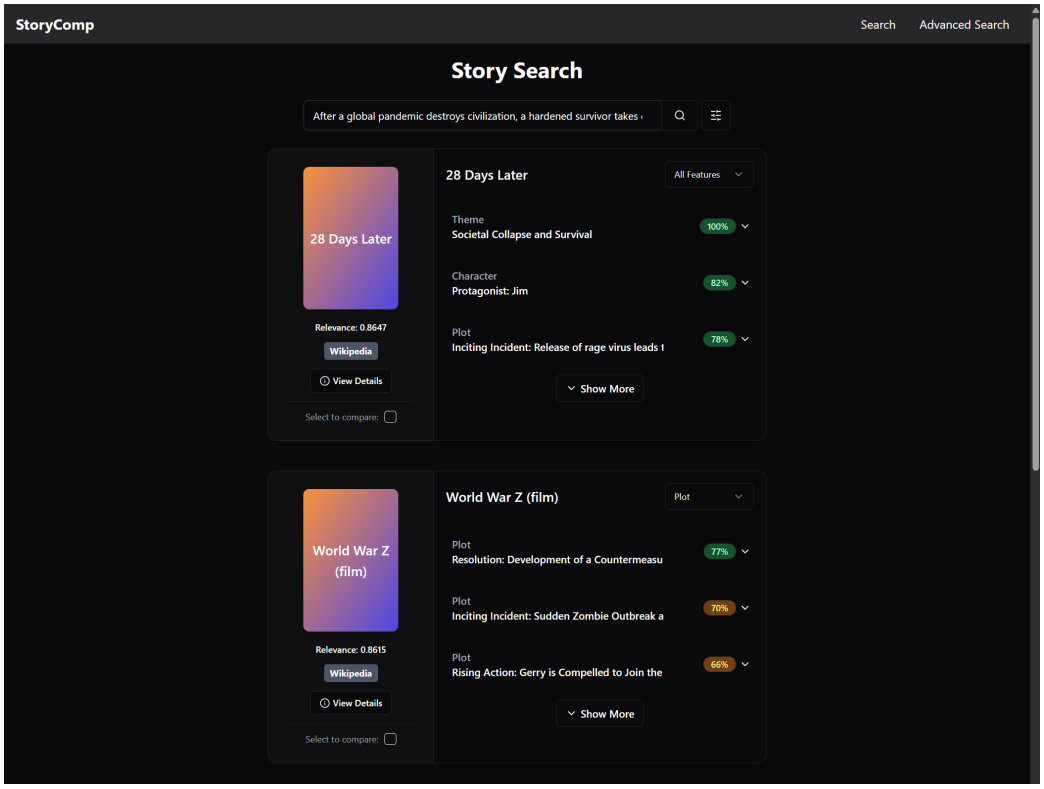
Tento súbor ciest zabezpečuje základnú funkcionalitu vyhľadávania príbehov. Vstupným bodom je cesta `GET /`, ktorá inicializuje používateľskú reláciu (session) a zobrazí hlavnú stránku vyhľadávania (`SearchPage`). Samotné vyhľadávanie, iniciované z tejto stránky, smeruje na cestu `GET /search`. Táto cesta parsuje parametre dopytu (textový reťazec, zvolený ranking profil, a filtre na autora, názov či rok vydania) z objektu požiadavky `Req` a volá funkciu `handle_search`. Funkcia `handle_search` následne komunikuje s Vespa klientom, získava výsledky a pripravuje dáta pre zobrazenie. Výsledky sú prezentované ako zoznam komponentov `StoryCard`.



Obr. 5.2: Používateľské rozhranie pre štandardné vyhľadávanie príbehov. Zobrazuje vstupné pole pre dopyt a nápovedu pre formuláciu dopytov, ktorá zdôrazňuje potrebu opisovať dej, postavy a celkovú tému namiesto jednoduchých kľúčových slov.

Taktiež je implementované stránkovanie výsledkov, čo zabezpečuje dedikovaná cesta `GET /search/more` prijíma aktuálne parametre vyhľadávania a na základe stavu stránkovania (uloženého v `sess`) načíta ďalšiu sadu výsledkov. HTMX atribút `hx-swap="beforeend"` (definovaný v komponente `ShowMoreButton`) zabezpečí, že nové výsledky sú plynulo pridané na koniec existujúceho zoznamu.

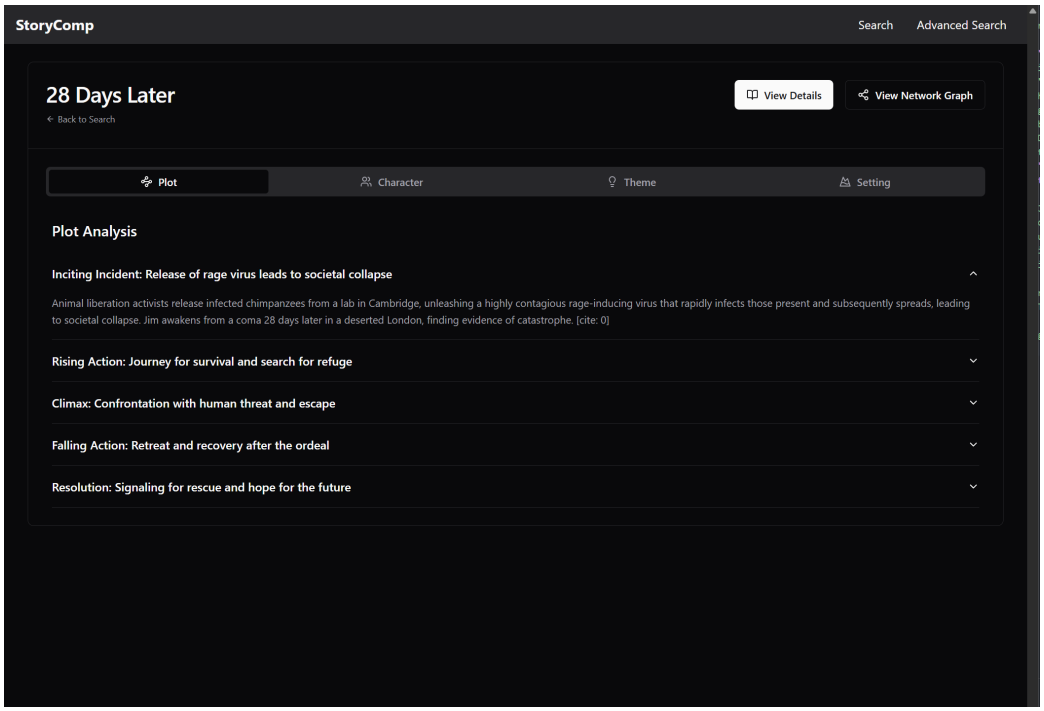
[12]Knižnica fastlite: https://github.com/AnswerDotAI/fastlite.
[13]Knižnica Pydantic: https://docs.pydantic.dev/.



Obr. 5.3: Zobrazenie výsledkov štandardného vyhľadávania. Každý nájdený príbeh je prezentovaný formou karty, ktorá obsahuje názov, celkové skóre relevancie a vizualizáciu podobnosti pre jednotlivé extrahované témy a aspekty. Používateľ si môže zobraziť viac detailov alebo porovnať vybrané príbehy.

Detailné zobrazenie príbehu a sieťová vizualizácia

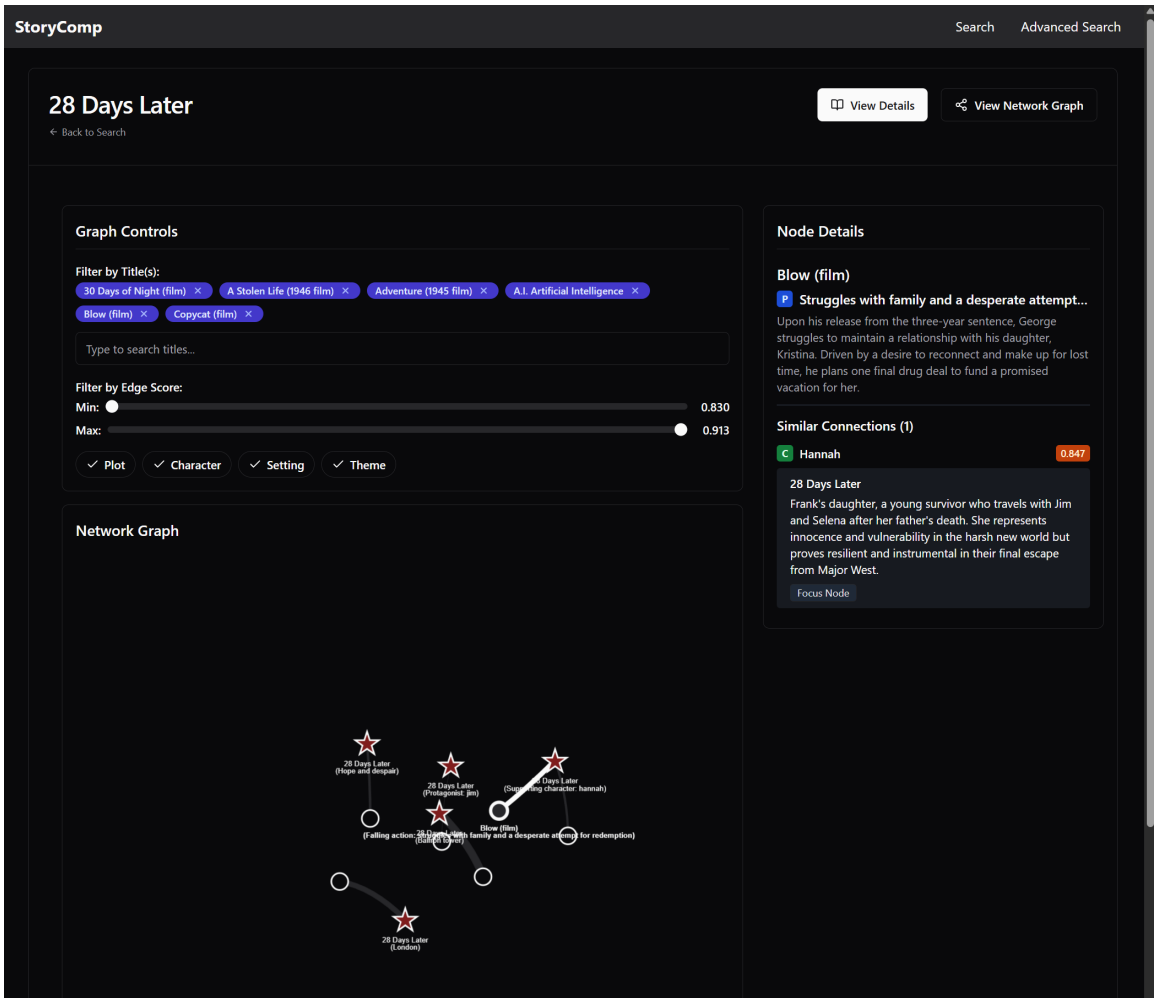
Pre hlbšiu analýzu jednotlivých príbehov slúži detailný pohľad, dostupný cez cestu `GET /story/{story_id}`. Táto cesta načíta všetky potrebné dáta o príbehu pomocou metódy `VespaTitleClient.get_document_details` a zobrazí ich v komponente `StoryPageView`, ktorý obsahuje štruktúrovanú analýzu rozdelenú do tematických kariet (`DetailsTabView`).



Obr. 5.4: Detailný pohľad na analyzovaný príbeh, konkrétne záložka „Plot“ (Dej). Táto časť zobrazuje štruktúrovanú analýzu deja rozdelenú na kľúčové udalosti ako úvodný incident (Inciting Incident), stúpajúcu akciu (Rising Action), klimax (Climax), klesajúcu akciu (Falling Action) a rozuzlenie (Resolution). Každá udalosť je popísaná a môže obsahovať odkaz na relevantnú pasáž v zdrojovom texte (napr. [cite: 0]).

Významnou súčasťou detailného pohľadu je interaktívny graf sémantických vzťahov. Ten je dostupný cez `GET /story/{story_id}/network`. Generovanie dát pre tento graf, ktoré zahŕňa výpočet podobností medzi mnohými prvkami príbehov (`find_similar_stories` a `_process_similarity_results_for_graph`), je výpočtovo náročné. Preto sa pri prvej požiadavke na túto cestu spúšťa asynchrónna úloha na pozadí (`_run_search_and_save`). Výsledky tejto úlohy sa ukladajú do cache súboru. Klientska strana používateľského rozhrania (komponent `NetworkView`) medzitým periodicky (pomocou atribútu `hx-trigger="every Xs"`) dopytuje stav cez cestu `GET /partials/story/{story_id}/network`. Akonáhle sú dáta pripravené, táto cesta vráti HTML fragment s dátami pre graf, ktorý sa následne vykreslí pomocou knižnice Vis.js[14], integrovanej cez `static/js/network.js`. Tento prístup zabezpečuje, že používateľ dostane hneď odpoveď (pohľad prejde do stavu načítavania) aj po iniciovaní dlhšieho procesu a po jeho dokončení sú jeho výsledky automaticky prezentované používateľovi.

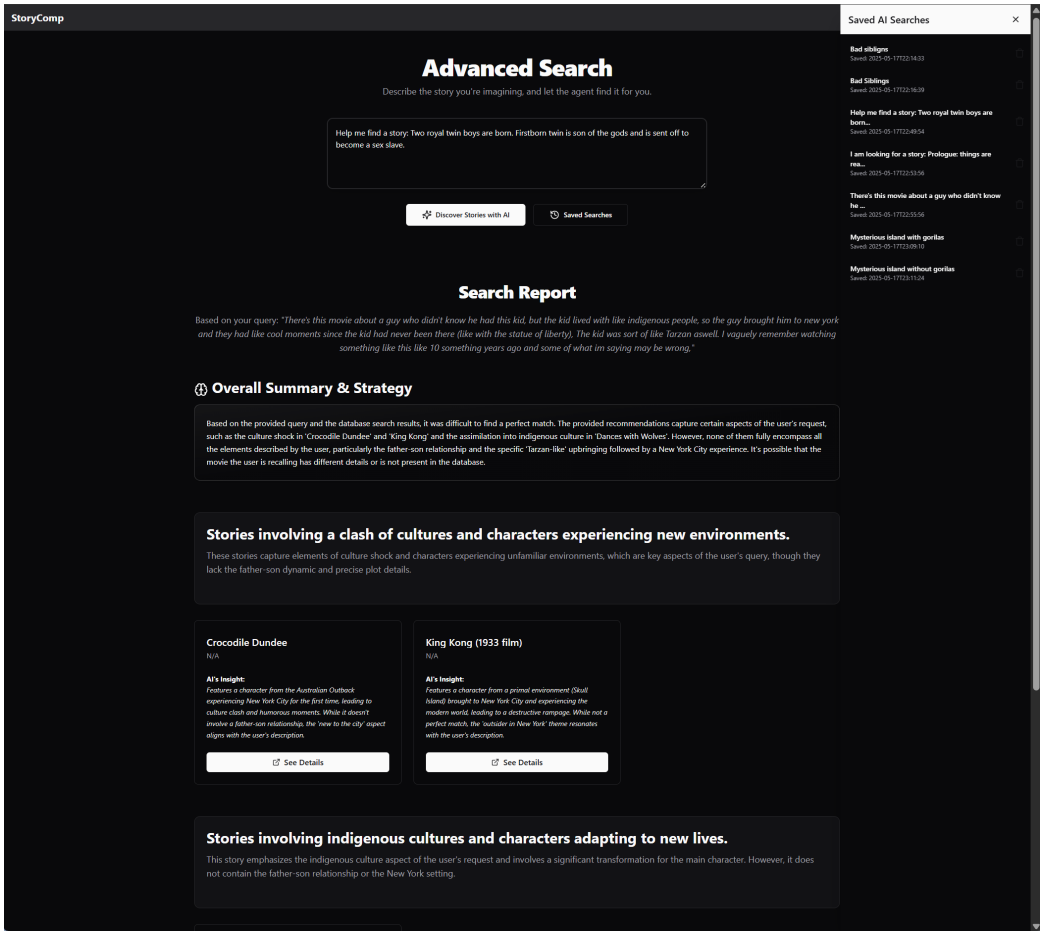
[14]Knižnica Vis.js pre vizualizáciu: https://visjs.github.io/vis-network/docs/network/.



Obr. 5.5: Interaktívna sieťová vizualizácia sémantických vzťahov medzi analyzovanými prvkami príbehov. Uzly v grafe reprezentujú príbehy alebo ich špecifické naratívne elementy a hrany znázorňujú mieru ich sémantickej podobnosti. Vizuálne odlíšené sú uzly prislúchajúce zdrojovému príbehu. Ovládacie prvky na ľavej strane umožňujú používateľovi filtrovať graf podľa názvu diela, minimálneho a maximálneho skóre podobnosti hrán a selektívne zobrazovať prepojenia na základe naratívnych aspektov (dej, postava, prostredie, téma). Panel vpravo („Node Details“) zobrazuje detaily vybraného uzla a jeho priame sémantické spojenia s inými prvkami, vrátane skóre podobnosti.

RAG vyhľadávanie

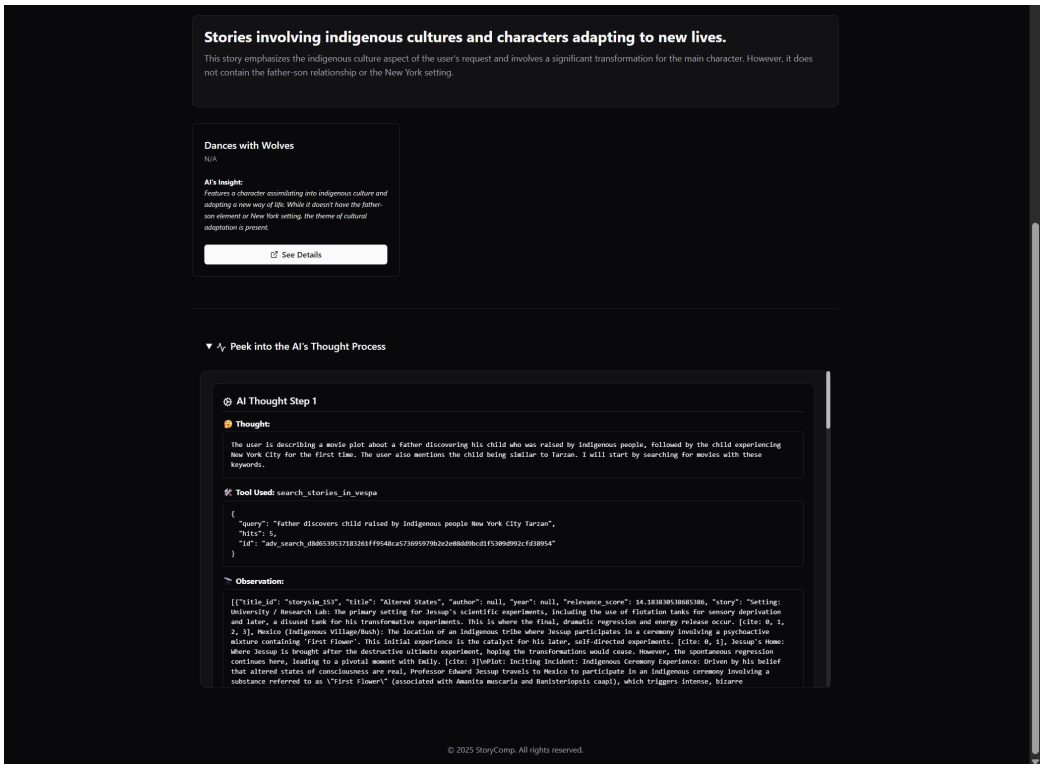
Pre RAG vyhľadávanie je k dispozícii rozhranie využívajúce agenta `StoryDiscoveryAgent`. Hlavná stránka tohto modulu, dostupná na `GET /advanced-search`, prezentuje formulár `AdvancedSearchForm` pre zadanie naratívneho dopytu a bočný panel `SavedQueriesSheet`, ktorý zobrazuje históriu uložených vyhľadávaní a ktorého interaktivita je riadená pomocou Alpine.js.



