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

GENEROVÁNÍ DOKUMENTACE KE ZDROJOVÉMU KÓDU V JAZYCE PYTHON

GENERATING DOCUMENTATION TO SOURCE CODE IN PYTHON

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JURAJ NOVOSÁD**
AUTHOR

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

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

154278

Ústav: Ústav počítačové grafiky a multimédií (UPGM)
Student: **Novosád Juraj**
Program: Informační technologie
Název: **Generování dokumentace ke zdrojovému kódu v jazyce Python**
Kategorie: Umělá inteligence
Akademický rok: 2023/24

Zadání:

1. Seznamte se s volně dostupnými neuronovými modely, vhodnými pro generování dokumentovaného kódu, a styly dokumentace zdrojového kódu v jazyce Python, používanými v rámci populárních projektů.
2. Shromážděte a zpracujte komentované kódy tak, aby bylo možné data využít pro adaptaci jazykových modelů.
3. Vyberte několik existujících modelů a vyzkoušejte jejich adaptaci na připravených datech.
4. Vyhodnoťte kvalitu adaptovaných modelů a porovnejte výsledky s dostupnými konkurenčními řešeními.
5. Navrhněte systém, který by umožnil užívateli používat adaptované modely na běžně dostupném hardware.
6. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- dle doporučení vedoucího

Při obhajobě semestrální části projektu je požadováno:

- funkční prototyp řešení

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

Vedoucí práce: **Smrž Pavel, doc. RNDr., Ph.D.**
Vedoucí ústavu: Černocký Jan, prof. Dr. Ing.
Datum zadání: 1.11.2023
Termín pro odevzdání: 9.5.2024
Datum schválení: 22.4.2024

Abstrakt

Cieľom práce je adaptovať vybrané jazykové modely na doménových dátach a vytvoriť systém, ktorý by umožnil ich použitie na bežne dostupnom hardware. Modely boli adaptované pre generovanie dokumentácie k nedokumentovanému zdrojovému kódu v programovacom jazyku Python, tak aby dodržiavali konvenciu Google Style. Prerekvizita adaptovania modelu bola získať doménové dáta a vhodne ich spracovať pre účely fine-tuningu modelu. Táto práca sa zameriava na fine-tuning modelov s ppočtom parametrov menej ako jedna miliarda, z dôvodu umožnenia inferencie aj na bežne dostupnom hardware. Časťou práce bolo objektívne zhodnotiť kvalitu adaptovaných modelov. Z tohto dôvodu som vyvinul nástroj, ktorý na vybranom korpuse ohodnotí kvalitu generovanej dokumentácie na vybraných modeloch. Vyhodnotenie adaptovaných modelov ukázalo, že dosahujú porovnateľný výkon ako násobne väčšie modely trénované pre všeobecné úlohy, napríklad gpt-3.5-turbo-0125. Výsledkom práce je server, schopný horizontálneho škálovania, ktorý integruje možnosti nielen adaptovaných modelov cez ľahko použiteľné API.

Abstract

The aim of this work is to adapt selected language models on domain data and to develop a system that would allow their use on commonly available hardware. The models have been adapted to generate documentation for undocumented source code in the Python programming language to follow the Google Style convention. A prerequisite of model adaptation was to obtain domain data and process it appropriately for the purpose of model fine-tuning. This work focuses on fine-tuning models with fewer than one billion parameters, for the sake of enabling inference even on commonly available hardware. Part of the work was to objectively evaluate the quality of the adapted models. For this reason, I developed a tool that evaluates the quality of the generated documentation on a selected corpus of models. The evaluation of the adapted models showed that they achieve comparable performance to multiply larger models for general tasks, such as gpt-3.5-turbo-0125. The result of this work is a server capable of horizontal scaling that integrates the capabilities of more than just the adapted models through an easy-to-use API.

Klíčová slova

veľké jazykové modely, generovanie textu, sequence 2 sequence, transformers, HTTP, fastapi, huggingface, Python, BLEU, Rouge, Meteor, sentence-transformers

Keywords

Large language models, text generation, sequence 2 sequence, transformers, HTTP, fastapi, huggingface, Python, BLEU, Rouge, Meteor, sentence-transformers

Citace

NOVOSÁD, Juraj. *Generování dokumentace ke zdrojovému kódu v jazyce Python.* Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. RNDr. Pavel Smrž, Ph.D.

Generování dokumentace ke zdrojovému kódu v jazyce Python

Prohlášení

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

.....
Juraj Novosád
8. května 2024

Poděkování

Chcel by som sa poďakovať vedúcemu mojej bakalárskej práce doc. RNDr. Pavlovi Smržovi Ph.D. za jeho odborné vedenie, ochotu a cenné rady pri vypracovávaní tejto práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Vďaka rapídnemu rozvoju strojového učenia a umelej inteligencie sa do popredia dostali veľké jazykové modely, ktoré zmenili náš prístup k mnohým úlohám spracovania prirodzeného jazyka. Medzi tieto úlohy spadá aj generovanie programovej dokumentácie.

Komplexná, dobre štruktúrovaná dokumentácia má pri programovaní kľúčový význam. Pomáha nielen pochopiť, čo daný kus kódu robí, ale slúži aj ako vodítko pre vývojárov pri úpravách a správe ich projektov. Vytvorenie takejto dokumentácie je však často prácna a časovo náročná úloha, ktorá presúva pozornosť vývojárov od samotného programovania.

V dnešnej dobe obrovského množstva digitálnych informácií organizácie vytvárajú kolosálne softwarové projekty, ktorým často chýba dostatočná dokumentácia. Manuálne úsilie zdokumentovať takéto rozsiahle softwarové projekty je neuveriteľne vyčerpávajúce, ak nie neuskutočniteľné. Automatizovaný dokumentačný nástroj schopný lokálneho spustenia ponúka nenahraditeľné riešenie, ktoré šetrí drahocenné vývojárske úsilie a čas.

Využitím zdatnosti veľkých jazykových modelov môžeme generovať dokumentáciu pre úplne nedokumentovaný alebo nedostatočne zdokumentovaný zdrojový kód, najmä v populárnych programovacích jazykoch, ako je Python. Spoločnosti ako OpenAI ponúkajú modely, ktoré sú schopné poskytovať pôsobivé výsledky. Tieto modely sú však zvyčajne obrovské, vyžadujú vysoký výpočtový výkon a často sú použiteľné len prostredníctvom API, čo vedie k problémom so závislosťou, nákladmi a praktickosťou. Pre široké a pravidelné používanie majú modely schopné lokálneho a efektívneho behu na bežnom hardvéri skutočne výhodu.

Cieľom tejto práce je vyplniť túto medzeru – absenciu jazykových modelov, ktoré dokáže prevádzkovať lokálne dostupný hardware, na efektívne generovanie dokumentácie. Modely boli adaptované a vyladené tak, aby vyhovovali jazyku Python, pričom dodržujú všeobecne uznávanú konvenciu Google Style. Dôraz bol kladený na modely s menej ako miliardou parametrov, čo zabezpečuje, že sú dobre prispôsobené na lokálne nasadenie aj na bežnom hardvéri.

Kapitola 2 popisuje technológie použité na adaptáciu, dôraz kladie na popis architektúry a použitých modelov. Kapitola 3 je zameraná na proces prípravy dát a samotný fine-tuning. Kapitola 4 popisuje spôsob hodnotenia kvality generovanej dokumentácie. Venuje sa teoretickému popisu použitých metrík, ich integrácii v riešení a vývoju nástroja, ktorý na zvolenom datasete zhodnotí schopnosť modelu generovať dokumentáciu. Koniec kapitoly je venovaný ohodnoteniu adaptovaných modelov a ich porovnaniu s konkurenčnými riešeniami. Kapitola 5 je venovaná implementácii servera integrujúceho nielen adaptované modely. Predstavuje pružnú architektúru modelu, ktorá môže implementovať rôzne modely neurónových sietí. V kapitole je konceptuálne predstavená architektúra servera, ako spracováva požiadavky a ako dokáže efektívne využívať hardware.

Kapitola 2

Strojové spracovanie jazyka

V informatike a lingvistike prechádza oblasť spracovania prirodzeného jazyka rýchlym vývojom hlavne v posledných rokoch. Spracovanie prirodzeného jazyka primárne zahŕňa interakciu medzi počítačom a človekom. Konečným cieľom je bezproblémová interakcia medzi človekom a počítačom bez tradičných programovacích jazykov, ktorá vytvára pôdu pre strojové spracovanie údajov získaných zo zdrojov v prirodzenom jazyku.

V oblasti programovania patrí spracovanie jazyka k technike analýzy syntaxe a sémantiky programovacieho jazyka. Vo svojom počiatku bolo spracovanie prirodzeného jazyka založené na popisovaní daného jazyka pomocou pravidiel, ktorých aplikovaním na text bol dosiahnutý cieľ. Programátori venovali veľa času a úsilia vytváraniu jednoznačných pravidiel, ktoré inštruujú počítač ako interpretovať neštruktúrované dáta. Tieto inštrukcie, pravidlá, museli byť veľmi rozsiahle a zahrnovať rôzne vetné konštrukcie. Aj napriek úsiliu nebolo možné spracovať vstup mimo známe pravidlá.

Zmeny prišli s nástupom strojového učenia, ktoré prinieslo viac flexibility do spracovávania jazyka. Algoritmy, ktoré sa dokázali učiť z dát, redukovali nutnosť explicitného programovania. Avšak fáza učenia bola do veľkej miery závislá od návrhu funkcií, ktoré extrahujú relevantné vlastnosti dát. Relevantné vlastnosti, znaky, vstupných dát sa v obore strojového učenia nazývajú `features`. Teda napríklad v spracovaní prirodzeného jazyka sa zo vstupného textu extrahujú features. Čo sa ukázalo ako zložitý problém, pretože vyjadrovacie schopnosti prirodzených jazykov ďaleko presahujú možnosti programovacích jazykov,

Hlboké neurónové siete predstavujú ďalší krok vo svete nielen spracovania prirodzeného jazyka. Prinášajú oveľa silnejší výpočtový aparát, ktorý okrem iného umožňuje automatickú extrakciu featur zo vstupu. Modely neurónových sietí, hlavne `Recurent Neural Networks` a neskôr `Long Short-Term Memory` a `Gated Recurrent Units`, posunuli spracovanie sekvenčných dát, ich schopnosť identifikovať vzory v dátach a generalizovať.

Tieto techniky sú základ pre oveľa silnejší koncept: architektúra modelov typu Transformer. Táto architektúra bude predstavená bližšie v nasledujúcej časti.

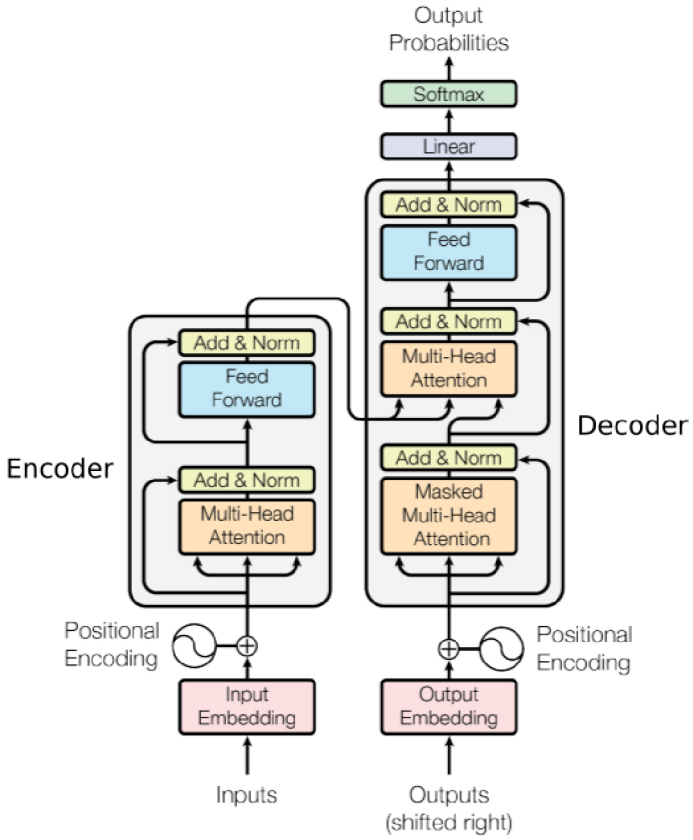
2.1 Jazykové modely postavené na architektúre Transformers

Revolúciu vo svete neurónových sietí spôsobilo uvedenie novej architektúry, nazývanej Transformers, predstavená bola v článku **Attention is all you need** [15]. Novo predstavené princípy a možnosti zohrali kritickú rolu v evolúcii výpočtových modelov na porozumenie prirodzeného jazyka.