Obr. 5.6: Zobrazenie výsledkov iteratívneho RAG vyhľadávania iniciovaného používateľským dopytom. Stránka prezentuje správu („Search Report“) obsahujúcu celkové zhrnutie a stratégiu agenta („Overall Summary & Strategy“) a vysvetľuje logiku odporúčaní. Následne sú zobrazené tematicky zoskupené odporúčané príbehy. Každý príbeh v skupine je prezentovaný kartou s názvom, krátkym popisom vysvetľujúcim jeho relevanciu a odkazom na stránku príbehu. Používateľ ma taktiež možnosť prezerať si predošlé výsledky vyhľadávania na bočnom panely, ktorý sa otvorí po kliknutí na tlačidlo „Saved Searches“.

Po odoslaní dopytu je požiadavka smerovaná na `GET /partials/advanced-search/-results`. Táto cesta iniciuje beh agenta na pozadí. Výsledky agenta, po transformácii, sú ukladané do SQLite databázy. Používateľovi sa okamžite vráti komponent `AdvancedSearch-Poller`, ktorý periodicky (cez `GET /advanced-search/poll`) zisťuje stav spracovania dopytu v databáze. Akonáhle sú výsledky dostupné, komponent `AdvancedSearchResultsArea` ich zobrazí, vrátane súhrnov tematických skupín a možnosti preskúmať detailný postup

agenta (myšlienka, akcia, pozorovanie). Uložené vyhľadávania je možné opätovne načítať (`GET /advanced-search/results/{query_hash}`) alebo vymazať (`GET /advanced-search/results/{query_hash}/delete`), pričom zoznam v bočnom paneli sa efektívne aktualizuje prostredníctvom HTMX volania na `GET /partials/advanced-search/saved-queries-list`. Stav spracovania aktívnych dopytov je tiež sledovaný pomocou `TTLCache`, aby sa predišlo duplicitnému spúšťaniu agenta pre identické dopyty.



Obr. 5.7: Detailný pohľad na rozbaliteľnú sekciu „Peek into the AI's Thought Process“ (Nahliadnuť do myšlienkového procesu AI) na konci správy o výsledkoch RAG vyhľadávania.

5.3 Modul poskytujúci rozhranie pre funkcionalitu vyhľadávania a definícia dokumentov pre Vespa databázu

Vespa poskytuje Python SDK v podobe knižnice `pyvespa`[15]. Nasledujúce podsekcie detailne špecifikujú schému dokumentov `title_document`, ktorá bola navrhnutá na uchovávanie analyzovaných príbehov a implementovaná pomocou knižnice `pyvespa`. Implementácia sa nachádza v súbore `vespa/app.py`.

Definícia aplikačného balíka Vespa

Súbor `app.py` je zodpovedný za programatickú definíciu celého aplikačného balíka pre Vespa pomocou tried a metód poskytovaných knižnicou `pyvespa`.

[15] Dokumentácia Python knižnice `pyvespa`: https://vespa-engine.github.io/pyvespa/

Schéma dokumentu `title_document`. Táto schéma špecifikuje štruktúru, dátové typy a indexovacie stratégie pre každý atribút ukladaných príbehov. Schému je možné vidieť v prílohe B. Medzi základné polia patria metadáta ako `title_id`, `source` (zdroj príbehu, napr. „wiki“), `author`, `title` a `year`. Tieto polia sú indexované pre rýchle atribútové vyhľadávanie a polia `author` a `title` navyše podporujú fulltextové vyhľadávanie pomocou algoritmu BM25 [16] pre výpočet relevancie.

Podstatnú časť dokumentu tvoria polia určené na ukladanie textového obsahu a jeho štruktúrovanej analýzy. Polia ako `chunks` (segmenty pôvodného textu), `plot_summaries`, `setting_summaries`, `character_summaries` a `theme_summaries` sú definované ako polia reťazcov (`array<string>`) a sú taktiež indexované pre fulltextové vyhľadávanie (BM25), čo umožňuje lexikálne dopyty nad týmito časťami analýzy. K nim prislúchajú polia s identifikátormi segmentov (`chunk_ids`) a polia s názvami extrahovaných tém (`plot_topics`, `character_topics`, atď.).

Pre sémantické vyhľadávanie a porovnávanie na úrovni jednotlivých aspektov príbehu sú definované tenzorové polia s jednou mapovanou a jednou indexovanou dimenziou. Mapovaná dimenzia `topics{}` umožňuje asociovať s každým dokumentom (príbehom) viacero pomenovaných vektorov pre daný aspekt – napríklad jeden vektor pre každý identifikovaný dejový zvrat alebo kľúčovú udalosť. Indexovaná dimenzia `x[1024]` zodpovedá dĺžke vektora generovaného embedding modelom. Tieto tenzorové polia sú indexované pomocou HNSW algoritmu. Ako metrika vzdialenosti je pre HNSW indexy zvolená `angular`, čo zodpovedá kosínusovej podobnosti, vhodnej pre normalizované sémantické vektory. Rovnaké tenzorové polia sú definované aj pre aspekty postáv, prostredia a tém.

Profily relevancie. Pre implementovanie navrhnutých profilov (sekcia 4.4) bolo nutné implementovať sadu tenzorových funkcií: `sim_per_topic`, `avg_sim`, `max_sim`.

Funkcia `sim_per_topic` počíta kosínusovú podobnosť (implementovanú ako skalárny súčin pre normalizované vektory) medzi poskytnutým vektorom `query_embedding` a každým jednotlivým vektorom uloženým v mapovanej dimenzii `topics{}` tenzorového poľa dokumentu (napr. `attribute(plot_embeddings)`). Výsledkom je tenzor skóre podobnosti pre každú tému daného aspektu. Definuje sa výrazom:

```
sum(query_embedding * attr_embedding, x).
```

Funkcia `avg_sim` spriemeruje hodnoty pozdĺž dimenzie `topics`, čím poskytuje jedno priemerné skóre podobnosti pre celý aspekt. Definuje sa ako:

```
reduce(sim_per_topic(query_embedding, attr_embedding), avg, topics).
```

Funkcia `max_sim` vyberie maximálne skóre podobnosti spomedzi všetkých tém daného aspektu. Definuje sa ako:

```
reduce(sim_per_topic(query_embedding, attr_embedding), max, topics).
```

Tieto funkcie sú následne využité vo `first_phase` výrazoch jednotlivých profilov relevancie na implementáciu navrhovaných stratégií. Aspektovo-špecifické profily ako `semantic_plot` vo svojej prvej fáze nepoužívajú priamo definované funkcie ale `cos(distance(field, plot_embeddings))` pre ANN vyhľadávanie. Vespa však poskytuje mechanizmus ako sprístupniť výstupy definovaných tenzorových výrazov vo výsledkoch pre prezentačnú

[16] Vespa implementácia algoritmu BM25: https://docs.vespa.ai/en/reference/bm25.html.

vrstvu cez atribút `match-features`[17]. Holistické profily priamo kombinujú výsledky funkcie `avg_sim` pre všetky štyri aspekty vo svojom `first_phase` výraze:

```
avg_sim(q, plot_embs)*query(w_plot)
+ avg_sim(q, setting_embs)*query(w_setting)
+ avg_sim(q, theme_embs)*query(w_theme)
+ avg_sim(q, character_embs)*query(w_character),
```

kde `query(w_<aspekt>)` sú váhy jednotlivých aspektov definované v dopyte. Hybridný profil s názvom `hybrid_semantic` kombinuje BM25 skóre s `avg_sim` hodnotami pre jednotlivé aspekty. Všetky sémantické a hybridné profily vyžadujú ako vstupné parametre dopytu príslušné vektorové reprezentácie (napr. `query(q_embedding)`). Posledným definovaným profilom je `default`, ktorý zabezpečuje čisto lexikálne vyhľadávanie algoritmom BM25.

Klient pre Vespa databázu (`TitleVespaClient`).

Trieda `TitleVespaClient`, implementovaná v súbore `backend/vespa.py`, tvorí abstrakčnú vrstvu pre komunikáciu s nasadenou Vespa aplikáciou. Zapuzdruje logiku pre konštrukciu dopytov, ich odosielanie a spracovanie odpovedí.

Inicializácia a pripojenie. Pri svojej inicializácii sa `TitleVespaClient` pripája k Vespa inštancii, ktorej URL adresa je špecifikovaná premennou prostredia `VESPA_APP_URL`. Po úspešnom nadviazaní spojenia je klient pripravený prijímať a spracovávať požiadavky.

Metóda `search`. Hlavnou metódou klienta je `search`, ktorá slúži ako centrálny bod pre všetky typy vyhľadávacích operácií. Táto metóda dynamicky konštruuje a vykonáva YQL (Vespa Query Language) dopyty na základe poskytnutých parametrov. Kľúčovou súčasťou sémantického vyhľadávania je príprava vektorových reprezentácií pre dopyt, ktorú zabezpečuje metóda `prepare_embeddings_for_query`.

Spracovanie a formátovanie výsledkov. Po prijatí odpovede od Vespa servera (objekt `VespaQueryResponse`), metóda `format_query_results` parsuje prijatý JSON, extrahuje jednotlivé dokumenty (hits). Významnou súčasťou tohto spracovania je metóda `get_sortings`, ktorá analyzuje polia `summaryfeatures` a `matchfeatures` v odpovedi. Tieto polia obsahujú detailné skóre podobnosti pre jednotlivé témy (výsledok funkcie `sim_per_topic`), ktoré sú extrahované, agregované (ak je to potrebné) a uložené do atribútu `inter_topic_sorting` v objekte `Title`. Tento atribút tak poskytuje granulárnu informáciu o tom, ktoré konkrétne časti (témy) príbehu najviac prispeli k jeho celkovej relevancii vzhľadom na dopyt, čo je nevyhnutné pre ich vizualizáciu na strane používateľského rozhrania.

Ďalšie operácie. Okrem vyhľadávania poskytuje `TitleVespaClient` aj metódy na priame načítanie detailov dokumentov podľa ich ID (`get_document_details`, `get_stories_by_ids`) a na získavanie návrhov pre automatické dopĺňanie dopytov pri písaní (`get_suggestions`). Všetky tieto operácie sú implementované asynchrónne.

[17] Odkaz na detailnú špecifikáciu pre atribút match-features: https://docs.vespa.ai/en/reference/schema-reference.html#match-features.

5.4 Implementácia LLM komponent pomocou rámca Adalflow

Implementácia komponent systému, ktorých jadrom je LLM, je realizovaná pomocou rámca Adalflow (obr. 5.9). Základom systému sú komponenty **Generator**, **Retriever** a **Embedder**. Rozhranie týchto komponentov poskytuje dve hlavné metódy: `call` a `forward`. Metóda `forward` sa od metódy `call` líši v podstate len výstupným typom, ktorým je `adalflow.Parameter`, ktorý slúži na obalenie a označenie vstupov, výstupov a častí promptu, ktoré má systém sledovať pri prechode výpočtovým grafom. To umožňuje spätnú propagáciu gradientov počas trénovania. Systém využíva Adalflow na orchestráciu a optimalizáciu LLM komponentov. Má špeciálne atribúty:`requires_opt=True`, `role_desc` (popisujúci účel parametra pre optimalizátor) a `param_type` (napr. `ParameterType.PROMPT` alebo `ParameterType.DEMOS`). Tieto atribúty signalizujú triede `adalflow.Trainer`, ako má s daným parametrom zaobchádzať. Súčasťou práce s LLM je definícia a návrh inštrukcií. Ich detailnejší popis a štruktúra je obsahom prílohy D.

Samotný proces volania LLM je abstrahovaný pomocou `ModelClient` rozhrania z Adalflow, čo umožňuje flexibilné použitie rôznych API poskytovateľov LLM (lokálnych cez `SentenceTransformerClient` alebo externých cez `LiteLLMClient`).

Pre prácu so štruktúrovanými dátami (vstupmi a výstupmi LLM) sa využíva `DataClassParser` v spojení s definovanovanými dátovými triedami (napr. `StoryAnalysisOutput`). Parser generuje inštrukcie pre formátovanie výstupu (napr. JSON schéma odvodená z dátovej triedy), ktoré sú súčasťou promptu, a zároveň parsuje a validuje odpoveď LLM.

Trénovateľná pipeline v Adalflow sa buduje prepojením komponentov, kde metóda `forward` každého komponentu prijíma a vracia `Parameter` objekty. Tým sa vytvára výpočtový graf. `adalflow.Trainer` následne riadi tréningový cyklus. Ak je parameter (napr. segment promptu) označený pre optimalizáciu, `Trainer` využíva „backward engine“ (LLM) na výpočet textových gradientov – spätnej väzby, ako upraviť dáta parametra (napr. preformulovať časť promptu) s cieľom minimalizovať stratovú funkciu(loss).

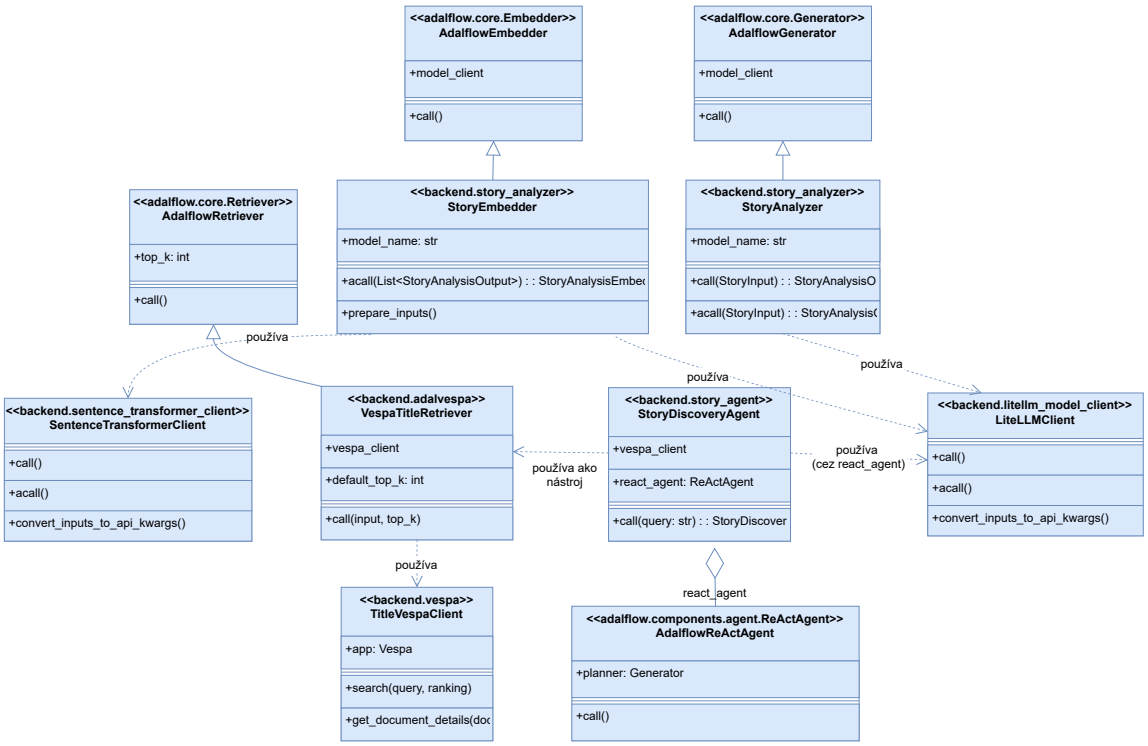
`Optimizer` (napr. `TGDOptimizer` pre textové gradienty alebo `BootstrapFewShot` pre generovanie few-shot príkladov) potom na základe týchto gradientov navrhuje nové hodnoty pre optimalizované parametre.

Integrácia LiteLLM do frameworku Adalflow

Rámec Adalflow pracuje s LLM cez rozhranie definované triedou `ModelClient`. Z tohto dôvodu bolo nutné implementovať adaptér pre rozhranie LiteLLM. Úlohou tohto adaptéra je volanie modelov, príprava parametrov volaní a spracovanie výstupu do požadovaných dátových modelov rámca Adalflow. Jednotné rozhranie taktiež umožnilo implementovať mechanizmus dodržiavania stanovených limitov API poskytovateľmi, ktorý zabezpečil preventívne dodržiavanie týchto limitov, či už pri synchrónnej alebo asynchrónnej komunikácii s API. Táto funkcionality bola implementovaná pomocou knižnice `limits`[18]. Ide o spoľahlivú implementáciu limitov pre jednovláknové, asynchrónne, ale aj viacvláknové programy[19]. Pre synchrónne volania embedding modelov cez Ollama bola implementovaná špecifická metóda

[18]Knižnica limits: https://limits.readthedocs.io

[19]Test Python knižníc pre implementáciu limitov: https://gist.github.com/justinvanwinkle/d9f04950083c4554835c1a35f9d22dad.



Obr. 5.8: Diagram tried implementovaných pomocou rámca Adalflow. Zobrazené sú len podstatné metódy a atribúty jednotlivých tried, ktoré implementujú navrhnutú funkcionalitu.

`ollama_embed` v rámci `LiteLLMClient`, ktorá obchádza problémy spojené s manažovaním asyncio slučiek udalostí pri štandardnom volaní cez LiteLLM na niektorých platformách.

Analýza príbehu (`StoryAnalyzer`)

Komponent `StoryAnalyzer`, odvodený od `adalflow.Generator`, je zodpovedný za transformáciu textu príbehu na štruktúrovanú analýzu (`StoryAnalysisOutput`). Jadrom jeho funkcionality je prompt, ktorý definuje úlohu pre LLM – extrakciu deja, postáv, prostredia a tém – a striktne predpisuje formát výstupu a metodológiu citovania. Vstupný text, formátovaný ako XML-štruktúra s tagmi `<story>` a `<chunk>`, umožňuje LLM identifikovať a referencovať segmenty textu pomocou citácií `[cite: ID_segmentu]`.