Typický model Transformera pozostáva z dvoch hlavných častí: **Enkodér** a **Dekodér**, ktoré sú zostavené z niekoľkých vrstiev. Srdce týchto častí je postavené na mechanizme **Self-Attention**, ktorý významne prispieva k schopnosti modelu porozumieť významu slov v kontexte.

Prečo je mechanizmus Self-Attention taký dôležitý? Tento mechanizmus zachytí vzťahy a závislosti vo vstupných sekvenciách. Inak povedané, upraví každú časť vstupu podľa kontextu, v ktorom sa nachádza. Napríklad ak si zoberieme slovo strom, jeho význam môže byť acyklický graf, drevnatá rastlina alebo iné. Self-attention podľa kontextu upraví význam vektorovej reprezentácie vstupu, a tým dedukuje, o aký druh stromu ide. Tento mechanizmus sa vykonáva paralelne vo viacerých inštanciách v časti **Multi-Headed Attention**.

Na Obrázku 2.1 je architektúra modelu Transformer. Postupne je ďalej popísané ako principiálne fungujú jednotlivé časti.



Obrázek 2.1: Architektúra jazykového modelu typu Transformers [15].

Pri spracovaní textu Transformer najprv konvertuje vstupné tokeny, časti slov reprezentované vektormi, na embeddingy. **Embedding** je reprezentácia vstupu, ktorá zachytáva sémantiku. Embeddingy sa sčítajú s pozičným enkódovaním, na obrázku 2.1 **Positional Encoding**, čo pomôže modelu pochopiť, v akom poradí sú slová poskladané v texte.

Predspracované sekvencie vstupujú do enkodéru. Jeho úlohou je porozumieť kontextu a vytvoriť vektor fixnej veľkosti, ktorý reprezentuje celú vstupnú sekvenciu. Treba podotknúť, že enkodér negeneruje novú informáciu, len spracuje vstup. Dekodér tento vektor prevezme ako jeden zo svojich vstupov. Ďalší vstup je na obrázku 2.1 pomenovaný **Outputs**

(shifted right). V praxi funguje nasledovne. Od enkodéru má skomprimovanú vstupnú sekvenciu a na vlastnom vstupe má štartovaciu sekvenciu tokenov. Na základe toho vygeneruje pre každý token v slovnej zásobe pravdepodobnosť, že bude nasledovať. Typicky vyberie ten najpravdepodobnejší token a pridá ho k výstupnej sekvencii, ktorá je zavedená do vstupu dekodéru. Takto sa postupne generujú nasledujúce tokeny, pokiaľ dekodér nevygeneruje ukončovací token.

Predchádzajúci odstavec vysvetľuje princíp fungovania Transformera, ktorý má enkodér aj dekodér, často sa nazýva **Sequence-to-sequence** (Seq2Seq) model. Takéto modely sú vhodné na preklad alebo sumarizovanie textu. Známe Seq2Seq modely sú napríklad T5 či Bart. Existujú však aj modely, ktoré:

- Majú iba enkodér. Ich použitie je výhodné pre klasifikovanie dát napríklad analýza sentimentu. Voláme ich **AutoEncoding** modely.
- Majú iba dekodér. Sú vhodné na generovanie textu. Voláme ich **AutoRegressive** modely. Napríklad gpt či Bloom.

V tejto práci boli na finetuning použité modely, ktoré stavajú na architektúre Transformer. Tieto modely ďalej posúvajú a vylepšujú pôvodný koncept za účelom dosiahnutia lepších výsledkov.

2.1.1 Jazykový model CodeT5+

V článku **CodeT5+: Open Code Large Language Models for Code Understanding and Generation** [16] bola predstavená rodina modelov codeT5+ určená predovšetkým pre úlohy spojené s programovaním. Vývoj zastrešovala spoločnosť Salesforce. Tieto modely sú architektúry encoder-decoder, ktorá je upravená tak, aby vedela pracovať aj v režimoch iba s enkodérom alebo iba s dekodérom. Trénovanie týchto modelov prebiehalo na zmesi úloh, konkrétne:

- **Span denoising** – Náhodná výmena 15% tokenov v texte za maskovacie tokeny. Model má za úlohu doplniť maskované tokeny.
- **Decoder-only causal language modelling** – Model dostane na začiatku len štartovací token a z neho má vytvoriť syntakticky správny kus kódu.
- **Seq2Seq causal language modelling** – Model dostane 10 % až 90 % sekvencie, jeho úlohou je správne doplniť chýbajúcu časť.

Kombinovanie úloh pri trénovaní umožnilo modelu naučiť sa zmysluplné reprezentácie kontextu zdrojového kódu a získavať chýbajúce informácie na rôznych úrovniach: časti funkcií, časti celých programov alebo celé programy.

Trénovanie modelov prebiehalo na datasete Github Code dataset na vybraných programovacích jazykoch: Python, Java, Ruby, Javascript, Go, PHO, C, C++, C#. Rodina obsahuje modely od veľkosti 220 miliónov do 16 miliárd. V tejto práci sú použité modely codet5p-220m a codet5p-770m pre finetuning.

2.1.2 Jazykový model Longformer

Článok **Longformer: The Long-Document Transformer** [2] predstavil nový koncept Transformera navrhnutého na spracovanie dlhých dokumentov, preto je aj nazvaný longformer. Kým bežné modely trasnformer, napríklad T5 a jeho varianty, vykazujú problémy pri

spracovávaní dlhších textov kvôli kvadratickej náročnosti výpočtovej zložitosti mechanizmu self-attention, longformer toto obmedzenie rieši elegantnejšie.

Kľúčová inovácia longformera spočíva v jeho úprave attention mechanizmu. Štandardný self-attention mechanizmus nahrádza attention mechanizmom postaveným na posuvnom okienku, ktoré je výpočtovo menej náročné a dokáže efektívne pracovať s dlhými sekvenciami. Model udržiava pevné kontextové okno pre každý token, čím sa stropujú zdroje na spracovanie každého tokenu.