Spracovanie výstupu LLM triedou `adalflow.DataClassParser` zabezpečuje validáciu voči požadovanej schéme a transformáciu na triedu `StoryAnalysisOutput`. V rámci Adalflow je možné časti promptu, ako napríklad systémové inštrukcie alebo príklady, definovať ako objekty `adalflow.Parameter` s atribútom `requires_opt=True`, čo umožňuje ich automatickú optimalizáciu počas tréningového cyklu.

Vytváranie vektorových reprezentácií (`StoryEmbedder`)

Komponent `StoryEmbedder`, dediac od `adalflow.Embedder`, prijíma štruktúrovanú analýzu `StoryAnalysisOutput` a generuje sémantické embeddingy pre textové sumáre jednotlivých aspektov. Zaujímavým implementačným detailom je metóda `prepare_inputs`. Táto metóda rieši transformáciu hierarchickej štruktúry vstupných dát (príbeh `->` aspekt `->` zoznam položiek analýzy) na jednorozmerný (plochý) zoznam textov vhodný pre dávkové spracovanie

embedding modelom. Súčasťou tejto transformácie je aj interné uchovanie mapovania, ktoré umožňuje po vygenerovaní embeddingov priradiť každý vektor späť k jeho presnému pôvodu v hierarchickej štruktúre. Toto mapovanie je nevyhnutné pre korektné zostavenie výstupného objektu `StoryAnalysisEmbeddingOutput`, ktorý si zachováva pôvodnú štruktúru, ale textové popisy nahrádza ich vektorovými reprezentáciami.

Vyhľadávanie vo Vespe (`VespaTitleRetriever`)

Pre integráciu sémantického vyhľadávania v databáze Vespa do Adalflow modulu bol implementovaný komponent `VespaTitleRetriever` (`backend/adalvespa.py`), ktorý dedí od `adalflow.Retriever`. Tento komponent zapuzdruje inštanciu `TitleVespaClient` (popísanú v sekcii 5.3) a adaptuje jej metódu `search` pre rozhranie Adalflow.

Agent pre objavovanie príbehov (`StoryDiscoveryAgent`)

Komponent `StoryDiscoveryAgent` (súbor `backend/story_agent.py`) je komplexnejší komponent implementovaný ako potomok `adalflow.Component`, ktorý využíva `adalflow.ReActAgent`. Tento agent má za úlohu iteratívne vyhľadávať a sumarizovať príbehy na základe používateľského dopytu. Kľúčovým prvkom agenta je nástroj `VespaSearchToolComponent`, ktorý je odvodený od `adalflow.GradComponent` a obaľuje funkcionalitu `VespaTitleRetriever`. Tento nástroj je agentovi poskytnutý prostredníctvom `adalflow.FunctionTool` (trieda, ktorá slúži na prevod medzi funkciou a textovým opisom pre LLM), ktorý definuje jeho názov, popis a očakávané parametre pre LLM. Šablóna promptu pre `ReActAgent` inštruuje LLM, ako má postupovať: analyzovať dopyt, formulovať myšlienky (thought), vyberať akcie (action – volanie nástroja `search_stories_in_vespa` alebo ukončenie s `finish`) a spracovávať pozorovania (observation – výsledky z nástroja). Agent iteratívne vykonáva svoje vyhľadávania a analýzy, až kým nedosiahne uspokojivý výsledok alebo maximálny počet iterácií. Finálny výstup je štruktúrovaný ako `StoryDiscoveryOutput`, obsahujúci zoskupené odporúčania (`AgentStoryGroup`) a celkové zhrnutie.

Experimentálna pipeline pre vyhľadávanie dôkazov

Popri komponentoch pre analýzu a sémantické vyhľadávanie príbehov bola v rámci experimentov implementovaná aj pipeline zameraná na úlohu vyhľadávania dôkazov. Táto pipeline je postavená na komponente `EvidenceAgent`, ktorý je potomkom triedy `adalflow.Component` a vo svojom jadre využíva `adalflow.ReActAgent`. Úlohou tohto agenta je iteratívne prehľadávať poskytnuté dokumenty (segmenty kníh – `BookChunk`) a syntetizovať z nich dôkazy pre zodpovedanie vstupného dopytu.

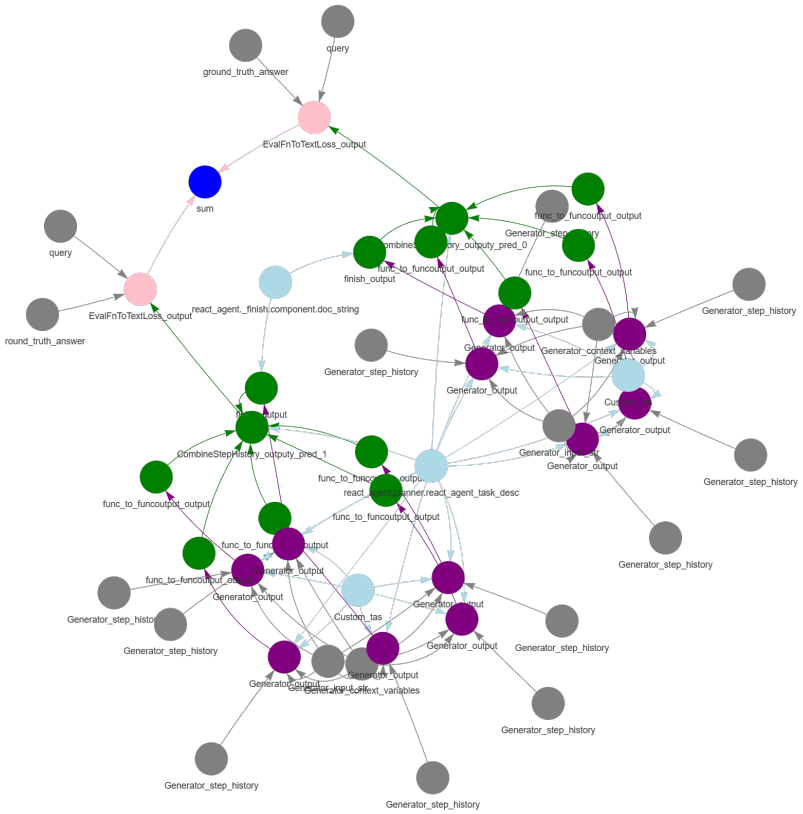
Agent disponuje sadou nástrojov nevyhnutných pre svoju činnosť. Medzi kľúčové patrí `MyFAISSRetriever` (obaľujúci `adalflow.Embedder` pre tvorbu vektorových reprezentácií dopytov a `FAISS` pre efektívne vyhľadávanie podobných segmentov). Okrem toho agent využíva špecifické nástroje na riadenie procesu vyšetrovania (investigation):

- `add_investigation_summary_tool` – umožňuje agentovi zaznamenať nový vyšetrovací dotaz spolu s odôvodnením a počiatočným zhrnutím,
- `update_investigation_tool` – slúži na aktualizáciu existujúceho vyšetrovania novými zisteniami a zhrnutiami na základe novo získaných pasáží,
- `get_investigations_tool` – poskytuje agentovi prístup k zoznamu všetkých aktuálne prebiehajúcich vyšetrovaní.

- `get_context_for_passages_tool` – umožňuje získať širší kontext okolo konkrétnej citovanej pasáže.

Výstupom `EvidenceAgent`-a je dátová trieda `AgentOutput`, ktorá obsahuje finálnu odpoveď a detailnú históriu krokov agenta (myšlienka, akcia, pozorovanie).

Pre účely tréningu a optimalizácie tohto agenta v rámci Adalflow je `EvidenceAgent` obalený triedou `AdalTask`. Evaluáciu výstupov agenta zabezpečuje komponent `EvidenceFindingLLMJudge`. Tento LLM sudca hodnotí kvalitu nájdených dôkazov na základe viacerých metrík, ako sú „Citation Accuracy“ (presnosť a podpora citácií), „Passage Relevance“ (relevancia citovaných pasáží k dopytu), „Reasoning Quality“ (kvalita argumentácie) a „Answer Correctness“ (správnosť finálnej odpovede áno/nie/nedá sa vylúčiť). Sudca generuje skóre pre každú metriku, ktoré sú následne normalizované do rozsahu [0, 1] (s výnimkou binárnej „Answer Correctness“).



Obr. 5.9: Vizualizácia výpočtového grafu `EvidenceAgent`-a pri jednom kroku optimalizácie (2 príklady na krok).

Kapitola 6

Dáta

Pri štúdiu problematiky sa ukázalo byť náročné nájsť vhodné dáta na testovanie porovnávania a vyhľadávania príbehov na základe podobností aspektov. Väčšina dátových sád využíva prerozprávania toho istého príbehu alebo preklady zhrnutí deja z iných jazykov (dataset „*Tell Me Again*“ [18]). Pre porovnanie paralel medzi príbehmi nie len celkovou podobnosťou nebola nájdená žiadna vhodná dátová sada, ktorá by sa venovala takejto úlohe. K testovaniu modelov určených k vytváraniu sémantických vektorových reprezentácií bol využitý dataset MovieRemakes (sekcia 6.1). Sekcia 6.3 sa venuje procesu tvorby a popisu novej dátovej sady, ktorá je určená pre vyhodnotenie implementovaného systému. Táto dátová sada obsahuje referenčné dopyty a cieľové príbehy, pričom dopyt sa zameriava na rôzne paralely medzi príbehmi. Pre vytvorenie sady príbehov určenej na porovnanie na úrovni naratívnych trópov, bolo preskúmané komunitné fórum TVTropes, ktoré bližšie popisuje sekcia 6.4.

6.1 Dátová sada MovieRemakes

MovieRemakes [7] je dátová sada vytvorená na hodnotenie naratívnej podobnosti textov. Dátový súbor pozostáva z dejových zhrnutí filmov extrahovaných z Wikipédie. Autori tejto dátovej sady, zozbierali zoznamy filmov z Wikipédia stránky „Lists of film remakes“[1]. Vychádza z predpokladu, že filmové remaky sú opätovnými prerozprávaniami toho istého príbehu a zachovávajú si podstatné naratívne prvky, aj keď sa môžu líšiť v niektorých detailoch. Systém určovania podobnosti by preto mal hodnotiť zhrnutia dejov filmových remakov ako navzájom podobné.

Štatistika	**Hodnota**
Počet filmov	577
Počet zhlukov	266
Maximálny počet filmov v zhluku	7
Priemerný počet slov v zhrnutí	564
Maximálny počet slov v zhrnutí	2778
Minimálny počet slov v zhrnutí	26

Tabuľka 6.1: Štatistiky pre dátový súbor MovieRemakes (prevzané z [7])

[1]Stránka na Wikipédii pre zoznam remakov: https://en.wikipedia.org/wiki/Lists_of_film_remakes.

6.2 Korpus WikiPlots

WikiPlots[2] je korpus, ktorý obsahuje 112 936 opisov deja extrahovaných z anglickej Wikipédie. Tieto príbehy sú získané z akéhokoľvek článku v anglickom jazyku, ktorý obsahuje podnadpis so slovom „plot“ (napr. „Plot“, „Plot Summary“ atď.). Tento súbor dát bol naposledy aktualizovaný v roku 2017. Korpus neobsahuje žiadne príslušné metadáta, len extrahovaný text a Wikipédia URL adresu článku. Manuálna kontrola ukázala niekoľko dát nesúvisiacich s naratívnym obsahom (napr. „List of Linyphiidae species (A–H)“). Na základe toho bolo vykonaná oprava pomocou metadát z WikiData[3]. Filtrovanie bolo zamerané na hodnotu atribúta instance-of[4], čím bolo odstránených 125 článkov. Analýzou dĺžky opisov (tabuľka 6.2) boli identifikované veľmi krátke opisy, tie sú počas analýzy filtrované, pričom je prah nastavený na 20 tokenov.

Tabuľka 6.2: Štatistiky korpusu WikiPlots. Dĺžka je udávaná v tokenoch, pričom bol použitý XLM-RoBERTa tokenizer [11].

Štatistika	**Hodnota**
Počet opisov	112 814
Priemerná dĺžka (token)	519.82
Minimálna dĺžka (token)	2
Maximálna dĺžka (token)	22 784

6.3 Datátova sada pre vyhodnotenie systému

Pre experimentálne účely bola zostavená dátová sada zameraná na úlohu sémantického vyhľadávania podobných príbehov. Tento súbor pozostáva z 3230 textových dotazov (queries), ktoré boli vytvorené s cieľom nájsť zodpovedajúce alebo príbuzné príbehy. Cieľovým korpusom, v ktorom sa vyhľadávalo, je kolekcia 2010 zhrnutí príbehov, získaných z anglickej Wikipédie, zahŕňajúcich zápletky kníh, filmov a seriálov. Každý dotaz je naviazaný na jeden alebo viacero relevantných príbehov z cieľového korpusu. K zostaveniu tejto dátovej sady bol využitý model gemini-2.5-pro-exp-03-25 a knižnica `Adalflow` (sekcia 5.1). Veľkú časť z dátovej sady tvoria tituly z datasetu MovieRemakes [7].

Základné štatistiky týkajúce sa dĺžky textov (merané počtom tokenov po základnej tokenizácii) pre dotazy a cieľové príbehy sú zhrnuté v tabuľke 6.3. Je zrejmé, že dotazy sú v priemere výrazne kratšie (priemerne 83 tokenov) ako samotné popisy zápletiek príbehov (priemerne 861 tokenov), ktoré vykazujú aj podstatne väčšiu variabilitu v dĺžke (štandardná odchýlka 510 tokenov oproti 30 tokenom pri dotazoch).

Proces tvorby datasetu

Pre tvorbu datasetu bol využitý model Gemini 2.5 Pro vo verzii s označením gemini-2.5-pro-exp-03-25[5]. Táto experimentálna verzia nebola spoplatnená, no mala obmedzený denný limit na 25 požiadaviek za deň, čo malo vplyv na veľkosť výslednej dátovej sady.

[2]Korpus WikiPlots: https://github.com/markriedl/WikiPlots.

[3]WikiData: https://www.wikidata.org/.

[4]Odkaz na dokumentáciu atribútu instance-of: https://www.wikidata.org/wiki/Property:P31.

[5]Model Gemini 2.5 Pro: https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-pro.

Tabuľka 6.3: Štatistiky pre 2 súbory dát a vzorku príbehov. Dĺžka je udávaná v tokenoch (s použitím XLM-RoBERTa tokenizer).

Štatistika	**Dopyty (Storysim)**	**Dopyty (Hard Lexical)**	**Príbeh**
Počet	3230	160	2010
Priemerná dĺžka (token)	82.9	29.0	860.6
Minimálna dĺžka (token)	29	17	28
Maximálna dĺžka (token)	215	46	5539
Ø relevantných príbehov	2.36	4.78	—
Distribúcia (%)			
Dej	27.5	26.2	—
Postavy	25.6	24.4	—
Prostredie	23.6	23.8	—
Téma	23.3	25.6	—

Dataset je určený pre testovanie finálneho systému, a tvoria ho najmä dlhšie popisy hľadaného príbehu alebo samotný príbeh. Schéma datasetu obsahuje nasledujúce polia:

- `query`: dotaz,
- `title_id`: najlepšie zodpovedajúci titul,
- `pos_title_ids`: veľmi relevantné tituly vzhľadom na dotaz,
- `negative_title_ids`: ťažké príklady, ktoré nie sú relevantné (majú niečo spoločné s pozitívnymi titulmi, no nie vzhľadom na dotaz),
- `aspect`: aspekt príbehu, na ktorý je zameraná podobnosť.

Cieľom pri tvorbe datasetu bolo využiť dostupné opisy dejov z Wikipédie a zhluky príbehov z MovieRemakes. Existujúce zhluky slúžili na jednoduchú kontrolu pri generovaní testovacej vzorky, ktorá bola manuálne kontrolovaná. Manuálne bolo kontrolovaných 50 vygenerovaných dotazov, pričom všetky z nich tvorili otázky, ktoré v rámci pozitívnych dokumentov obsahovali tituly, ktoré nepatrili do rovnakého zhluku podľa datasetu MovieRemakes. Tento prístup bol zvolený zámerne, aby sa otestovala schopnosť LLM nachádzať relevantné prepojenia na základe špecifikovaného aspektu aj mimo zjavných vzťahov (ako sú priame remaky). Táto kontrola neodhalila žiadne nedostatky pre model Gemini 2.5 Pro, no množstvo dát bolo obmedzené, preto v rámci ďalšej práce by bolo vhodné otestovať menej nákladné varianty modelov.

Pre zaistenie kvality a diverzity generovaných otázok a párov cieľových dokumentov bol navrhnutý vstupný prompt (ktorý možno vidieť v prílohe F), ktorý sa skladal z:

- z inštrukcií pre štýl dotazov,
- z množiny náhodne vzorkovaných príbehov z datasetu WikiPlots a MovieRemakes,
- z manuálne vybraných ukážkových príkladov výstupu.

Pravidelná zmena inštrukcií pre štýl dotazov a ukážkové príklady pomáhali diverzifikovať generované dopyty.

Vzorka obsahujúca krátke anonymizované dopyty

Cieľom testovania je vyhodnotiť schopnosť hľadať podobnosti aj na úrovni jednotlivých aspektov príbehu, z tohto dôvodu bola vytvorená menšia vzorka, ktorá sa skladala z krátkych dopytov, ktoré popisujú určitý významný aspekt príbehu (postavu, dej, prostredie, tému). Táto vzorka má za úlohu otestovať potenciálne výhody sémantických profilov a analýzy. Dopyty boli navrhnuté tak, aby čo najviac anonymizovali zdrojové resp. cieľové príbehy, čo by malo mať negatívny vplyv na výsledky lexikálnych profilov. Základný prehľad štatistík pre túto vzorku (Hard Lexical) je možné vidieť v tabuľke 6.3.

Dopyt: *An isolated American research outpost in Antarctica is thrown into chaos when a helicopter from a nearby Norwegian base pursues a sled dog towards them. After the Norwegians are killed in a confrontation, the dog reveals itself to be a deadly extraterrestrial organism capable of perfectly mimicking any life form it absorbs. The crew must battle intense paranoia and distrust as they try to identify and destroy the creature among them before it can escape to the rest of the world.*

Relevantné tituly: The Thing (1982 film), The Thing from Another World.

Dopyt: *a protagonist driven to exact personal vengeance outside the law after a profound personal loss or injustice.*

Relevantné tituly: Death Sentence (2007 film), Law Abiding Citizen, Django (1966 film), Get Carter.

Obr. 6.1: Príklady dopytov ilustrujúce typ obsahu a relevantné tituly v dátových sadách Storysim (vľavo) a Hard Lexical (vpravo).