Longformer využíva aj takzvaný global attention pre konkrétne tokeny, jeden token môže vplývať na všetky ostatné tokeny v sekvencii. Táto vlastnosť je výhodná napríklad pre úlohy na odpovedanie otázok, kde tokeny signalizujúce, že ide o otázku, majú vplyv na celú sekvenciu. Tieto inovácie umožňujú spracovávať dlhé sekvencie pri zachovaní presnosti a znížení výpočtovej zložitosti.

2.2 Frameworky na pracovanie s neurónovými sieťami

Táto kapitola je venovaná predstaveniu použitých nástrojov, ktoré prekonávajú medzery medzi komplexnou teóriou a reálnou implementáciou, sprístupňujú strojové učenie verejnosti. Primárne budú predstavené nástroje od spoločnosti Huggingface a knižnica **Ctranslate2** od OpenNMT. Exitujú aj ďalšie knižnice, ako napríklad TensorFlow, PyTorch, Keras a iné. Pričom napríklad PyTorch a TensorFlow tvoria podklad pre iné knižnice vyššej úrovne abstrakcie. V nasledujúcich sekciách sú vysvetlené výhody vybraných nástrojov, ich možnosti a funkcionalita.

2.2.1 Nástroje vyvíjané spoločnosťou HuggingFace

Spoločnosť HuggingFace zanechala nezmazateľnú stopu v oblasti strojového učenia. Hlavnou výhodou ich systému nástrojov je jednoduchá možnosť používania najmodernejších modelov.

Ekosystém HuggingFace siaha ďaleko za už dobre známu knižnicu **transformers**. Zahŕňa širokú škálu zdrojov vrátane modelov, datasetov, metrík hodnotenia, ktoré sú dostupné prostredníctvom intuitívneho rozhrania. HuggingFace hub slúži ako úložisko už natrénovaných modelov spolu s pridruženými datasetmi a metrikami, čo uľahčuje zdieľanie a použitie modelov na rôzne úlohy.

Transformers je komplexná knižnica poskytujúca rozhranie pre inferenciu na modeloch dostupných na huggingface hube. Jej rozhranie poskytuje vysokoúrovňové funkcie, vďaka ktorým môžeme použiť modely na doslova pár riadkoch kódu, takisto však poskytuje rozhranie aj pre pokročilejších užívateľov. Ako názov napovedá, je špeciálne navrhnutá na prácu s modelmi typu Transformer. Poskytuje aj rozhranie pre trénovanie cez Trainer API, ktoré zjednodušuje proces finetuningu.

Ako sa hovorí, model je len tak dobrý, ako dataset, na ktorom bol trénovaný. Preto HuggingFace vyvíja aj knižnicu **datasets**. Sprístupňuje kolekcie datasetov dostupných na huggingface hube v rozhraní programovacieho jazyka Python. Cieľom knižnice je eliminovať únavné aspekty manipulácie a predbežného spracovania datasetov, čím sa uľahčuje a zrýchľuje proces adaptácie modelu.

Ďalším užitočným nástrojom je knižnica **evaluate**, ktorá umožňuje používateľom využívať populárne hodnotiace metriky na hodnotenie výkonnosti modelu.

Genialita ekosystému HuggingFace nespočíva len v poskytovaní týchto zdrojov, ale aj v ich vzájomnom prepojení, čím sa uľahčuje riešenie komplexných úloh.

2.2.2 Inferencia pomocou Ctranslate2

Ctranslate2 je výkonné inferenčné prostredie pre Transformer modely vyvíjané skupinou OpenNMT. Táto knižnica je primárne vyvíjaná v jazyku C++, pre maximálnu rýchlosť, ale poskytuje aj rozhranie pre Python. Jej orientácia pre produkčné použitie zaisťuje efektívne využívanie prostriedkov a stabilitu. Táto knižnica je použitá ako behové prostredie pre adaptované modely v časti 5.1. Vybral som ju hlavne pre možnosti inferencie na grafických kartách, ako aj na procesore, a možnosti špecifikovania kvantizácie.

V tomto projekte pôsobí skôr ako *proof-of-concept*, že adaptované modely sú použiteľné na bežnom hardware.

Kapitola 3

Príprava dát a fine-tuning modelov

3.1 Konvencie dokumentovania Python zdrojových kódov

Štandardný spôsob dokumentovania Python kódu je použitie **docstring**. Docstring je text v prirodzenom jazyku, ktorý popisuje daný objekt, alebo modul. Formálne je definovaný v PEP257 [10], ktorý stanovuje syntaktickú formu dokumentovania. Teda, to že docstring musí byť v trojitých úvodzovkách hneď za prototypom objektu alebo na začiatku modulu a je prístupný ako `__doc__` atribút daného objektu. Nestanovuje však sémantiku, ani vnútornú štruktúru.

3.1.1 Google Style

Google definuje štruktúru docstringov v **Google Python Style Guide** [4]. Je to najpoužívanejšia konvencia dokumentácie, používajú ju napríklad projekty Tensorflow, Django, SciPy. Popisuje, pre ktoré funkcie je dokumentácia povinná a pre ktoré je voliteľná. Ďalej definuje, ako by mali byť docstringy napísané, ich štruktúru:

- Krátky popis funkcie, maximálne na niekoľko viet. Nie je uvedený žiadnym kľúčovým slovom.
- `Args:` – Argumenty funkcie alebo metódy. Zoznam, kde je každý argument uvedený v tvare "meno (dátový typ): popis".
- `Returns:` – Význam a typ návratovej hodnoty. Ak je objekt generátor, odsek je uvedený slovom `Yields:`.
- `Raises:` – List výnimiek, ktoré môže objekt vyvolať. Zapísaný v tvare "dátový typ výnimky: popis kedy sa vyvolá".
- `Attributes:` – List verejných atribútov triedy.
- Ďalšie odseky uvedené kľúčovým slovom definované užívateľom. Napríklad `Examples` či `See also`.
- Voliteľný obsiahly popis.

```
"""One line summary.

In depth description of what the function or method does.
Usually on multiple lines.

Args:
    first (type): First parameter description.
    second: Second parameter description.

Returns:
    One or more lines describing the return value or values.

Raises:
    Exception: In case of an error, an exception is raised.
"""
```

Obrázek 3.1: Ukážka s popisom pre Google Style typ dokumentácie.