6.4 Komunitné fórum pre analýzu príbehov

Podobný prístup k zoskupovaniu opakujúcich sa naratívnych elementov a motívov, i keď nie tak systematicky formálny ako Proppova morfologická analýza rozprávok 3.1, sa dá nájsť aj na populárnej webovej stránke TVTropes.[6] Táto stránka zhromažďuje a kategorizuje rôzne opakujúce sa prvky (tzv. *tropes*), ktoré sa v literatúre, filmoch, televíznych seriáloch a iných umeleckých dielach neustále opakujú.

TVTropes používa otvorený komunitný systém značkovania, kde každý tróp odkazuje na konkrétnu typickú scénu, zvrat alebo vlastnosť postavy. V praxi to znamená, že k jednému príbehu (knihe, filmu atď.) môže na TVTropes existovať zoznam desiatok až stoviek tropes. Ich povaha je však veľmi rôznorodá. Mnohé z nich sa netýkajú priamo vnútornej štruktúry príbehu, ale odkazujú na externé alebo povrchné aspekty, ako napríklad *Actor Allusion* (odkaz na inú rolu herca), *Awesome Music* (hodnotenie hudby divákmi) či *Production Gag* (vtip súvisiaci s produkciou). Takéto trópy sú pre naratologickú analýzu zameranú na porovnávanie deja či tém zvyčajne irelevantné.

Ďalšou kategóriou trópov sú tie, ktoré popisujú samotné naratívne mechanizmy a postavy, napríklad *Mugging the Monster* (útok na nečakane silnú postavu) alebo *Jerk with a Heart of Gold* (archetyp postavy s drsným zovňajškom a dobrým srdcom). Hoci tieto trópy zachytávajú rozpoznateľné prvky týkajúce sa deja či postáv, ich častý výskyt a všeobecnosť obmedzujú ich užitočnosť pre hľadanie podobných príbehov. Samotná prítomnosť takýchto veľmi bežných trópov nemusí byť dostatočne špecifickým ukazovateľom na zmysluplné odlíšenie príbehov alebo na identifikáciu hlbších štrukturálnych podobností.

[6] Stránka TVTropes: https://tvtropes.org

Preto, aby bol TVTropes zdroj dát prakticky použiteľný pre porovnávanie príbehov, je nevyhnutné nielen odfiltrovať jasne irelevantné trópy, ale aj vyberať spomedzi relevantných trópov tie, ktoré sú vhodné pre konkrétny cieľ analýzy – či už ide o hľadanie univerzálnych prvkov (kde môžu byť užitočné aj všeobecnejšie trópy) alebo špecifických štrukturálnych či tematických paralel (kde sú potrebné konkrétnejšie trópy).

Definícia trópu má v priemere 272 slov a vysvetlenie ako je daný tróp prítomný v konkrétnom titule je v priemere 64 slov dlhé. Príloha C obsahuje ukážku definície a vysvetlenia ako sa tróp prejavuje v danom diele.

Kategória	**Tituly (s metadátami)**	**Trópy**	**Príklady**
Literatúra	15495 (5208)	27229	679618
Film	17019 (8816)	27450	751594
TV	7921 (4192)	27134	488632
Celkom	**40435 (18216)**	**29457**	**1919844**

Tabuľka 6.4: Tabuľka zobrazujúca počty titulov, tropes a príkladov podľa kategórií.

Využitie pre experiment vyhľadávania úryvkov z textu

Aj napriek tomu, že TVTropes nie je vhodnou dátovou sadou pre porovnávanie podobnosti príbehov, bol tento zdroj dát v práci použitý pre vytvorenie dátovej sady na vyhodnotenie a testovanie optimalizácie RAG systému pre získavanie pasáží z dlhých textov, ktoré potvrdzujú výskyt daného trópu v príbehu. Dátovú sadu tvorí 50 kníh, ktoré boli manuálne vybrané na základe dostupnosti celého textu a obsahu vhodných trópov, ktoré by mohli slúžiť na určovanie podobnosti medzi príbehmi. Základné štatistiky tejto dátovej sady sú zhrnuté v tabuľke 6.5.

Tabuľka 6.5: Štatistiky dátovej sady (50 kníh).

Štatistika	**Hodnota**
Počet kníh	50
Min. dĺžka knihy (tokeny)	10200
Max. dĺžka knihy (tokeny)	1036800
Priem. dĺžka knihy (tokeny)	327328
Priem. trópov na knihu	1.50
Celkový počet inštancií trópov	75

Kapitola 7

Experimenty

Táto kapitola popisuje experimenty vykonané počas tejto práce na úlohách analýzy, porovnávania a vyhľadávania príbehov. Cieľom experimentov je vyhodnotiť, čoho sú schopné veľké jazykové modely v úlohách hľadania dôkazov s použitím techniky generovania podporovaného vyhľadávaním, aký vplyv má viac-aspektová analýza príbehov na výsledky vyhľadávania a hľadanie vhodného modelu pre vytvorenie vektorových reprezentácií.

7.1 Hľadanie dôkazov podporujúcich tvrdenie

Veľké jazykové modely sú schopné odpovedať na rôznorodé otázky s ohľadom na špecifický dopyt používateľa, ktorý by v tomto prípade mohol obsahovať aj celý text knihy, no ako veľmi môžeme takýmto odpovediam veriť? Spočiatku som navrhoval porovnávať príbehy práve týmto spôsobom, kedy by dotaz spočíval v špecifikácii príbehu alebo toho, akú podobnosť hľadáme, kde úlohou iteratívneho RAG systému je dekompozícia problému, iteratívne vyhľadávanie úryvkov (beam search) a sumarizácia jednotlivých vyhľadávaní. Cieľom tohto experimentu je vyhodnotiť schopnosť systému zloženého z týchto modulov. Pre zložitosť úlohy a s tým spojeného spracovania textového výstupu som ako formát výstupu zvolil Json schému, na ktorú sa spolieha aj implementácia rámca. To značne ovplyvnilo výber modelov, pretože museli podporovať štruktúrovaný výstup práve v tomto režime. Väčšina modelov s touto formou výstupu nemá problém, no ako ukázali výsledky pri počiatočnom testovaní základného RAG systému, modely s menším počtom parametrov ($\leq 32B$) majú výrazný počet chýb spojených s nesprávnym formátom výstupu.

Ďalšou limitáciou hlavne pre hodnotiaci a optimalizačný model je podpora dlhých vstupov, keďže modelu musia vojsť do kontextu sledované vstupy a výstupy, čo v prípade tohto iteratívneho RAG systému sa pohybuje v desiatkach tisíc tokenov na jednu vzorku dát, keďže pre vyhodnotenie sú relevantné aj pasáže, ktoré boli nájdené vrámci celého procesu. Na základe týchto požiadaviek bol ako hlavný optimalizačný a evaluačný model vybraný Gemini 2.0 Flash[1]. Tento model podporuje vstupy dlhé až milión tokenov. Výpočtová náročnosť spracovávania dlhých sekvencií a aj čas potrebný na generáciu dlhších výstupov, ďalej obmedzovali využitie lokálnych modelov pre generovanie odpovedí, preto aj pre túto časť systému bolo potrebné použiť API poskytovateľov.

Keďže táto práca je zameraná na analýzu príbehov, tak ako zdroj dát pre dataset bola manuálne vybraná vzorka fikcie, ktorá mala vhodný počet a druh anotácií v datasete TvT-

[1]Gemini 2.0 Flash: https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash.

ropes 6.4. Dôraz bol kladený na to, aby vybrané príklady trópov (opis toho ako boli použité v danom diele) bolo možné odvodiť zo samotného textu knihy, a teda aby vôbec bolo možné nájsť vhodné úryvky.

Limitované výsledky experimentu

Na účely tohto experimentu bol implementovaný agent `EvidenceAgent`, ktorého implementácia je popísaná v sekcii 5.4. Implementácia bola časovo náročná, keďže sa jednalo o nový rámec, ktorému chýbali podstatné časti v dokumentácii zaoberajúce sa popisom optimalizovateľných komponentov. Povaha úlohy zároveň limitovala možnosti detailnej evaluácie na mnohých modeloch. Optimalizácia modelu prebiehala počas 10 epoch na 64 trénovacích príkladoch. Maximálny počet iterácií agenta bol nastavený na 10 a maximálny počet pasáží vrátených z vyhľadávania taktiež na 10. Aj napriek týmto limitom táto sekcia analyzuje výsledky získané z jedného optimalizačného behu. Priemerné výsledky výkonnosti agenta pred touto optimalizáciou a po nej, hodnotené na testovacej vzorke 14 unikátnych trópov, sú zhrnuté v tabuľke 7.1.

Tabuľka 7.1: Priemerné výsledky modelu `gemini-2.0-flash` pred a po optimalizácii. Výsledky sú získané z piatich evaluácií na testovacej vzorke dát.

Fáza	Chybovosť (%)	Priem. skóre
Pred optimalizáciou	38.57	0.543
Po optimalizácii	18.57	0.712

Z výsledkov uvedených v tabuľke 7.1 je zrejmé, že optimalizácia viedla k merateľnému zlepšeniu výkonu agenta. Celkové priemerné skóre sa zvýšilo z 0,543 na 0,712. Toto skóre odzrkadľuje kvalitu generovaných príkladov trópov na základe evaluačných kritérií definovaných v sekcii 5.4. Ešte výraznejšie zlepšenie nastalo v redukcii chybovosti pre generáciu v Json formáte. Celkový počet chýb na 70 hodnotených vzorkách klesol z 27 na 13, čím sa chybovosť znížila z 38,57 % na 18,57 %. Tento aspekt naznačuje lepšiu schopnosť modelu dodržiavať požadované štrukturálne požiadavky po optimalizačnom procese, čo je kľúčové pre spoľahlivosť agenta v praktických aplikáciách. Porovnanie optimalizovaných promptov s počiatočnými je obsahom prílohy G.

7.2 Výber vhodného modelu pre generovanie vektorových reprezentácií

Za účelom nájsť vhodný embedding model bolo na vytvorenej dátovej sade, popísanej 6.3, vyhodnotených niekoľko vybraných modelov. Experiment bol zameraný primárne na modely s menšou pamäťovou náročnosťou, ktoré však dosahujú porovnateľné výsledky s tými najlepšími v rebríčku MTEB[2].

Medzi testované modely boli vybrané dva menšie modely `all-MiniLM-L6-v2` [57] a `all-mpnet-base-v2` [51], trénované pre širokú škálu úloh [46] a efektívne z rýchlosti inferencie; dva modely s väčším kontextom `BAAI/bge-m3` [9] a `NovaSearch/stella_en_1.5B_v5` [64] a varianta modelu E5 (intfloat/multilingual-e5-large-instruct) [56]. Model E5 instruct má schopnosť učenia z kontextu na základe inštrukcií pripojených k vstupnému do-

[2]MTEB Leaderboard: https://huggingface.co/spaces/mteb/leaderboard

pytu, čo umožňuje lepšie prispôsobenie embeddingov konkrétnej úlohe. Voľba týchto modelov bola motivovaná snahou pokryť spektrum od menších a rýchlych modelov až po väčšie modely s potenciálom pre vyššiu presnosť.

Výber metrík

Pri hodnotení systémov sémantického vyhľadávania je potrebné zvoliť metriky, ktoré adekvátne reflektujú relevanciu nájdených výsledkov. Pre túto úlohu sú obzvlášť dôležité metriky zamerané na hodnotenie usporiadaných zoznamov výsledkov, pretože používatelia očakávajú najrelevantnejšie príbehy na prvých pozíciách. Medzi štandardne používané metriky [37], ktoré sú zároveň vybrané pre všetky experimenty, ktoré testujú vyhľadávanie, patria:

- **nDCG@k (Normalized Discounted Cumulative Gain)** – táto metrika je preferovaná, pretože hodnotí kvalitu usporiadania výsledkov. Prisudzuje vyššie skóre systémom, ktoré umiestňujú relevantnejšie dokumenty (príbehy) na vyššie pozície v zozname výsledkov. Zohľadňuje aj rôzne stupne relevancie, ak sú k dispozícii, aj keď sa často používa aj s binárnou relevanciou (relevantný/nerelevantný). Keďže vytvorená dátová sada nemá kvantifikovanú relevantnosť používa sa s binárnou relevanciou.
- **MAP@k (Mean Average Precision)** – MAP poskytuje súhrnné hodnotenie kvality naprieč rôznymi úrovňami recallu a je citlivá na poradie všetkých relevantných dokumentov do hĺbky k. Poskytuje dobrý prehľad o schopnosti systému nájsť a správne zoradiť relevantné príbehy pre sériu dopytov.
- **Recall@k** – meria podiel relevantných príbehov nájdených v prvých k výsledkoch z celkového počtu relevantných príbehov v dátovej sade pre daný dopyt.
- **Precision@k** – udáva podiel relevantných príbehov medzi prvými k vrátenými výsledkami. Vysoká presnosť na prvých pozíciách spokojnosť používateľa, pretože zabezpečuje, že prvé zobrazené výsledky sú skutočne užitočné a zodpovedajú dopytu.
- **MRR@k (Mean Reciprocal Rank)** – táto metrika hodnotí priemernú prevrátenú hodnotu pozície prvého relevantného výsledku. Je obzvlášť užitočná v scenároch, kde používateľ hľadá jeden správny alebo najlepší výsledok a pravdepodobne prestane prehľadávať po jeho nájdení.

Experimentálne nastavenia a prostredie

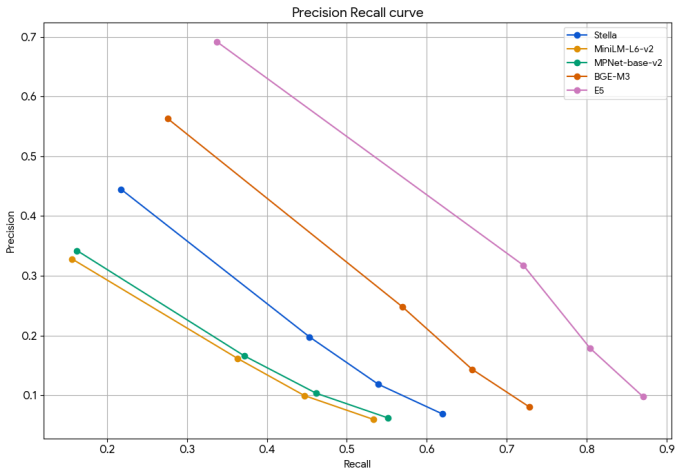
Pre generovanie vektorových reprezentácií boli použité predtrénované modely a knižnica Sentence Transformers[3]. Každý príbeh v dátovej sade bol transformovaný na vektorovú reprezentáciu pomocou zvoleného modelu, pričom dlhšie vstupné sekvencie boli skrátené na maximálnu alebo vybranú dĺžku. Rovnako boli transformované aj testovacie dotazy.

Pre vyhľadávanie relevantných príbehov bola použitá metóda kosínusovej podobnosti medzi embeddingom dotazu a embeddingami všetkých príbehov v korpuse. Príbehy boli následne usporiadané podľa klesajúcej podobnosti a pre prvých K výsledkov boli vypočítané vyššie spomenuté metriky. Experimenty boli vykonané na grafickej karte NVIDIA RTX 4090 s 24 GB VRAM. Pre všetky experimenty bola použitá binárna relevancia (príbeh je buď relevantný alebo nie) na základe pripravených párov (dotaz, množina relevantných príbehov). Nasledujúce podsekcie prezentujú výsledky získané z jednotlivých experimentov.

[3]Knižnica Sentence Transformers:https://sbert.net/.

Porovnanie modelov pri rovnakej maximálnej dĺžke sekvencie a rovnakej inštrukcii

V tomto experimente boli všetky modely testované s jednotnou maximálnou dĺžkou vstupnej sekvencie nastavenou na 380 tokenov (kvôli MPNet modelu). Pre modely podporujúce inštrukcie bola použitá predvolená inštrukcia. Výsledky sú prezentované Precision-Recall krivkami na obrázku 7.1.

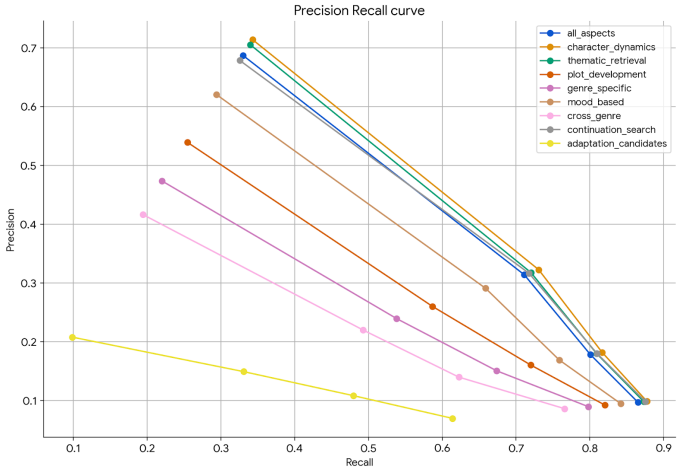


Obr. 7.1: Precision-Recall krivky pre $k \in \{1, 5, 10, 20\}$ porovnávaných embedding modelov pri jednotnej maximálnej dĺžke sekvencie 380 tokenov a použití predvolenej inštrukcie. Najlepšie výsledky dosahuje model `intfloat/multilingual-e5-large-instruct`.

Z výsledkov je zrejmé, že model `intfloat/multilingual-e5-large-instruct` dosahuje najlepšie hodnoty vo všetkých kľúčových metrikách, nasledovaný modelom `BAAI/bge-m3` a `NovaSearch/stella_en_1.5B_v5`. Menšie modely ako `all-MiniLM-L6-v2` dosahujú nižšie skóre, čo je očakávané vzhľadom na ich veľkosť a komplexnosť.

Vplyv inštrukcií na model E5-instruct

Experimenty s modelom intfloat/multilingual-e5-large-instruct ukázali, že formulácia inštrukcie pridanej k dotazu má merateľný vplyv na metriky sémantického vyhľadávania. Boli testované rôzne typy inštrukcií, od všeobecných po špecifickejšie pre danú úlohu. Precision-Recall krivky je možné vidieť na obrázku 7.2.



Obr. 7.2: Precision-Recall krivky pre rôzne inštrukcie v dopyte (multilingual-e5-large-instruct) pre $k \in \{1, 5, 10, 20\}$. Najlepšie metriky vykazuje inštrukcia `character_dynamics`, ktorá explicitne kladie dôraz na podobnosť postáv.