3.1.2 Numpydoc Style

Konvencia pre písanie dokumentácie v štýle Numpydoc je definovaná v **Numpydoc documentation** [8]. Tento štýl je používaný v projektoch ako napríklad Numpy, SciPy, scikit-learn. Je postavená na syntaxi značkovacieho jazyka reStructuredText.

Numpydoc definuje nasledovné sekcie dokumentácie:

- Popis objektu na jeden riadok.
- Obsiahlejší popis objektu, popisuje funkcionalitu, nie implementáciu.
- `Parameters:` – Argumenty funkcie alebo metódy. V tvare "meno : popis na novom riadku".
- `Attributes:` – Zoznam verejných atribútov triedy.
- `Returns:` – Vysvetlenie významu návratovej hodnoty.
- `Yields:` – Pre generátory popis vrátených hodnôt.
- `Raises:` – Zoznam výnimiek, ktoré môžu nastať počas behu, ich dátový typ a popis.
- A ďalšie sekcie: `See Also`, `Notes`, `Examples`, `Warns`, `Warnings`, `References`.

Táto konvencia dovoľuje použiť len definované názvy sekcií a doporučuje, v akom poradí majú byť usporiadané.

```
"""One line summary.

In depth description of what the function or method does.
Usually on multiple lines.

Parameters
----------
first :
    First parameter description.
second :
    Second parameter description.

Returns
-------
int
    One or more lines describing the return value or values.

Raises
------
Exception
    In case of an error, an exception is raised.
"""
```

Obrázek 3.2: Ukážka Numpydoc štýlu dokumentácie.

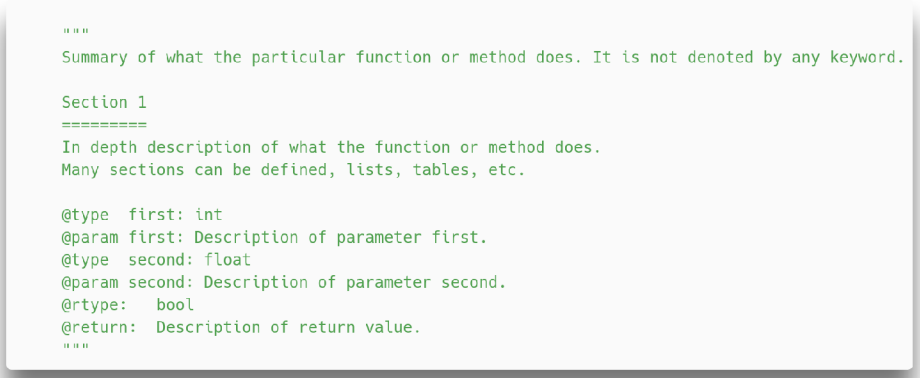
3.1.3 Epytext Style

Historicky inšpirovaný javadoc štýlom, táto konvencia je definovaná v **Epytext documentation** [3]. Táto konvencia je používaná napríklad v projekte twisted. Epytext je značkovací jazyk, ktorý takisto definuje štruktúru jednotlivých častí dokumentácie. Textové polia, ako popis, sú formátované konštrukciami podobnými značkovaciemu jazyku markdown. Po časti popisujúcej funkcionalitu nasleduje popis argumentov, atribútov, návratových hodnôt, atď. Tieto časti popisujeme pomocou tagov definovaných v dokumentácii Epytextu, je možné si definovať vlastné tagy.

Popis niektorých základných tagov:

- `@param` – Popis parametra alebo argumentu, sú definované aliasy @parameter, @arg, @argument.
- `@type` – Dátový typ argumentu.
- `@return` – Popis návratovej hodnoty, alias je @returns.
- `@rtype` – Dátový typ návratovej hodnoty, alias @returntype.
- `@raise` – Výnimky, ktoré môžu vzniknúť počas behu programu.

Samotná štruktúra používania tagov je názorne ukázaná na obrázku 3.3.



```
"""
Summary of what the particular function or method does. It is not denoted by any keyword.

Section 1
=========
In depth description of what the function or method does.
Many sections can be defined, lists, tables, etc.

@type  first: int
@param first: Description of parameter first.
@type  second: float
@param second: Description of parameter second.
@rtype:   bool
@return:  Description of return value.
"""
```

Obrázek 3.3: Ukážka Epytext štýlu dokumentácie.

3.2 Finetuning vybraných modelov

Finetuning je v podstate proces ladenia predtrénovaného modelu, tak aby podával lepší výkon v špecifikovanej doméne. Aby sme dosiahli maximálny výkon pre užšie zameranú oblasť, musíme upraviť model na dátach špecifických pre danú úlohu.

Potreba finetuningu je ešte dôležitejšia v oblasti spracovania prirodzeného jazyka. Jazyk so svojimi jemnými nuansami, kontextovo závislými významami a rôznymi odchýlkami si vyžaduje model, ktorý porozumie špecifikám problému, a tak nájde optimálne riešenie. Pokiaľ je doména riešeného problému dostatočne presne zadefinovaná a nie príliš obšírna, finetuning umožňuje použiť menšie modely adaptované na doménových dátach. Dobre adaptovaný model môže v danej doméne dosahovať lepšie výsledky ako podstatne väčší model, natrénovaný len na všeobecných dátach.

Táto kapitola popisuje využitie nástrojov HuggingFace na prípravu dát a na samotný finetuning. V tejto práci sa využívajú modely **CodeT5+** a **Longformer encoder-decoder**. Práve tieto modely boli vybrané pre ich nízke požiadavky na hardware, teda sa dajú spustiť aj na bežne dostupných počítačoch a majú vyhovujúcu architektúru s enkodérom aj dekodérom.

3.2.1 Príprava trénovacích dát

Trénovacie dáta pochádzajú z korpusu CodeSearchNet. Tento dataset vznikol ako dáta pre CodeSearchNet Challenge predstavený v článku **CodeSearchNet Challenge Evaluating the State of Semantic Code Search** [5]. Dataset obsahuje viac ako 6 miliónov funkcií v šiestich programovacích jazykoch (Go, Python, Java, Ruby, PHP, JavaScript) a slovný popis pre 2 milióny funkcií. Pôvodne je tento dataset určený na trénovanie a evaluáciu modelov generujúcich zdrojový kód z popisu v prirodzenom jazyku.

Funkcie sú získané z open source projektov dostupných na Github. Zdrojové kódy boli spracované tak, aby jedna položka v datasete bola jedna funkcia s metadátami o tom, z akého repozitára pochádza, z akého súboru, meno a telo funkcie. Takto bolo získaných 1 156 085 funkcií v jazyku Python, z toho je 503 502 s nejakým druhom dokumentácie. Avšak

po preskúmaní som zistil, že vo veľa prípadoch je dokumentácia neštruktúrovaná, a preto nie je vhodná pre strojové spracovanie.

Dataset som spracovával opakovane, postupne som sa učil a zlepšoval postupy získavania relevantných dát z korpusu. Skripty použité na spracovávanie datasetu sú v zložke projektu `dataset_processing`. Základom je skript `dataset_processing.py`. Tento skript načíta časť datasetu so zdrojovými kódmi v jazyku Python. Jeden záznam v datasete má položku s funkciou a položku pre extrahovaný docstring. Funkcia stále obsahuje docstring, preto je prvý krok odstrániť ho. Pre účely trénovania je vhodné mať na jednej strane čistú funkciu bez popisu, na druhej strane vzorovú dokumentáciu.

Rozsah dokumentácie v jednotlivých funkciách je rozdielny. Veľká časť je príliš stručná, typicky jedna veta, a naopak niektoré sú až príliš dlhé. Príliš krátke popisy by pravdepodobne viedli model ku generovaniu príliš stručnej dokumentácie a nemajú vhodnú štruktúru. Odstránené boli aj funkcie s príliš dlhými popismi. Dlhé popisy obsahujú aj informácie, ktoré nie sú odvoditeľné z programu funkcie. Preto nie je na mieste očakávať, že by trénované modely vedeli odvodiť informácie navyše, ktoré nie sú vo vstupnom kóde a kompromitovali by trénovanie. Takto sa znížil počet dát zo zhruba 500 000 na približne 300 000.

Cieľom je natrénovať model, ktorý dodržuje štruktúru dokumentácie. Generovaná dokumentácia by mala podliehať niektorému zo štýlov v kapitole 3.1. Preto je nutné model trénovať len na vybranej konvencii dokumentácie. Dva najznámejšie štýly dokumentácie (Google Style a Numpydoc) som popísal pomocou gramatík (v prílohe A).

Na spracovanie textu pomocou gramatiky som použil Python knižnicu **Lark**[1]. Je vhodná na spracovávanie bezkontextových gramatík, teda primárne na programovacie jazyky. Syntax pre popis gramatiky je obohatená o predpripravené neterminály pre prvky ako medzery a rôzne odriadkovania, a taktiež je možné spracovanie ovplyvniť direktívami, napríklad na nespracovávanie bielych znakov. Gramatiky sú navrhnuté tak, aby počítali s rôznym formátovaním, a do istej povolenej miery aj s alternáciami štýlu.

Filtrovanie datasetu pomocou gramatík

Prefiltrovanie pomocou gramatík zúžilo dataset na približne 30 000 vzoriek pre Google Style dokumentáciu a 13 000 pre Numpydoc štýl dokumentovania. Pre ďalší postup bola použitá len časť datasetu s Google Style dokumentáciou. Dôvody pre tento krok sú hlavne jeho väčšia rozšírenosť a väčšia dostupnosť trénovacích dát.

Korpus trénovaçích dát sa však zúžil až príliš. Je možné, že v pôvodnom korpuse bolo viac funkcií anotovaných Google štýlom, avšak gramatika ich vylúčila. Je to preto, že nuansy prirodzeného jazyka a rôznych autorov vytvárajú rôzne nepredvídané konštrukcie. Snažil som sa prejsť dataset, aspoň sčasti, a identifikovať anomálie, prípadne ich zahrnúť v gramatike. Avšak tým, že dokumentácia je v prirodzenom jazyku a nemusí byť písaná striktne podľa konvencií, bolo ťažké navrhnúť gramatiku tak, aby bola úplne spoľahlivá. Návrh gramatiky podliehal prístupu zamietnuť to, čo je otázne.

Generovanie dokumentácie pre trénovacie účely

Pre zlepšenie možností adaptácie som sa rozhodol rozšíriť dataset pomocou veľkého jazykového modelu **gpt-3.5-turbo-0125** od OpenAI. Táto možnosť sa ukázala ako výhodná, pretože daný model je dostatočne výkonný na túto úlohu a má skvelú podporu API. Skript, ktorý anotuje dataset, je v priečinku `dataset_processing` pod menom `annotate_gpt.py`.

[1]Repozitár lark knižnice: https://github.com/lark-parser/lark

Skript paralelne vo viacerých vláknach posiela dotazy na OpenAI servery. Program pri spustení inicializuje zadaný počet vlákien a načíta časť datasetu. Vzhľadom na obmedzený počet dotazov na deň a na minútu bol dataset anotovaný po častiach a následne spojený dokopy. Obmedzenie bolo nastavené na 10 000 dotazov za deň a 500 dotazov za minútu. Architektúra paralelne pracujúcich vlákien urýchlila anotáciu, keďže jedno vlákno nedokázalo maximálne využiť obmedzenie počtov dotazov za minútu. Každé vlákno dostane na začiatku časť datasetu, ktorú má spracovať. Ak server začne odmietať požiadavky kvôli prekročeniu počtu requestov za minútu, tak počká niekoľko sekúnd a začne posielať znova. Na konci sa už anotované časti datasetu zo všetkých vlákien spoja dokopy a uložia. Týmto spôsobom sa mi podarilo anotovať 65 000 funkcií navyše.

Výsledné štatistiky datasetu

Dataset použitý na trénovanie a validáciu je spojený z ručne anotovaných funkcií a z automaticky anotovaných funkcií z datasetu code_search_net. Rozdelenie je nasledovné:

- trénovacia časť má 81 000 funkcií, zmiešaných z ručne a automaticky anotovaných funkci.
- testovacia časť má 7 000 funkcií, iba ručne anotované funkcie.
- Validačná časť má 7 900 funkcií, iba ručne anotované funkcie.