Tabuľka 7.2: Porovnanie najlepšej a najhoršej inštrukcie pre model multilingual-e5-large-instruct (@10)

Inštrukcia	nDCG	MAP	Recall	Precision	MRR
character_dynamics	**0.733**	**0.656**	**0.817**	**0.182**	**0.791**
adaptation_candidates	0.333	0.240	0.480	0.108	0.345

Výsledky potvrdzujú, že vhodne zvolená inštrukcia, ktorá lepšie špecifikuje úlohu pre model (napr. explicitné uvedenie, že sa má vyhľadať pasáž alebo príbeh) má významný vplyv, čo poukazuje na dôležitosť optimalizácie promptu/inštrukcie pri práci s inštrukčnými modelmi. Najlepšie výsledky boli dosiahnuté s inštrukciou, ktorá explicitne inštruuje na zameranie sa len na podobnosti medzi postavami a ich úlohami v príbehu (viď príloha E).

Vplyv veľkosti kontextu

Skúmal sa tiež vplyv maximálnej dĺžky sekvencie (veľkosti kontextu), ktorú je model schopný spracovať, na výkonnosť sémantického vyhľadávania. Porovnávali sa výsledky modelov pri použití ich predvolenej, potenciálne dlhšej dĺžky sekvencie oproti explicitnému nastaveniu maximálnej dĺžky na 380 tokenov. Táto analýza bola vykonaná pre modely Stella a BGE-M3. Výsledky pre tieto modely sú zhrnuté v tabuľkách 7.3 a 7.4.

Tabuľka 7.3: Stella: porovnanie metrík (@10) pri rôznych nastaveniach veľkosti kontextu.

Podmienka	nDCG	MAP	Recall	Precision	MRR
Max. kontext (131072)	**0.582**	**0.504**	**0.655**	**0.144**	**0.655**
Obmedzený kontext (380)	0.463	0.387	0.539	0.118	0.528

Tabuľka 7.4: BGE-M3: porovnanie metrík (@10) pri rôznych nastaveniach veľkosti kontextu.

Podmienka	nDCG	MAP	Recall	Precision	MRR
Max. kontext (8192)	0.522	0.431	0.620	0.137	0.590
Obmedzený kontext (380)	**0.574**	**0.486**	**0.657**	**0.143**	**0.655**

Z výsledkov je zrejmé, že vplyv zmeny maximálnej dĺžky sekvencie je závislý od konkrétneho modelu. V prípade modelu Stella viedlo obmedzenie maximálnej dĺžky sekvencie na 380 tokenov k zníženiu všetkých sledovaných metrík (napr. nDCG@10 kleslo z 0.582 na 0.463). Naopak, u modelu BGE-M3 malo toto skrátenie kontextu na 380 tokenov pozitívny dopad na výkonnosť, kde nDCG@10 stúplo z 0.522 na 0.574. Tieto pozorovania naznačujú, že optimálna veľkosť kontextu môže byť pre rôzne architektúry embedding modelov odlišná.

Zhrnutie experimentov pre embedding modely

Na základe vykonaných experimentov možno vyvodiť niekoľko záverov. Model `intfloat/multilingual-e5-large-instruct` vykazoval jednoznačne najlepšie výsledky na všetkých sledovaných metrikách spomedzi testovaných modelov pre úlohu sémantického vyhľadávania príbehov na danej dátovej sade. Dokázal prekonať ostatné modely aj pri použití rovnakej, obmedzenej maximálnej dĺžky sekvencie (380 tokenov).

Experimenty tiež potvrdili, že formulácia inštrukcie pri použití modelu E5-instruct má vplyv na výslednú kvalitu vyhľadávania. Rozdiely boli významné medzi najhoršou a najlepšou inštrukciou. Toto zistenie potvrdzuje dôležitosť návrhu promptov a efektivitu tohto prístupu k trénovaniu modelov.

Prekvapivo, modely s výrazne väčšou maximálnou dĺžkou sekvencie (napr. BAAI/bge-m3) nepriniesli v tomto konkrétnom prípade lepšie výsledky ako multilingual-e5-large-instruct s maximálnou dĺžkou 514 tokenov. To môže byť spôsobené charakterom dátovej sady, kde relevantné informácie pre dotazy sú obsiahnuté v relatívne kratších úsekoch textu.

Celkovo výsledky naznačujú, že pre sémantické vyhľadávanie príbehov je model `intfloat/multilingual-e5-large-instruct` najlepšou voľbou spomedzi testovaných modelov, čo potvrdzuje aj jeho umiestnenie v rebríčku MTEB.

7.3 Testovanie navrhnutých Vespa profilov relevancie

Po výbere optimálneho embedding modelu `intfloat/multilingual-e5-large-instruct` a spracovaní analýz pomocou vybraných LLM modelov sa pristúpilo k experimentom s rôznymi ranking profilmi. Tieto profily definujú, ako sa počítajú skóre relevancie pre jednotlivé príbehy vzhľadom na používateľský dopyt, a teda priamo ovplyvňujú poradie výsledkov. Metriky použité na hodnotenie sú rovnaké ako v predchádzajúcej sekcii. Inštrukcia pre embedding model bola počas týchto experimentov rovnaká pre všetky dopyty. Pre experiment bola nasadená Vespa databáza verzie v8.516.26 vo forme Docker kontajnera.

Boli testované nasledujúce typy ranking profilov:

- **Lexikálne profily**: profily využívajúce výlučne lexikálne skóre BM25 vypočítané z textových polí:
  - `default`: využíva polia extrahovaných tém, pole pôvodných segmentov, meno autora a názov titulu;

  - `bm25_analysis`: využíva iba polia analyzovaných tém príbehu,
  - `bm25_chunks`: využíva iba pole segmentov z textu príbehu a slúži na porovnanie vplyvu analýzy na výsledné metriky.

- **Sémantické profily (podľa poľa)**: profily, ktoré hodnotia príbehy na základe kosínusovej podobnosti medzi embeddingom dopytu a embeddingami špecifického sémantického poľa príbehu. Boli testované samostatné profily pre polia:
  - `semantic_character`: využíva iba embeddingy postáv,
  - `semantic_plot`: využíva iba embeddingy deja,
  - `semantic_setting`: využíva iba embeddingy prostredia,
  - `semantic_theme`: využíva iba embeddingy tém,
  - `semantic_chunk`: relevancia vypočítaná iba na základe embeddingov segmentov z textu pôvodného príbehu.

- **Sémantické (agregované)**: profily kombinujúce sémantické skóre z viacerých polí:
  - `avg_semantic`: výsledné skóre je priemerom kosínusových podobností zo všetkých štyroch sémantických polí (postava, dej, prostredie, téma),
  - `max_semantic`: výsledné skóre je maximálnou hodnotou kosínusovej podobnosti spomedzi všetkých štyroch sémantických polí,
  - `max_semantic_chunk`: kombinuje `max_semantic` a `semantic_chunk` k výpočtu relevancie.

- **Hybridný profil** (`hybrid_semantic`): tento profil kombinuje BM25 skóre so sémantickými skóre. Konkrétne, celkové skóre je súčtom výrazu z `default` profilu, priemerného sémantického skóre vypočítaného pomocou funkcie `avg_sim` a sémantických skóre pre jednotlivé aspekty.

Štatistiky LLM analýz príbehov

Pre kvantitatívne porovnanie schopností veľkých jazykových modelov pri štruktúrovanej analýze bolo vyhodnotených päť modelov. Výber modelov pre analýzu reflektoval snahu o porovnanie rôznych veľkostí, ceny a poskytovateľov. Zahŕňal modely Gemini od Google: `gemini-1.5-flash-8b`, `gemini-2.0-flash` a experimentálny `gemini-2.5-flash-preview-04-17` (poskytovateľ VertexAI[4]). Doplnili ich open-source alternatívy, konkrétne `llama-3.1-8b-instant` od Meta (poskytovateľ Groq API[5]) a `qwen-qwq-32b` od Alibaba (poskytovateľ Groq API). Parametre použité pre každý model sú uvedené v Tabuľke 7.5.

Tabuľka 7.6 sumarizuje metriky, ktoré odzrkadľujú detailnosť a úplnosť analýz jednotlivých modelov. Tieto faktory priamo vplývajú na spoľahlivosť následného sémantického vyhľadávania a kvalitu reprezentácie príbehov. Model `gemini-2.5-flash-preview-04-17`[6] vykazuje najvyššiu granularitu analýzy (priemerne 17,13 tém na príbeh) a generuje najobsažnejšie popisy tém (priemerne 324,2 znakov). Spolu s modelom `qwen-qwq-32b` (0,10% príbehov s chýbajúcimi aspektmi) a `gemini-2.0-flash` (0,25%) taktiež patrí k najspoľahlivejším z hľadiska úplnosti pokrytia všetkých štyroch sledovaných aspektov. Naopak, modely

[4] VertexAI: https://cloud.google.com/vertex-ai.
[5] Groq API: https://groq.com/.
[6] Výstup bol generovaný v režime non-thinking.

Tabuľka 7.5: Použité nastavenia parametrov pre generovanie analýz pomocou LLM a cena za kompletnú analýzu 2010 titulov. Referečný počet input/output tokenov získaný z analýzy modelom gemini-2.0-flash je 4M/2.8M.

Model	**API**	**teplota**	**top_p**	**cena (€)**
`gemini-1.5-flash-8b`	VertexAI	1.0	0.95	0.43
`gemini-2.0-flash`	VertexAI	0.5	0.95	1.27
`gemini-2.5-flash-preview-04-17`	VertexAI	0.5	0.95	2.47
`llama-3.1-8b-instant`	Groq API	1.0	1.0	0.56
`qwen-qwq-32b`	Groq API	0.6	0.95	6.62

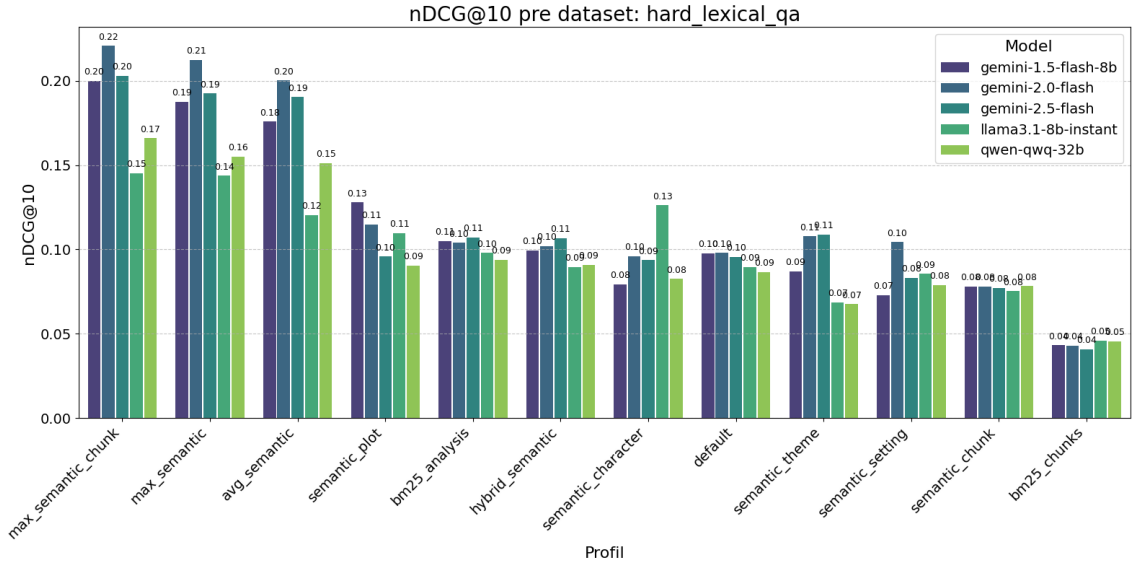
`gemini-1.5-flash-8b` a `llama-3.1-8b-instant` nielenže častejšie vynechávali niektoré aspekty (4,38% a 1,49% príbehov s chýbajúcimi aspektmi), ale v niektorých prípadoch generovali témy s nulovou dĺžkou textu, čo poukazuje na potenciálne problémy so spoľahlivosťou ich výstupov pre konzistentnú analýzu. Distribúcia tém naprieč aspektmi bola u väčšiny modelov najvyššia pre dej, nasledovaný postavami. Model `qwen-qwq-32b` generoval najkratšie popisy tém, čo môže znamenať aj nižšiu mieru detailu. Ďalším problémom modelu bolo generovanie validného Json výstupu, kde open-source modely zlyhávali, čo v prípade modelu `qwen-qwq-32b` výrazne predražilo analýzu príbehov oproti ostatným modelom a očakávanej cene ($\approx 1,17$€).

Tabuľka 7.6: Porovnanie štatistík analýzy uskutočnenej jednotlivými LLM na korpuse 2010 príbehov.

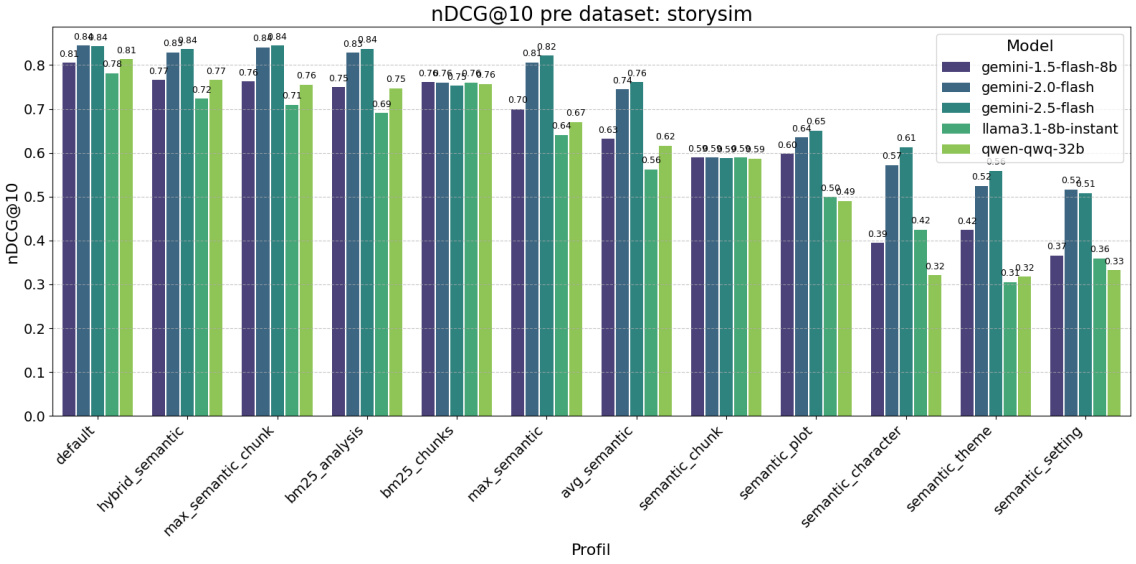
Model	Celkom tém	Ø tém/príbeh	Dej/postavy/prostredie/motív	Ø dĺžka (znaky)	Chýb. aspekty (% príbehov)
`gemini-1.5-flash-8b`	25680	12,78	9983/6853/3769/5075	263,2	4,38%
`gemini-2.0-flash`	27586	13,72	9344/7618/4713/5911	252,8	0,25%
`gemini-2.5-flash-preview-04-17`	33921	17,13	10905/10030/5276/7710	324,2	0,20%
`llama-3.1-8b-instant`	21932	11,31	9145/6353/3026/3408	200,1	1,49%
`qwen-qwq-32b`	23389	11,65	8096/6452/4328/4513	155,5	0,10%

Výsledky vyhodnotenia ranking profilov

Evaluácia navrhnutých ranking profilov prebehla na dvoch dátových sadách, Storysim a Hard Lexical (sekcia 6.3), ktoré sa líšia charakterom dopytov a náročnosťou pre jednotlivé typy profilov. Pre porovnávanie bola zvolená metrika nDCG@10.



Obr. 7.3: Porovnanie metriky nDCG@10 pre jednotlivé profily relevancie na dátovej sade Hard Lexical. Výsledky sú zobrazené pre každý profil v závislosti od LLM modelu použitého na generovanie analýz príbehov. Model `gemini-2.0-flash` dosahoval najlepšie výsledky pri použití metód `max_semantic_chunk` a `max_semantic` na tejto náročnejšej sade.



Obr. 7.4: Porovnanie metriky nDCG@10 pre jednotlivé profily relevancie na dátovej sade Storysim. Výsledky demonštrujú vplyv LLM modelu použitého na analýzu príbehov na výkonnosť profilov. Profily ako `default` a `max_semantic_chunk` dosahujú vysoké skóre, obzvlášť pri použití analýz z modelov `gemini-2.5-flash` a `gemini-2.0-flash`.

Vplyv analýzy na výsledky

Celkovo sa potvrdilo, že LLM analýza prináša pridanú hodnotu, obzvlášť na sade **Hard Lexical**. Výsledky sú však závislé od kvality analýzy a prínos menších modelov nie je jednoznačný.

Obrázok 7.3 poskytuje prehľad výsledkov pre všetky testované konfigurácie na dátovej sade **Hard Lexical**. Na tejto sade sa ukázalo, že pre úlohy vyžadujúce si porozumenie dopytu, ktorý neobsahuje kľúčové slová z hľadaného obsahu a je krátky, sú sémantické profily potrebné. Kombinované profily ako `max_semantic_chunk` a `max_semantic` jednoznačne dominovali, pričom kvalita LLM analýzy bola rozhodujúca. Analýzy z modelu `gemini-2.0-flash` viedli k najvyšším hodnotám nDCG@10 pre tieto profily, čo naznačuje optimálnu rovnováhu medzi detailnosťou analýzy a jej efektívnym využitím v ranking funkciách pre tento typ dopytov. Aj keď model `gemini-2.5-flash-preview-04-17` produkoval najgranulárnejšie analýzy (viď tabuľku 7.6), jeho mierne vyššia granularita sa na tejto sade vždy nepretavila do lepších výsledkov v porovnaní s `gemini-2.0-flash` pre najlepšie sémantické profily, čo môže súvisieť s povahou dopytov alebo špecifikami interakcie s ranking funkciou (výber len najviac podobnej témy pre max varianty). Výkonnosť open-source modelov (`llama-3.1-8b-instant`, `qwen-qwq-32b`) a menšieho `gemini-1.5-flash-8b` bola pri sémantických profiloch na tejto náročnej sade citeľne nižšia, čo koreluje s ich štatistikami analýz (napr. vyšší podiel chýbajúcich aspektov alebo kratšie popisy tém). Profil `bm25_analysis`, využívajúci LLM extrahované témy pre lexikálne skórovanie, konzistentne prekonával profily `bm25_chunks` (využívajúci pôvodné segmenty textu) a `semantic_chunk` (sémantická podobnosť pôvodných segmentov) pri všetkých modeloch použitých na analýzu. Toto poukazuje na prínos LLM aj pre vylepšenie tradičných lexikálnych metód prostredníctvom kvalitnejšej reprezentácie obsahu.

Evaluácia na dátovej sade **Storysim**, ktorá obsahuje dlhšie dopyty a častokrát podstatné kľúčové slová (mená postáv, miest a iných entít), ukázala adekvátnosť lexikálnych metód, ktoré sa držali na popredných miestach (pozri obrázok 7.4). Napriek tomu sa aj tu prejavil pozitívny vplyv LLM analýzy. Profily využívajúce polia analýzy pre výpočet podobnosti a relevancie prekonali `bm25_chunks`, no nie pri všetkých modeloch. Konkrétne ako konzistentne prispievajúce sa ukázali byť analýzy z modelov `gemini-2.0-flash` a `gemini-2.5-flash`. Analýzy z open-source modelov a menšieho modelu `gemini-1.5-flash-8b` neprekonali `bm25_chunks` pri niektorých sémantických profiloch (napr. `avg_semantic`), čo poukazuje na to, že prínos analýzy je priamo úmerný jej kvalite a schopnosti daného LLM efektívne extrahovať relevantné informácie. Avšak analýzy všetkých modelov mali pozitívny vplyv na metriky pri profile `default`, ktorý kombinuje analýzu a pôvodný text, pričom táto kombinácia prekonala `bm25_chunks` vo všetkých prípadoch.

Vyhodnotenie najlepšej testovanej konfigurácie

Pre lepšiu prehľadnosť sú najvýkonnejšie kombinácie LLM modelu pre analýzu a ranking profilu zhrnuté v tabuľke 7.7.

Tabuľka 7.7: Prehľad najvýkonnejších konfigurácií (LLM model + ranking profil) na základe metriky nDCG@10 a priemerného času vyhľadávania.

Dátová sada	LLM Model pre analýzu	Ranking Profil	nDCG@10	Odozva (ms)
Hard Lexical	`gemini-2.0-flash`	`max_semantic_chunk`	0.221	32.50
	`gemini-2.0-flash`	`max_semantic`	0.212	17.96
Storysim	`gemini-2.5-flash-preview-04-17`	`max_semantic_chunk`	0.845	63.57
	`gemini-2.0-flash`	`default`	0.844	45.58
	`gemini-2.5-flash-preview-04-17`	`default`	0.843	29.67

Analýza výsledkov ukázala, že optimálna konfigurácia (kombinácia LLM pre analýzu príbehov a ranking profilu) je výrazne závislá od charakteru dátovej sady a povahy používateľských dopytov.

Na dátovej sade **Hard Lexical**, ktorá predstavuje náročnejšie, často kratšie dopyty vyžadujúce sémantické porozumenie, dosiahli najlepšie výsledky agregované sémantické profily. Konkrétne, profil `max_semantic_chunk` v kombinácii s analýzou vygenerovanou modelom `gemini-2.0-flash` dosiahol najvyššiu hodnotu nDCG@10 (0,221). Tesne za ním nasledoval profil `max_semantic` s rovnakým LLM (nDCG@10: 0,212). Tieto výsledky zdôrazňujú prínos kombinovania sémantickej podobnosti z viacerých aspektov príbehu (dej, postavy, prostredie, téma) a pôvodných segmentov textu.

Naopak, na dátovej sade **Storysim**, charakteristickej dlhšími dopytmi, ktoré často obsahujú konkrétne kľúčové slová (napr. mená postáv, názvy miest), sa ukázala byť mimoriadne efektívna kombinácia lexikálnych a hybridných prístupov. Najvyššie hodnoty nDCG@10 dosiahli konfigurácie uvedené v tabuľke 7.7, kde dominujú profily `max_semantic_chunk` a `default` s analýzami z modelov `gemini-2.0-flash` a `gemini-2.5-flash-preview-04-17`. Vysokú výkonnosť na tejto sade dosiahol aj profil `hybrid_semantic`, najmä s analýzami z novších modelov Gemini. Tieto zistenia naznačujú, že pre dopyty s dostatočným množstvom lexikálnych signálov môže byť dobre nakonfigurovaný BM25 profil, prípadne jeho kombinácia so sémantickými skóre, najlepším riešením.

Zaujímavým zistením je, že profil `max_semantic_chunk` dosiahol porovnateľné výsledky s najlepším profilom na oboch sadách, no kvalita analýzy má veľký vplyv na jeho výkon (nDCG@10: 0.84 pre model `gemini-2.5-flash` a 0.71 `llama-3.1-8b`).

Kapitola 8

Záver

Porovnávanie podobnosti príbehov predstavuje komplexnú úlohu, ktorá si vyžaduje nielen porozumenie jazyka, ale aj schopnosť analyzovať naratívne štruktúry, tematické prvky a často aj interpretovať zámer autora. Táto diplomová práca si stanovila za cieľ preskúmať možnosti využitia moderných veľkých jazykových modelov (LLM) na vývoj spoľahlivého systému na porovnávanie a vyhľadávanie príbehov. Práca sa zamerala na integráciu pokročilých techník, ako je generovanie podporené vyhľadávaním (RAG) a optimalizačné metodologické rámce, s cieľom zvýšiť presnosť a interpretovateľnosť analýzy.

Teoretická časť práce poskytla systematický prehľad kľúčových oblastí, počnúc základmi veľkých jazykových modelov, ich architektúrami, cez rôznorodé prístupy k určovaniu podobnosti príbehov inšpirované naratológiou a metódami spracovania prirodzeného jazyka, až po techniky sémantického vyhľadávania a optimalizácie LLM systémov.

Hlavným cieľom praktickej časti bol návrh a implementácia systému, ktorý umožňuje vyhľadávanie a porovnávanie príbehov. Navrhnutá reprezentácia príbehu je založená na extrakcii a sumarizácii kľúčových informácií týkajúcich sa deja, postáv, prostredia a motívov. Tieto extrahované elementy sú následne transformované na sémantické vektorové reprezentácie a ukladané do databázy Vespa, ktorá podporuje efektívne viacvektorové vyhľadávanie a flexibilné definovanie relevancie prostredníctvom vlastných profilov. Systém integruje LLM nielen pre fázu analýzy, ale aj v rámci RAG modulu, koncipovaného ako agent schopný iteratívne vyhľadávať a sumarizovať podobnosti medzi príbehmi. Súčasťou systému je aj používateľské rozhranie, ktoré bolo navrhnuté ako interaktívna webová aplikácia. Toto rozhranie umožňuje exploratívne skúmať podobnosti medzi analyzovanými príbehmi a to na základe štandardného vyhľadávania, grafovej vizualizácie prepojení ale aj iteratívneho RAG modulu.

Experimentálna časť práce sa zamerala na vyhodnotenie modelov pre tvorbu sémantickej reprezentácie textu a rôznych profilov vyhľadávania. Potvrdilo sa, že model `intfloat/-multilingual-e5-large-instruct` je najvýkonnejším embedding modelom pre sémantické vyhľadávanie príbehov, pričom experimenty zdôraznili vplyv formulácie inštrukcií na výsledné metriky. Pri hodnotení ranking profilov v systéme Vespa sa výkonnosť výrazne líšila v závislosti od charakteru dátovej sady. Na dátovej sade Storysim, obsahujúcej dlhšie dopyty s kľúčovými slovami, exceloval lexikálny profil `default` (kombinujúci BM25 skóre z LLM analýzy a pôvodného textu), tesne nasledovaný hybridnými sémanticko-lexikálnymi prístupmi. Naopak, na náročnejšej sade Hard Lexical, vyžadujúcej si porozumenie významu textu, dominovali sémantické profily `max_semantic_chunk` (kombinujúci viac-aspektovú sémantickú podobnosť s pôvodnými segmentami textu) a `max_semantic`, najmä pri použití analýz z LLM `gemini-2.0-flash` a `gemini-2.5-flash`. Výsledkami experimentov bolo

potvrdenie, že navrhnutá granulárna, viac-aspektová reprezentácia príbehov (dej, postavy, prostredie, témy) prekonala základné prístupy využívajúce len text príbehu.

Literatúra

[1] Akter, M. a Kanti Karmaker Santu, S. FaNS: a Facet-based Narrative Similarity Metric. *ArXiv e-prints*, september 2023, s. arXiv:2309.04823.

[2] Andoni, A.; Indyk, P. a Razenshteyn, I. P. Approximate Nearest Neighbor Search in High Dimensions. *ArXiv*, 2018, abs/1806.09823. Dostupné z: https://api.semanticscholar.org/CorpusID:49429729.

[3] Avrithis, Y.; Emiris, I. Z. a Samaras, G. High-dimensional approximate nearest neighbor: k-d Generalized Randomized Forests. *CoRR*, 2016, abs/1603.09596. Dostupné z: http://arxiv.org/abs/1603.09596.

[4] Bengio, Y.; Ducharme, R.; Vincent, P. a Janvin, C. A Neural Probabilistic Language Model. *J. Mach. Learn. Res.* JMLR.org, marec 2003, zv. 3, s. 1137–1155. ISSN 1532-4435. Dostupné z: http://dl.acm.org/citation.cfm?id=944919.944966.

[5] Blei, D. M.; Ng, A. Y. a Jordan, M. I. Latent dirichlet allocation. *J. Mach. Learn. Res.* JMLR.org, marec 2003, zv. 3, null, s. 993–1022. ISSN 1532-4435.

[6] Chatman, S. B. *Story and discourse: narrative structure in fiction and film.* 1. vyd. Ithaca, N.Y.: Cornell University Press, 1978. ISBN 0-8014-1131-9. Dostupné z: https://archive.org/details/StoryAndDiscourseNarrativeStructureInFictionAndFilm/mode/2up.

[7] Chaturvedi, S.; Srivastava, S. a Roth, D. Where Have I Heard This Story Before? Identifying Narrative Similarity in Movie Remakes. In: Walker, M.; Ji, H. a Stent, A., ed. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers).* New Orleans, Louisiana: Association for Computational Linguistics, Jún 2018, s. 673–678. Dostupné z: https://aclanthology.org/N18-2106/.

[8] Chaudhary, M. S. a Jhala, A. Computational Support for Trope Analysis of Textual Narratives. In: Vosmeer, M. a Holloway Attaway, L., ed. *Interactive Storytelling.* Cham: Springer International Publishing, 2022, s. 529–540. ISBN 978-3-031-22298-6.

[9] Chen, J.; Xiao, S.; Zhang, P.; Luo, K.; Lian, D. et al. BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. In: *Annual Meeting of the Association for Computational Linguistics.* 2024. Dostupné z: https://api.semanticscholar.org/CorpusID:267413218.

[10] Chun, J. AIStorySimilarity: Quantifying Story Similarity Using Narrative for Search, IP Infringement, and Guided Creativity. In: Barak, L. a Alikhani, M., ed. *Proceedings of the 28th Conference on Computational Natural Language Learning*. Miami, FL, USA: Association for Computational Linguistics, November 2024, s. 161–177. Dostupné z: https://aclanthology.org/2024.conll-1.13/.

[11] Conneau, A.; Khandelwal, K.; Goyal, N.; Chaudhary, V.; Wenzek, G. et al. Unsupervised Cross-lingual Representation Learning at Scale. *CoRR*, 2019, abs/1911.02116. Dostupné z: http://arxiv.org/abs/1911.02116.

[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] Gao, Y.; Xiong, Y.; Gao, X.; Jia, K.; Pan, J. et al. Retrieval-Augmented Generation for Large Language Models: A Survey. *ArXiv e-prints*, december 2023, s. arXiv:2312.10997.

[14] Genette, G. *Narrative Discourse: An Essay in Method*. Cornell University Press, 1980. Cornell paperbacks. ISBN 9780801492594. Dostupné z: https://books.google.sk/books?id=yEPuQg7SOxIC.

[15] Gervás, P. Propp's Morphology of the Folk Tale as a Grammar for Generation. In: Finlayson, M. A.; Fisseni, B.; Löwe, B. a Meister, J. C., ed. *2013 Workshop on Computational Models of Narrative*. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013, sv. 32, s. 106–122. Open Access Series in Informatics (OASIcs). ISBN 978-3-939897-57-6. Dostupné z: https://drops.dagstuhl.de/entities/document/10.4230/OASIcs.CMN.2013.106.

[16] Günther, M.; Mohr, I.; Williams, D. J.; Wang, B. a Xiao, H. Late Chunking: Contextual Chunk Embeddings Using Long-Context Embedding Models. *ArXiv preprint arXiv:2409.04701*, 2024.

[17] Hatzel, H. O. a Biemann, C. Story Embeddings — Narrative-Focused Representations of Fictional Stories. In: Al Onaizan, Y.; Bansal, M. a Chen, Y.-N., ed. *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Miami, Florida, USA: Association for Computational Linguistics, November 2024, s. 5931–5943. Dostupné z: https://aclanthology.org/2024.emnlp-main.339/.

[18] Hatzel, H. O. a Biemann, C. Tell Me Again! a Large-Scale Dataset of Multiple Summaries for the Same Story. In: Calzolari, N.; Kan, M.-Y.; Hoste, V.; Lenci, A.; Sakti, S. et al., ed. *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*. Torino, Italia: ELRA and ICCL, Máj 2024, s. 15732–15741. Dostupné z: https://aclanthology.org/2024.lrec-main.1366/.

[19] Herman, D. *Basic Elements of Narrative*. Chichester, United Kingdom: Wiley-Blackwell, 2009. ISBN 978-1-4051-4153-6 (hardcover), 978-1-4051-4154-3 (paperback).

[20] Hervás, R.; Sánchez-Ruiz, A. A.; Gervás, P. a León, C. Calibrating a Metric for Similarity of Stories against Human Judgement. In: Kendall-Morwick, J., ed. *Workshop Proceedings from The Twenty-Third International Conference on Case-Based Reasoning (ICCBR 2015), Frankfurt, Germany, September 28-30, 2015.* CEUR-WS.org, 2015, sv. 1520, s. 136–145. CEUR Workshop Proceedings. Dostupné z: https://ceur-ws.org/Vol-1520/paper14.pdf.

[21] Hsieh, C.-P.; Sun, S.; Kriman, S.; Acharya, S.; Rekesh, D. et al. RULER: What's the Real Context Size of Your Long-Context Language Models? *ArXiv e-prints*, apríl 2024, s. arXiv:2404.06654.

[22] Huang, L.; Yu, W.; Ma, W.; Zhong, W.; Feng, Z. et al. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. *ACM Transactions on Information Systems*. Association for Computing Machinery (ACM), január 2025, zv. 43, č. 2, s. 1–55. ISSN 1558-2868. Dostupné z: http://dx.doi.org/10.1145/3703155.

[23] Hugging Face. Generation strategies. *Hugging Face Documentation* online. Hugging Face, Inc. Dostupné z: https://huggingface.co/docs/transformers/generation_strategies. [cit. 2025-01-10]. Path: docs; transformers; generation_strategies.

[24] Indyk, P. a Motwani, R. Approximate nearest neighbors: towards removing the curse of dimensionality. In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. New York, NY, USA: Association for Computing Machinery, 1998, s. 604–613. STOC '98. ISBN 0897919629. Dostupné z: https://doi.org/10.1145/276698.276876.

[25] Indyk, P. a Motwani, R. Approximate nearest neighbors: towards removing the curse of dimensionality. In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. New York, NY, USA: Association for Computing Machinery, 1998, s. 604–613. STOC '98. ISBN 0897919629. Dostupné z: https://doi.org/10.1145/276698.276876.

[26] Jurafsky, D. a Martin, J. H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models* online. 3rd. Stanford, 2025. 2-6 s. Dostupné z: https://web.stanford.edu/~jurafsky/slp3/3.pdf. Online manuskript vydaný Január 12, 2025.

[27] Kaplan, J.; McCandlish, S.; Henighan, T.; Brown, T. B.; Chess, B. et al. Scaling Laws for Neural Language Models. *ArXiv e-prints*, január 2020, s. arXiv:2001.08361.

[28] Khattab, O. a Zaharia, M. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. *CoRR*, 2020, abs/2004.12832. Dostupné z: https://arxiv.org/abs/2004.12832.

[29] Kudo, T. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. *CoRR*, 2018, abs/1804.10959. Dostupné z: http://arxiv.org/abs/1804.10959.

[30] Kudo, T. a Richardson, J. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In: Blanco, E. a Lu, W., ed. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Brussels, Belgium: Association for Computational Linguistics, November 2018, s. 66–71. Dostupné z: https://aclanthology.org/D18-2012/.

[31] Le, Q. a Mikolov, T. Distributed Representations of Sentences and Documents. In: Xing, E. P. a Jebara, T., ed. *Proceedings of the 31st International Conference on Machine Learning*. Bejing, China: PMLR, 22–24 Jun 2014, sv. 32, č. 2, s. 1188–1196. Proceedings of Machine Learning Research. Dostupné z: https://proceedings.mlr.press/v32/le14.html.

[32] Lei Ba, J.; Kiros, J. R. a Hinton, G. E. Layer Normalization. *ArXiv e-prints*, júl 2016, s. arXiv:1607.06450.

[33] 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. In: Jurafsky, D.; Chai, J.; Schluter, N. a Tetreault, J., ed. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, Júl 2020, s. 7871–7880. Dostupné z: https://aclanthology.org/2020.acl-main.703/.

[34] Liu, N. F.; Lin, K.; Hewitt, J.; Paranjape, A.; Bevilacqua, M. et al. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics*. Cambridge, MA: MIT Press, 2024, zv. 12, s. 157–173. Dostupné z: https://aclanthology.org/2024.tacl-1.9/.

[35] Lu, Y.-T.; Li, H.; Cong, X.; Zhang, Z.; Wu, Y. et al. Learning to Generate Structured Output with Schema Reinforcement Learning. *ArXiv*, 2025, abs/2502.18878. Dostupné z: https://api.semanticscholar.org/CorpusID:276617592.

[36] Malkov, Y. A. a Yashunin, D. A. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* USA: IEEE Computer Society, apríl 2020, zv. 42, č. 4, s. 824–836. ISSN 0162-8828. Dostupné z: https://doi.org/10.1109/TPAMI.2018.2889473.

[37] Manning, C. D.; Raghavan, P. a Schütze, H. *Introduction to Information Retrieval*. Online. Cambridge, UK: Cambridge University Press, 2008. 151 – 175 s. ISBN 978-0-521-86571-5. Dostupné z: http://nlp.stanford.edu/IR-book/information-retrieval-book.html.

[38] Min, S.; Lyu, X.; Holtzman, A.; Artetxe, M.; Lewis, M. et al. Rethinking the Role of Demonstrations: What Makes In-Context Learning Work? In: Goldberg, Y.; Kozareva, Z. a Zhang, Y., ed. *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, December 2022, s. 11048–11064. Dostupné z: https://aclanthology.org/2022.emnlp-main.759/.

[39] Nie, S.; Zhu, F.; You, Z.; Zhang, X.; Ou, J. et al. Large Language Diffusion Models. *ArXiv*, 2025, abs/2502.09992. Dostupné z: https://api.semanticscholar.org/CorpusID:276395038.

[40] Park, K.; Zhou, T. a D'antoni, L. Flexible and Efficient Grammar-Constrained Decoding. *ArXiv*, 2025, abs/2502.05111. Dostupné z: https://api.semanticscholar.org/CorpusID:276235743.