3.2.2 Finetuning pomocou trainera z knižnice transformers

Prvé pokusy finetuningu boli robené pomocou ukážkového skriptu dostupného na Github stránkach knižnice transformers[2]. Skripty naprogramované v rámci tejto práce použité na finetuning vychádzajú zo spomenutého skriptu pôvodne určeného na trénovanie prekladu medzi prirodzenými jazykmi, čo je podobná úloha ako generovanie dokumentácie zo zdrojového kódu. Prakticky je to preklad zdrojového kódu do ľudsky čitateľnej štruktúrovanej dokumentácie.

Daný ukážkový skript som prepísal tak, aby bol modulárnejší a mal väčšiu funkcionalitu, obsahuje aj pôvodné časti kódu pôvodného ukážkového programu. Nie som výlučným autorom zdrojových kódov v zložke `fine_tuning`. Program je rozdelený do viacerých modulov tak, aby mal každý modul jasne definovanú zodpovednosť. Obsah zložky `fine_tuning` je nasledovný:

- `fine_tuning_main.py`
- `model_args.py`
- `trainer_stat_collector.py`
- `trainer_container.py`

Popis funkcionality jednotlivých modulov:

fine_tuning_main.py je vstupný bod programu. Jeho funkciou je načítať json konfiguráciu, podľa nej nastaviť logovanie, načítať dataset, model, `TrainerContainer`, `TrainerStatCollector` a po trénovaní nahrať model na HuggingFace hub a vytvoriť štatistiky.

[2] https://github.com/huggingface/transformers/blob/main/examples/pytorch/translation/run_translation.py

model_args.py je modul definujúci triedy s parametrami dôležitými pre načítanie modelu a trénovacích parametrov. Parametre sa po spustení programu načítajú z konfigurácie.

trainer_stat_collector.py definuje triedu `TrainerStatCollector`. Táto trieda je použitá ako callback trainera. Počas trénovania sú volané definované metódy a kolektor ukladá dôležité údaje z trénovania. Následne je po trénovaní vygenerovaná správa, formátovaná ako markdown. Správa obsahuje nasledovné údaje:

- Dátum a čas začiatku trénovania
- Názov trénovaného modelu
- Hash aktuálneho Git commitu
- Zadaná konfiguacia trénovania
- Graf priebehu loss funkcie v závislosti od epochy
- Metriky pre trénovanie, evaluáciu a generovanie
- 5 ukážkových funkcií s vygenerovanou dokumentáciou

Spolu so správou kolektor vytvorí priečinok, do ktorého sa ukladajú logovacie správy a viac vygenerovaných predikcií.

trainer_container.py zapúzdruje `Seq2SeqTrainer` z transformers knižnice.
Jeho zodpovednosť je pripraviť a tokenizovať dataset, rieši padding pri tokenizovaní, samotné trénovanie a vytváranie štatistík.

Trénovanie sa spustí z programu `fine_tuning_main.py`. Parametre trénovania sa načítajú z konfigurácie. Konfigurácia obsahuje kmeňový model, ktorý sa bude adaptovať, parametre trénovania, názov datasetu a aká časť z neho sa použije a ostatné parametre. Počas trénovania program do konzoly a do súboru súčasne zapisuje priebeh. Na konci sa do určeného adresára uloží adaptovaný model a do adresára pre štatistiky a logy sa uloží správa obsahujúce dôležité informácie o trénovaní.

Kapitola 4

Hodnotenie kvality modelov

Dôležitý krok, ktorý nasleduje po natrénovaní jazykového modelu, je ohodnotenie jeho schopností. Evaluácia modelov je kritická fáza v strojovom učení, keďže nám ponúka pohľad na efektívnosť modelu a vytvára cestu pre zlepšenie. V úlohe ako je generovanie Python dokumentácie je možné posúdiť kvalitu generovaných docstringov porovnávaním s referenčnými docstringami pomocou vhodných metrík.

Hodnotenie jazykových modelov nie je úplne priamočiara úloha, kvôli zložitosti prirodzeného jazyka. Jazykové modely sa môžu hodnotiť automaticky alebo manuálne.

- Automatické hodnotenie zahŕňa použitie metrík a rôznych algoritmických postupov.
- Manuálne hodnotenie je posudzovanie kvality generácie pomocou ľudskej inteligencie a intersubjektívneho postoja k hodnotenej úlohe.

Na rýchle a objektívne hodnotenie som vytvoril nástroj, ktorý na vybranom ručne anotovanom korpuse vyhodnotí kvalitu generovanej dokumentácie porovnávaním voči referencii.

Korpus na hodnotenie pozostáva zo 45 funkcií. Funkcie sú vybrané náhodne z nepoužitej časti korpusu code_search_net, ktoré som ručne anotoval tak, aby mala každá funkcia dve docstring dokumentácie. Je to preto, že sa objektívne nedá povedať o jednej formulácii, že je jediná dobrá. Anotované sú na dvakrát, tak aby boli popisy rôzne, sémanticky a syntakticky správne. Evaluátor potom môže porovnávať s tým popisom, ktorý má väčšiu zhodu s generovaným. Tento spôsob umožňuje aspoň o jeden stupeň väčšiu objektivitu.

Na internete som nenašiel dedikovanú metódu, metriku, na hodnotenie kvality generovanej dokumentácie. Preto som sa zameral na hľadanie spôsobov ako objektívne ohodnotiť kvalitu generovaného textu oproti referenčnému. Po naštudovaní problematiky som dospel k záveru, že použitie jednej vybranej metriky by nebolo dostatočne objektívne na účely tejto práce, keďže ani jedna nie je vyvíjaná s cieľom porovnávať štruktúrovanú programovú dokumentáciu.

Generovaná dokumentácia bola hodnotená viacerými metrikami, ktorých skóre je normalizované tak, aby bola výsledná hodnota v intervale $\langle 0, 1 \rangle$. Hlavný výsledok je priemer všetkých metrík, ale v rozšírenom vyhodnotení je vypočítaný aj súčin, suma a rozptyl. V nasledujúcej časti sú popísané použité metriky.

4.1 Metriky použité na hodnotenie kvality vygenerovaných modelov

Sacrebleu metrika na hodnotenie podobnosti textu

Jedným zo spôsobov, ako porovnávať text voči referencii, je použitie metriky SacreBLEU, predstavenej v článku **A Call for Clarity in Reporting BLEU Scores** [11]. SacreBLEU stavia na pôvodnom BLEU skórovaní a navrhuje, ako vylepšiť jeho nedostatky. Nasledujúci odstavec popisuje ako funguje porovnávanie referencie voči generovanému textu pre BLEU.