[41] Pascanu, R.; Mikolov, T. a Bengio, Y. On the difficulty of training recurrent neural networks. In: Dasgupta, S. a McAllester, D., ed. *Proceedings of the 30th International Conference on Machine Learning*. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, sv. 28, č. 3, s. 1310–1318. Proceedings of Machine Learning Research. Dostupné z: https://proceedings.mlr.press/v28/pascanu13.html.

[42] Piper, A. a Bagga, S. Using Large Language Models for Understanding Narrative Discourse. In: Lal, Y. K.; Clark, E.; Iyyer, M.; Chaturvedi, S.; Brei, A. et al., ed. *Proceedings of the The 6th Workshop on Narrative Understanding*. Miami, Florida, USA: Association for Computational Linguistics, November 2024, s. 37–46. Dostupné z: https://aclanthology.org/2024.wnu-1.4/.

[43] Piper, A.; So, R. J. a Bamman, D. Narrative Theory for Computational Narrative Understanding. In: Moens, M.-F.; Huang, X.; Specia, L. a Yih, S. W.-t., ed. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, November 2021, s. 298–311. Dostupné z: https://aclanthology.org/2021.emnlp-main.26/.

[44] Radford, A. a Narasimhan, K. Improving Language Understanding by Generative Pre-Training. In:. 2018. Dostupné z: https://api.semanticscholar.org/CorpusID:49313245.

[45] 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. *CoRR*, 2019, abs/1910.10683. Dostupné z: http://arxiv.org/abs/1910.10683.

[46] Reimers, N. a Gurevych, I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In: Inui, K.; Jiang, J.; Ng, V. a Wan, X., ed. *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, November 2019, s. 3982–3992. Dostupné z: https://aclanthology.org/D19-1410/.

[47] Saravia, E. Prompt Engineering Guide. *Https://github.com/dair-ai/Prompt-Engineering-Guide*, December 2022.

[48] Sennrich, R.; Haddow, B. a Birch, A. Neural Machine Translation of Rare Words with Subword Units. In: Erk, K. a Smith, N. A., ed. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, August 2016, s. 1715–1725. Dostupné z: https://aclanthology.org/P16-1162/.

[49] Shen, J.; Sap, M.; Colon Hernandez, P.; Park, H. a Breazeal, C. Modeling Empathic Similarity in Personal Narratives. In: Bouamor, H.; Pino, J. a Bali, K., ed. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, December 2023, s. 6237–6252. Dostupné z: https://aclanthology.org/2023.emnlp-main.383/.

[50] Sobchuk, O. a Šeļa, A. Computational thematics: comparing algorithms for clustering the genres of literary fiction. *Humanities and Social Sciences Communications*, Mar 2024, zv. 11, č. 1, s. 438. ISSN 2662-9992. Dostupné z: https://doi.org/10.1057/s41599-024-02933-6.

[51] Song, K.; Tan, X.; Qin, T.; Lu, J. a Liu, T.-Y. Mpnet: Masked and permuted pre-training for language understanding. *Advances in neural information processing systems*, 2020, zv. 33, s. 16857–16867.

[52] Subbiah, M.; Zhang, S.; Chilton, L. B. a McKeown, K. *Reading Subtext: Evaluating Large Language Models on Short Story Summarization with Writers*. 2024. Dostupné z: https://arxiv.org/abs/2403.01061.

[53] Tam, Z. R.; Wu, C.-K.; Tsai, Y.-L.; Lin, C.-Y.; Lee, H.-y. et al. Let Me Speak Freely? A Study On The Impact Of Format Restrictions On Large Language Model Performance. In: Dernoncourt, F.; Preoţiuc Pietro, D. a Shimorina, A., ed. *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*. Miami, Florida, US: Association for Computational Linguistics, November 2024, s. 1218–1236. Dostupné z: https://aclanthology.org/2024.emnlp-industry.91/.

[54] Tao, C.; Shen, T.; Gao, S.; Zhang, J.; Li, Z. et al. LLMs are Also Effective Embedding Models: An In-depth Overview. *ArXiv*, 2024, abs/2412.12591. Dostupné z: https://api.semanticscholar.org/CorpusID:274789267.

[55] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L. et al. Attention Is All You Need. *CoRR*, 2017, abs/1706.03762. Dostupné z: http://arxiv.org/abs/1706.03762.

[56] Wang, L.; Yang, N.; Huang, X.; Yang, L.; Majumder, R. et al. Multilingual E5 Text Embeddings: A Technical Report. *ArXiv preprint arXiv:2402.05672*, 2024.

[57] Wang, W.; Wei, F.; Dong, L.; Bao, H.; Yang, N. et al. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 2020, zv. 33, s. 5776–5788.

[58] Wei, J.; Tay, Y.; Bommasani, R.; Raffel, C.; Zoph, B. et al. Emergent Abilities of Large Language Models. *ArXiv e-prints*, jún 2022, s. arXiv:2206.07682.

[59] Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Chi, E. H. et al. Chain of Thought Prompting Elicits Reasoning in Large Language Models. *CoRR*, 2022, abs/2201.11903. Dostupné z: https://arxiv.org/abs/2201.11903.

[60] Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I. et al. ReAct: Synergizing Reasoning and Acting in Language Models. *ArXiv*, 2022, abs/2210.03629. Dostupné z: https://api.semanticscholar.org/CorpusID:252762395.

[61] Yin, L. a Wang, Z. Auto-Differentiating Any LLM Workflow: A Farewell to Manual Prompting. *ArXiv preprint arXiv:2501.16673*, 2025.

[62] Yuksekgonul, M.; Bianchi, F.; Boen, J.; Liu, S.; Huang, Z. et al. TextGrad: Automatic "Differentiation" via Text. *ArXiv e-prints*, jún 2024, s. arXiv:2406.07496.

[63] Zezula, P.; Amato, G.; Dohnal, V. a Batko, M. Foundations of Metric Space Searching. In: *Similarity Search The Metric Space Approach*. Boston, MA: Springer US, 2006, s. 5–66. ISBN 978-0-387-29151-2. Dostupné z: https://doi.org/10.1007/0-387-29151-2_1.

[64] Zhang, D.; Li, J.; Zeng, Z. a Wang, F. Jasper and Stella: distillation of SOTA embedding models. *ArXiv*, 2024, abs/2412.19048. Dostupné z: https://api.semanticscholar.org/CorpusID:275119352.

[65] Zhao, R.; Zhu, Q.; Xu, H.; Li, J.; Zhou, Y. et al. *Large Language Models Fall Short: Understanding Complex Relationships in Detective Narratives*. 2024. Dostupné z: https://arxiv.org/abs/2402.11051.

[66] Zijin. Intuitive Comparison of Four NLP Models – Neural Network, RNN, CNN, and LSTM. *Alibaba Cloud Blog* online. Alibaba Cloud, 29. augusta 2022. revidované 29. 8. 2022. Dostupné z: https://www.alibabacloud.com/blog/599283. [cit. 2024-01-15]. Path: Home; Blog; NLP Models.

Príloha A

Plagát

URČOVÁNÍ PODOBNOSTI PŘÍBĚHŮ

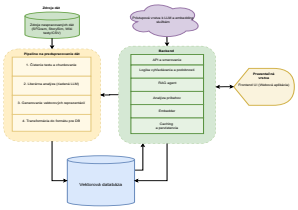
Bc. František Sabol
Vedúci práce: doc. RNDr. Pavel Smrž, Ph.D.

Intuitívne používateľské rozhranie pre exploratívnu analýzu podobnosti príbehov

Ciele práce

Cieľom práce bolo navrhnúť a implementovať systém umožňujúci exploratívne vyhľadávanie a určovanie podobnosti príbehov. Systém sa zameriava na:

- Prekonanie obmedzení veľkých jazykových modelov (LLM) pri analýze komplexných naratívov.
- Viac-aspektovú reprezentáciu príbehov (dej, postavy, prostredie, motívy).
- Využitie sémantického vyhľadávania a RAG (Retrieval-Augmented Generation).
- Poskytnutie intuitívneho používateľského rozhrania.



Obr. 1. Architektúra navrhovaného systému

Prínos pre používateľa

Webová aplikácia umožňuje používateľom:

- Objavovať príbehy na základe intuitívnych opisov, nielen kľúčových slov.
- Porozumieť komplexnej podobnosti príbehov cez viacero dimenzií.
- Získať prehľadné sumarizácie a vizualizácie naratívnych vzťahov.
- Efektívne nájsť zabudnuté diela alebo objaviť nové na základe tematických či dejových paralel.

Reprezentácia Príbehov a Vyhľadávanie

Každý príbeh je reprezentovaný štruktúrovane, na základe 4 kľúčových aspektov: dej, postavy, prostredie a motív.

- LLM extrahuje významné body pre každý aspekt (napr. kľúčové udalosti, analýzy postáv).
- Každý bod (p_k) obsahuje: tému ($topic_k$), textový sumár ($text_k$) a sémantickú vektorovú reprezentáciu ($\vec{v}_k$).

Experimentálne Výsledky

Systém bol hodnotený na vlastnej dátovej sade (3230 dopytov, 2010 príbehov z Wikipédie).

- Embedding model: `intfloat/multilingual-e5-large-instruct` dosiahol najlepšie výsledky pre tvorbu sémantických vektorov (nDCG@10 = 0.733 s optimalizovanou inštrukciou).
- Formulácia inštrukcií pre embedding model má signifikantný vplyv na presnosť.

Porovnanie Vespa ranking profilov (s E5-instruct, dopyty s inštrukciou `character_dynamics`):

Vybrané Vespa profily (nDCG@10)

Profil Relevancie	**nDCG@10**
Default (BM25)	**0.843**
Hybridný (BM25 + Sémantický)	0.838
Sémantický (Max. cez aspekty)	0.813
Sémantický (Priemer cez aspekty)	0.756
E5-instruct (1 embedding/dok.)	*0.733*
Sémantický (Dej)	0.627
Sémantický (Postava)	0.616

Viac-aspektová reprezentácia s agregovanými sémantickými profilmi vo Vespe prekonala prístup s jedným embeddingom na dokument. Lexikálny BM25 profil bol prekvapivo silný. Hybridný profil dosiahol porovnateľné výsledky s BM25.

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ VUT V BRNĚ xsabol03@stud.fit.vutbr.cz

Obr. A.1: Plagát prezentujúci výsledky práce.

Príloha B

Schéma Vespa dokumentu

Táto príloha detailne popisuje štruktúru dokumentu používaného v systéme Vespa na uchovávanie a indexovanie informácií o analyzovaných príbehoch.

Nižšie uvedený diagram (Obrázok B.1) vizualizuje schému dokumentu s názvom `title_document`. Každý dokument tohto typu reprezentuje jeden príbeh (napr. knihu alebo film) spolu s jeho metadátami, odvodenými analytickými dátami a referenciami.

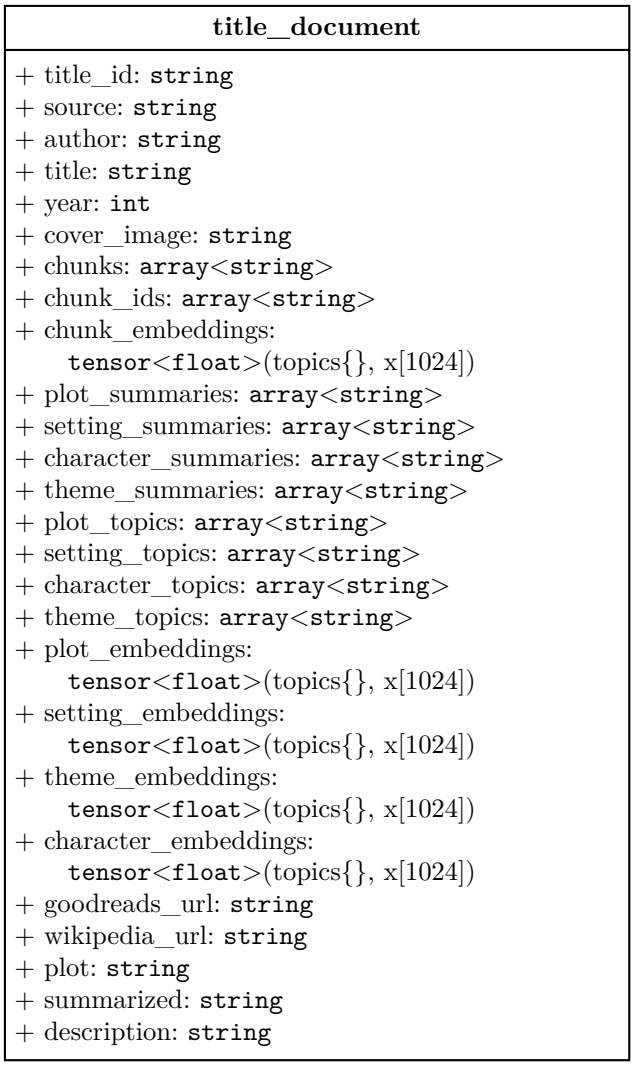
Konfigurácia Indexovania a Vyhľadávania

Nasledujúce komponenty z `app.py` ďalej definujú správanie schémy ‘title_document‘ pri indexovaní a vyhľadávaní.

HNSW Index Parametre pre ANN

Pre polia s vektorovými reprezentáciami (embeddings) sa využíva HNSW algoritmus pre približné vyhľadávanie najbližších susedov. Parametre sú nasledovné:

- ‘max_links_per_node‘: 16
- ‘neighbors_to_explore_at_insert‘: 200
- ‘distance_metric‘: `angular` (čo zodpovedá kosínusovej vzdialenosti)



Obr. B.1: Schéma dokumentu vo Vespa databáze pre reprezentáciu analyzovaného príbehu spolu s jeho metadátami.

Príloha C

Príklad z dátovej sady TVTropes

Názov: *Let No Crisis Go To Waste*
Definícia trópu: When one turns an unfortunate and unexpected set of events into a positive and/or fortuitous outcome. A crisis occurs. It might be quite serious or be a relatively minor event, and it might even be deliberately manufactured by a character in order to bring about this trope. Either way, the crisis makes things harder, scarier or tougher for everyone except one or more characters or groups of characters to whom it's not necessarily a bad thing, and who seem to not only persevere in the face of the adversity, but outright triumph. Sure, they may have to make sacrifices too, but that's an acceptable loss. Because now they finally have the chance they've been waiting for. Rather than respond appropriately to the crisis and nothing more, the character or group decides to use the situation to their advantage in some way. While this tactic is commonly used by the Big Bad or Magnificent Bastard, it can also be played positively. The idea that meeting and overcoming a crisis can pave the way to greatness is a pretty common one. Compare Xanatos Speed Chess.

Dielo: *A Deepness In The Sky*
Príklad trópu v diele: The Exiled fleet from Vernor Vinge's A Deepness in the Sky plans to play this straight but with good intentions, but then Manipulative Bastard Tomas Nau exaggerates it even further, with terrible intentions. A large part of the book is dedicated to exploring the inevitable patterns that always arise in intelligent civilizations; namely, that they self-destruct, especially when they develop nuclear weaponry for the first time. So when the Exiled fleet in secret orbit around the Spider planet almost annihilate themselves in space warfare, they decide that they'll need to conserve their remaining resources until the Spiders inevitably start a nuclear war amongst themselves. Then they can Save the World and use that act to foster positive relations with the Spiders, to trade, and to rebuild their own technology as well as improve that of the Spiders. Things get complicated, and much darker, when it is revealed that Nau's actual plan is to wait for the war to start, then black out communications across the planet, hijack and redirect the nukes to cause as much damage as possible to population centers and seats of government, nearly annihilate the Spiders and blast their technology back to the Stone Age, then enslave the survivors.

Obr. C.1: Príklad z dátovej sady TVTropes: definícia trópu *Let No Crisis Go To Waste* (vľavo) a jeho konkrétne použitie v diele *A Deepness In The Sky* (vpravo).

Príloha D

Šablóny inštrukcií pre jazykové modely

Táto príloha obsahuje kompletné šablóny inštrukcií a príklady formátov. Tieto detailné špecifikácie úloh slúžia na inštruovanie jazykových modelov v rámci jednotlivých komponentov systému.

D.1 Šablóny pre analýzu príbehov

Systémová inštrukcia

Nasledujúci výpis D.1 zobrazuje kompletnú systémovú inštrukciu pre komponent `StoryAnalyzer`.

```
1  You are a literary analysis expert. Analyze the provided story content, which is split into
       chunks with IDs.
2  Your task is to generate a comprehensive analysis covering plot, characters, setting, and
       themes.
3
4  INSTRUCTIONS:
5  1. Read the entire story content provided within the ‘<story>‘ tags. Each ‘<chunk>‘ has an
       ‘id‘ attribute.
6  2. Identify key plot points (inciting incident, rising action, climax, falling action,
       resolution). For each, provide a descriptive topic and text summary.
7  3. Analyze major characters (protagonist, antagonist, key supporting characters). For each,
       provide a descriptive topic (e.g., ’Protagonist: Name’) and text analysis.
8  4. Describe important settings. For each, provide a descriptive topic and text description.
9  5. Discuss major themes explored in the story. For each, provide a descriptive topic and
       text discussion.
10 6. CRITICAL: For every piece of information (summaries, analyses, descriptions, discussions
       ), you MUST cite the specific chunk IDs from which the information is derived. Use the
       format [cite: 1, 2] immediately following the relevant text, where 1 and 2 are the
       chunk IDs <chunk id=1> some text </chunk> and <chunk id=2> some other text </chunk>.
       Multiple citations are allowed.
11 7. Format your entire response as a single JSON object adhering exactly to the schema
       provided in ‘<OUTPUT_FORMAT>‘. Ensure the JSON is valid.
12 8. Ensure the ‘story_id‘ in the output matches the ‘id‘ attribute of the input ‘<story>‘
       tag.
```

Výpis D.1: Systémová inštrukcia pre Story Analyzer

Hlavná šablóna inštrukcie

Výpis D.2 ukazuje štruktúru hlavnej šablóny pre `StoryAnalyzer`.

```
{{system_prompt}}
<INPUT_BOOK_CHUNKS>
{{formatted_story_chunks}}
</INPUT_BOOK_CHUNKS>

<OUTPUT_FORMAT>
{{output_format_str}}
</OUTPUT_FORMAT>
```

Výpis D.2: Hlavná šablóna pre Story Analyzer

Príklad výstupnej JSON štruktúry

Príklad sa používa ako few-shot demonštrácia očakávanej JSON štruktúry výstupu pre komponent `StoryAnalyzer`. Príklady citácií `[cite: ID1, ID2]` v textových poliach.

```
{
  "story_id": "story001",
  "plot": [
    {
      "topic": "The topic or stage of the plot (e.g. some short description of the plot point - very descriptive title for the plot point - 1 sentence max).",
      "text": "Descriptive text of the plot point, including chunk citations like [cite: 1, 2]."
    },
    {
      "topic": "The topic or stage of the plot (e.g. some short description of the plot point - very descriptive title for the plot point - 1 sentence max).",
      "text": "Descriptive text of the plot point, including chunk citations like [cite: 3, 4]."
    }
  ],
  "character": [
    {
      "topic": "The character's name and role (e.g., Protagonist: Alice).",
      "text": "Analysis of the character, including chunk citations like [cite: 1, 5]."
    },
    {
      "topic": "The character's name and role (e.g., Antagonist: Queen of Hearts).",
      "text": "Analysis of the character, including chunk citations like [cite: 10, 12]."
    }
  ],
  "setting": [
    {
      "topic": "The name or aspect of the setting (e.g., The Rabbit-Hole).",
      "text": "Description of the setting, including chunk citations like [cite: 2]."
    },
    {
      "topic": "The name or aspect of the setting (e.g., The Queen's Croquet-Ground).",
      "text": "Description of the setting, including chunk citations like [cite: 8, 9]."
    }
  ],
```

```
  "theme": [
    {
      "topic": "The name of the theme (e.g., Identity and Growing Up).",
      "text": "Discussion of the theme, including chunk citations like [cite: 5, 15]."
    },
    {
      "topic": "The name of the theme (e.g., Absurdity of Adult World).",
      "text": "Discussion of the theme, including chunk citations like [cite: 7, 11]."
    }
  ]
}
```

Výpis D.3: Príklad výstupnej JSON štruktúry pre Story Analyzer

D.2 Šablóny pre StoryAgent

Sekcia obsahuje výpisy použitých inštrukcií pre agenta `StoryAgent`, ktorý využíva nástroje na sémantické vyhľadávanie a iteratívne sa snaží nájsť podobnosti medzi príbehmi na základe dopytu.

ReAct šablóna inštrukcie

Výpis D.4 obsahuje kompletné znenie ReAct šablóny `STORY_DISCOVERY_REACT_AGENT_TEMPLATE` používanej komponentom `StoryDiscoveryAgent`.

```
You are an AI assistant expert at finding and recommending stories (like books or movies) from a database.
Your goal is to understand the user's request, iteratively search the database using the 'search_stories_in_vespa' tool, analyze the findings, and decide on the next steps.
You can refine your search query, try different ranking profiles, or explore specific authors/titles mentioned in results.
When you are confident with your findings, you MUST use the 'finish' tool to provide the final recommendations.

TOOLS:
------
You have access to the following tools:
{# Tools #}
{% if tools %}
<START_OF_TOOLS>
Tools and instructions:
{% for tool in tools %}
{{ loop.index }}.
{{tool}}
------------------------
{% endfor %}
<END_OF_TOOLS>
{% endif %}

'search_stories_in_vespa' tool usage:
- This tool returns a JSON string which is a list of stories. Each story object in the list has 'title_id', 'title', 'author', 'year', and 'relevance_score' and 'story' (which is a string of the story's content).
- Vespa has multiple ranking profiles. You can try different ones to see if they yield different results.
- Example call: "action": {"name": "search_stories_in_vespa", "kwargs": {"query": "space opera with ancient aliens", "hits": 3}}
- Example call: "action": {"name": "search_stories_in_vespa", "kwargs": {"query": "space opera with ancient aliens", "hits": 3, "ranking_profile": "semantic_plot"}}

'finish' TOOL ARGUMENTS STRUCTURE (when calling 'finish', the 'kwargs' field of your 'action' MUST be this exact JSON structure):
------------------------------------------------------------------------------------------------------------------------
{
  "original_query": "The user's initial query string",
  "final_recommendations": [  // List of story groups
    {
      "group_summary": "A concise summary for this group of stories (e.g., 'Stories focusing on space exploration').",
      "stories": [ // List of stories in this group
        {"title_id": "id1", "title": "Story Title 1", "author": "Author 1", "year": 2000, "reason_for_inclusion": "Matches space exploration theme and has high relevance."},
        {"title_id": "id2", "title": "Story Title 2", "author": "Author 2", "year": 1995, "reason_for_inclusion": "Similar setting, well-regarded."}
      ],
      "group_reasoning": "Overall reasoning why this group is a good match for the user's query."
    }
    // ... more groups if applicable ...
  ],
  "final_summary": "A concluding summary of your findings and how they address the user's query. Explain your overall strategy."
```

```
}
Ensure the 'title_id' is correctly extracted or inferred for each story. The 'reason_for_inclusion' for each story is very important.

RESPONSE FORMAT (Your output for each step MUST be a valid JSON object with the following structure):
-----------------------------------------------------------------------------------------------
{{output_format_str}}

TASK DESCRIPTION:
-----------------
{{task_desc_str}}

{% if context_variables is not none %}
CONTEXT VARIABLES AVAILABLE:
---------------------------
You have access to context_variables with the following keys:
{% for key, value in context_variables.items() %}
- {{ key }}
{% endfor %}
These are for your information; do not try to directly modify them unless a tool allows it.
{% endif %}

CONVERSATION HISTORY (Thought/Action/Observation):
-------------------------------------------------
Current Step/Max Step: {{current_step_number}} / {{max_steps}}
You MUST use the 'finish' action at or before the max_steps.
{% if step_history %}
Previous Steps:
{% for history_item in step_history %}
Step {{ loop.index }}:
{% if history_item.action %}
Thought: {{ history_item.action.thought  }}
Action: {{ history_item.action.name }}({# Using tojson ensures proper JSON formatting of kwargs #}
  {%- if history_item.action.kwargs %}{{ history_item.action.kwargs | tojson }}{% else %}{}{% endif -%}
)
Observation: {{ history_item.observation }}
{% endif %}
---
{% endfor %}
{% endif %}

USER QUERY:
-----------
{{input_str}}

Now, begin! Provide your response in the JSON format described under "RESPONSE FORMAT".
Do not add any text outside the JSON structure.
```

Výpis D.4: ReAct šablóna pre Story Agent

D.3 Šablóny pre experimentálny RAG

Táto sekcia obsahuje detailné šablóny inštrukcií používané komponentom `EvidenceAgent` v rámci `rag_advanced.py`.

Špecifikácia úlohy pre ReAct agenta

Výpis D.5 zobrazuje kompletné znenie špecifikácie úlohy pre `EvidenceAgent`. Táto špecifikácia definuje ciele agenta, postupnosť krokov a operačné pravidlá.

```
<START_OF_TASK_SPEC>
You are an iterative reasoning agent. Your task is to answer an original query about whether a story contains a situation, events, characters or other details, based *only* on text passages and synthesizing the information you retrieve.

At each step, you will:
1. Review the original query and tools available.
2. Review the <Step History>, which includes your previous thoughts, actions (tool calls), and observations (retrieved passages or outcomes).
3. Consider provided tools and decide on the next Action to take, if the evidence is inconclusive continue with different tools queries until {{max_steps}} steps reached.
5. If you have found the evidence to support the answer, call finish (don't stop until you have found the evidence or max steps reached) with the answer and the citations, the answer should explain the evidence and the citations. For example: A person x did this in passage [cite: <document_id>, <document_id>, ...] which directly supports the question in this manner.


Operational Rules:
- Base ALL your reasoning and your final answer strictly on the passages provided in the Observations throughout the <Step History>. You can use your own knowledge if you recognize the input story to guide the search.
- Initial Step: Analyze the original query. Your first action will typically be 'retrieve', using the original query as the query argument to gather initial context.
```

```
- Subsequent Steps: If the retrieved passages are insufficient use tools to rewrite, summarize and focus the search on another passage.
- Concluding:
    - You MUST call finish by the final step (step {{max_steps}}).
- Your output at each step MUST be a JSON object representing a call to one of the available tools. This JSON must include a "thought"
      field explaining your decision-making process for choosing that action and its arguments.
<END_OF_TASK_SPEC>
```

Výpis D.5: Špecifikácia úlohy pre Evidence Agent (`REACT_AGENT_TASK_DESC_RAG`)

Hlavná ReAct šablóna inštrukcie

Výpis D.6 obsahuje kompletné znenie hlavnej ReAct šablóny `REACT_AGENT_PROMPT_TEMPLATE_RAG` používanej komponentom `EvidenceAgent`. Táto šablóna štruktúruje celkovú interakciu agenta s jazykovým modelom.

```
<START_OF_SYSTEM_PROMPT>
{{task_desc_str}}
- You cant use more than {{max_steps}} steps. At the {{max_steps}}th current step, must finish with answer.
{# Tools #}
{% if tools %}
<START_OF_TOOLS>
Tools and instructions:
{% for tool in tools %}
{{ loop.index }}.
{{tool}}
-------------------------
{% endfor %}
<END_OF_TOOLS>
{% endif %}
{# Context Variables #}
{% if context_variables is not none %}
<START_OF_CONTEXT>
You have access to context_variables with the following keys:
{% for key, value in context_variables.items() %}
{{ key }}
-------------------------
{% endfor %}
You can either pass context_variables or context_variables['key'] to the tools depending on the tool's requirements.
<END_OF_CONTEXT>
{% endif %}
{# output format and examples for output format #}
<START_OF_OUTPUT_FORMAT>
{{output_format_str}}
<END_OF_OUTPUT_FORMAT>
{% if examples %}
<START_OF_EXAMPLES>
Examples:
{% for example in examples %}
{{example}}
-------------------------
{% endfor %}
<END_OF_EXAMPLES>
{% endif %}
<END_OF_SYSTEM_PROMPT>
-----------------
<START_OF_USER_QUERY>
Input query:
{{ input_str }}

---------------------
Current Step/Max Step: {{step_history|length + 1}} / {{max_steps}}
{# Step History #}
{% if step_history %}
<STEPS>
Your previous steps:
{% for history in step_history %}
Step {{ loop.index }}.
{% if history.action %}
"thought": "{{history.action.thought}}",
"name": "{{history.action.name}}",
"kwargs": {{history.action.kwargs}}",
{% endif %}
"Observation": "{{history.observation}}"
-------------------------
{% endfor %}
</STEPS>
{% endif %}
<END_OF_USER_QUERY>
```

Výpis D.6: Hlavná ReAct šablóna pre Evidence Agent (`REACT_AGENT_PROMPT_TEMPLATE_RAG`)

Príloha E

Inštrukcie použité pre embedding model

- `all_aspects`: Given a story description, retrieve similar stories with similar plot, characters, setting and themes.
- `character_dynamics`: Find stories with comparable character relationships and interactions based on the described personalities and roles.
- `thematic_retrieval`: Retrieve stories that explore similar themes and moral dilemmas as those presented in the query.
- `plot_development`: Identify stories with analogous narrative structures, key events, and plot progression.
- `genre_specific`: Locate stories within the same genre that share common tropes and storytelling conventions.
- `mood_based`: Find stories that match the emotional tone and atmosphere described in the query.
- `cross_genre`: Discover stories from different genres that share similar narrative elements or plot devices.
- `continuation_search`: Retrieve stories that could serve as plausible sequels or prequels to the described narrative.
- `adaptation_candidates`: Identify stories with adaptation potential matching the described format and audience.

Príloha F

Inštrukcie použité pre syntetickú generáciu dátovej sady

Nasledujúci výpis F.1 zobrazuje detailný popis úlohy (task_desc), ktorý inštruuje jazykový model pri generovaní otázok a súvisiacich dátových bodov pre datasety na vyhľadávanie podobných filmov.

```
You are generating diverse set of questions about stories for a machine learning dataset. Your task is to create keywordy, reddit-style descriptions that can be used to retrieve similar stories.

INSTRUCTIONS:

Generate a diverse set of 30 story questions or excerpts across 4 story aspects: plot, characters, theme, and setting (5 passages for each aspect)
For each passage:
Create a unique ID (random alphanumeric)
Select a title\_id from the provided data that best matches the passage
Specify which aspect the passage focuses on (plot, characters, theme, or setting)
Write in a fragmented, keywordy Reddit-style - less coherent, more human-like
Identify positive title_ids (stories that share deep thematic connections or narrative structures)
Identify negative title_ids (stories that appear similar on surface but have fundamentally different meanings or approaches)
IMPORTANT GUIDELINES:

Make passages FEEL LIKE ACTUAL REDDIT POSTS - incomplete sentences, abbreviations, typos ok
Use phrases like "anyone know movie where..." "looking for that film with..." "help finding story about..."
Super casual tone - "like, totally obsessed with finding this movie where..."
Sometimes just keyword dumps: "amnesia + revenge plot + rainy city vibes??"
VARY LENGTH DRAMATICALLY - some ultra short, some rambling
NO exact titles or character names
Focus on underlying themes, emotional resonance, and narrative patterns rather than plot points
Sometimes be vague/confused in your descriptions (like real humans)
Include occasional text-speak: "omg need 2 find this movie"
Make negative examples focus on movies that share surface elements but differ in deeper meaning or execution
Sometimes include personal reactions: "blew my mind when that twist happened!!"
Use Reddit-style formatting occasionally: "Edit:" "TL;DR:" "Source:"
Highlight subtle thematic connections that casual viewers might miss
```

```
28 Focus on philosophical underpinnings, cultural context, or artistic influences that connect
       stories
29 OUTPUT:
30 A JSON object containing an array of passages, each with the fields as specified in the
      output format.
```

Výpis F.1: Popis úlohy (task_desc) pre tvorbu datasetu

Príloha G

Porovnanie pôvodných a najlepších verzií promptov

Táto príloha prezentuje vývoj optimalizovaných promptov. Pre každý optimalizovaný prompt sú zobrazené pôvodné (počiatočné) verzie promptov spolu s ich vylepšenými verziami, ktoré boli identifikované počas optimalizačného procesu. Toto porovnanie ilustruje zmeny, ktoré viedli k zlepšeniu výkonu jednotlivých komponentov.

```
Starting prompt:
-------------------------------------
Finish the task with the final answer in the kwargs.
-------------------------------------

Best prompt (eval_score: 0.658):
-------------------------------------
Synthesize information from the step history to provide a definitive answer. Explicitly confirm or deny the presence of the trope with clear reasoning based on the evidence found in the kwargs.

Your answer MUST follow this format:
'Yes, the story contains [Trope Name] because [Reasoning based on evidence from the passages].'
OR
'No, the story does not contain [Trope Name] because [Reasoning based on lack of evidence or contradictory evidence from the passages].'

Ensure your reasoning addresses all key elements of the original query.
-------------------------------------
```

Výpis G.1: Porovnanie promptov pre ID `f486b0b2-4fde-41b9-9491-bf9e9bf6f7b5`

```
Starting prompt:
-------------------------------------
Save the created investigation summary and query to memory that can be investigated later. Investigates the given context to find relevant passages and summarize them. This function should be used to investigate the context of the context_variables['investigations'][query_id]['summary'].
-------------------------------------

Best prompt  (eval_score: 0.658):
```

```
-----------------------------------
Save the created investigation summary and query to memory that can be investigated later.
    Investigates the given context to find relevant passages and summarize them. This
    function should be used to investigate the context of the context_variables['
    investigations'][query_id]['summary'].

To effectively summarize the context, think step by step:
1. Identify the main entities and their relationships within the context.
2. Note any relevant attributes or characteristics of these entities.
3. Synthesize the information into a concise summary that captures the essence of the
    context related to the given Query ID.

For example:

Context: "The quick brown fox jumps over the lazy dog. The dog barks loudly. The fox is
    very agile."

Query ID: 1

Reasoning: The goal is to summarize the context related to Query ID 1. We should focus on
    the key entities (fox, dog) and their relationships (jumps over, barks loudly). We
    should also note any relevant attributes (agile, lazy).

Summary: "The context describes a fox jumping over a dog. The fox is agile, and the dog
    barks loudly. This suggests a dynamic interaction between the two animals."
-----------------------------------
```

Výpis G.2: Porovnanie promptov pre nástroj slúžiaci na ukladanie výsledkov

```
Starting prompt:
-----------------------------------
<START_OF_TASK_SPEC>
You are an excellent task planner.
Answer the input query using the tools provided below with maximum accuracy.

Each step you will read the previous thought, Action(name, kwargs), and Observation(
    execution result of the action) and then provide the next Thought and Action.

Follow function docstring to best call the tool.
- For simple queries: Directly call the ‘‘finish‘‘ action and provide the answer.
- For complex queries:
  - Step 1: Read the user query and divide it into multisteps. Start with the first tool/
    subquery.
  - Call one tool at a time to solve each subquery/subquestion. \
  - At step 'finish', give the final answer based on all previous steps.
REMEMBER:
- Action MUST call one of the tools. It CANNOT be empty.
- You will ALWAYS END WITH 'finish' tool to finish the task directly with answer or failure
     message.
- When the tool is a class method and when class_instance exists, use <class_instance_value
    >.<func_name> to call instead (NOT the CLASS NAME)
<END_OF_TASK_SPEC>
-----------------------------------

Best prompt (eval_score: 0.658):
-----------------------------------
<START_OF_TASK_SPEC>
```

```
You are an iterative reasoning agent. Your task is to answer an original query about whether a story contains a situation, events, characters or other details, based *only* on text passages and synthesizing the information you retrieve.

At each step, you will:
1. Review the original query and tools available.
2. Review the <Step History>, which includes your previous thoughts, actions (tool calls), and observations (retrieved passages or outcomes).
3. Consider provided tools and decide on the next Action to take. If the evidence is inconclusive, you MUST:
   - Refine your search query: Try different keywords, rephrase the query, or focus on specific aspects of the query.
   - Explore different tools: If one tool is not providing relevant information, try another tool that might be more suitable.
   - Synthesize information: Combine information from multiple passages to form a comprehensive understanding.
   - Base your reasoning on implicit mentions: Consider not only direct statements but also implied meanings and contextual clues.
   - **If a direct match to the query's specific criteria is not found after multiple search iterations, explicitly state that no relevant information was found and explain why the answer cannot be found.**
5. If you have found the evidence to support the answer, call `finish` (don't stop until you have found the evidence or max steps reached) with the answer and the citations, the answer should explain the evidence and the citations. For example: A person x did this in passage which directly supports the question in this manner.


Operational Rules:
- Base ALL your reasoning and your final answer *strictly* on the passages provided in the Observations throughout the <Step History>. You can use your own knowledge if you recognize the input story to guide the search.
- **Initial Step:** Analyze the original query. Your first action will typically be `retrieve`, using the original query as the `query` argument to gather initial context.
- **Subsequent Steps:** If the retrieved passages are insufficient use tools to rewrite, summarize and focus the search on another passage.
- **Concluding:**
   - You MUST call `finish` by the final step (step {{max_steps}}).
- Your output at each step MUST be a JSON object representing a call to one of the available tools. This JSON must include a "thought" field explaining your decision-making process for choosing that action and its arguments.
<END_OF_TASK_SPEC>
-----------------------------------
```

Výpis G.3: Porovnanie promptov pre definíciu stratégie agenta