Už v roku 2002 bola predstavená metrika BLEU v článku **BLEU: A Method for Automatic Evaluation of Machine Translation** [9]. Hlavnou motiváciou bolo vytvoriť automatizovaný spôsob, ako hodnotiť kvalitu strojového prekladu textu v prirodzenom jazyku. Autori popisujú spôsob, ako efektívne a nezávisle na jazyku porovnať sety textov tak, aby boli čo najviac korelované s ľudským hodnotením kvality prekladu. Hodnotenie kvality spočíva v numerickom hodnotení podobnosti generovaného textu voči profesionálne vytvoreným referenčným prekladom.

Porovnávanie podobnosti textu prebieha hľadaním zhodných n-gramov a ich spočítaním. Tieto zhody sú pozične nezávislé, a čim viac zhody, tým lepšie skóre. Skóre P_n pre jeden reťazec pre zvolené n-gramy sa vypočíta podľa vzťahu 4.1.

$$P_n = \frac{\#matched\_n\text{-}grams}{\#candidate\_n\text{-}grams} \tag{4.1}$$

$\#matched\_n\text{-}grams$ – počet zhodných n-gramov
$\#candidate\_n\text{-}grams$ – počet n-gramov v predikcii

Skóre pre celý dataset je obdobne vypočítané ako suma všetkých zhodných n-gramov vydelené súčtom n-gramov v predikciách.

Celkové skóre je ovplyvnené faktorom *brevity penalty*, skrátene *BP*. Tento faktor penalizuje preklady, ktoré sú kratšie ako referencie. Je to preto, že model môže mať zdanlivo dobré skóre podľa zhody v n-gramoch, ale v skutočnosti len nevygeneroval dosť dlhé sekvencie. V optimálnom prípade má skóre *BP* hodnotu 1, za predpokladu že generované texty majú rovnakú alebo väčšiu dĺžku ako referencie. Inak proporcionálne penalizuje podľa skrátenia dĺžky. Brevity penalty sa vypočíta podľa vzťahu 4.2.

$$BP = \begin{cases} 1 & \text{ak } c > r \\ e^{(1-c/r)} & \text{ak } c \leq r \end{cases} \tag{4.2}$$

c – dĺžka generovaného textu
r – dĺžka referenčného korpusu

BLEU skóre je geometrický priemer skóre pre jednotlivé n-gramy vynásobený BP, vo vzťahu 4.3. Samotné skóre je číslo v intervale $\langle 0, 1 \rangle$.

$$\text{BLEU} = BP * \exp\left(\sum_{n=1}^{N} w_n \log P_n\right) \tag{4.3}$$

Autori článku použili $N = 4$, teda unigramy až 4-gramy, keďže s týmto dosahovali najlepšie výsledky v porovnaní s ručným hodnotením.

BLEU skóre je rozšírené pre porovnávanie jazykových modelov, avšak neposkytuje objektívne porovnanie. Autori SacreBLEU popisujú problémy pri porovnávaní skóre medzi rôznymi projektami.

BLEU nepopisuje jednu konkrétnu metriku, ale popis metódy, ktorá môže byť parametrizovaná. Niektoré parametre, ktoré vplývajú na skórovanie [11]:

- Počet použitých referencií.
- Pre viac referencií nie je stanovené, ako počítať penalizáciu za dĺžku.
- Maximálny stupeň n-gramov.
- Spracovanie n-gramov s nulovým počtom.
- Rozdiely v predspracovaní referenčných výsledkov – odstraňovanie diakritiky z textu, odstraňovanie špeciálnych znakov, tokenizácia atď.

Vo vedeckých prácach nemusí byť jasne povedané, aké parametre pre BLEU vyhodnotenie boli použité, čo sťažuje porovnávanie skóre medzi rôznymi projektmi.

Metrika SacreBLEU rieši tento problém vytvorením nástroja, Python knižnice, ktorý poskytuje univerzálny spôsob vyhodnotenia generovaného textu. Teda v jadre je to BLEU, ale má jasne definované parametre a spôsob vyhodnotenia. SacreBLEU nástroj je dostupný na stránkach Githubu[1].

V projekte je táto metriku integrovaná pomocou knižnice evaluate od spoločnosti HuggingFace. Modul je popísaný na stránkach HuggingFace spaces[2]. Trieda zapúzdrujúca implementáciu `SacreBleuEvaluator` je implementovaná v súbore `metrics.py`. Konfigurácia je v prílohe B.1.

Meteor metrika na hodnotenie podobnosti textu

Meteor je metrika, pôvodne predstavená v článku **METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments** [1], určená na hodnotenie kvality prekladu textu v prirodzenom jazyku. Meteor je akronym pre **M**etric For **E**valuation of **T**ranslation with **E**xplicit **OR**dering. Autori navrhujú túto metriku ako vylepšenie konkrétnych nedostatkov BLEU metriky. Je založená na explicitnom porovnávaní slova ku slovu medzi referenciou a výstupom modelu.

Implementačne Meteor porovnáva unigramy medzi referenciou a generovaným textom. Snaží sa namapovať čo najviac unigramov medzi dvoma textami tak, aby každý unigram v každom texte bol mapovaný na nula alebo jeden unigram v druhom texte. Takto vytvorené mapovanie autori pomenovali alignment. Alignment sa inkrementálne vytvára po úsekoch. Vo výsledku je alignment pole dvojíc, kde dvojica popisuje index unigramu z jednej sekvencie a index unigramu z druhej sekvencie, ktorý mu zodpovedá. Po vytvorení alignmentu sa skóre vypočíta nasledovne. Najprv sa vypočíta unigram precission (P) 4.4 a unigram recall (R) 4.5.

$$P = \frac{m}{\omega_t} \tag{4.4}$$

$$R = \frac{m}{\omega_r} \tag{4.5}$$

[1]SacreBLEU repozitár: https://github.com/mjpost/sacreBLEU

[2]HuggingFace integrácia SacreBLEU: https://huggingface.co/spaces/evaluate-metric/sacrebleu

m – počet namapovaných unigramov
ω_t – počet unigramov v generovanom texte
ω_r – počet unigramov v referenčnom texte

Následne sa spočíta *Fmean* kombinovaním P a R pomocou harmonickej strednej hodnoty 4.6.

$$Fmean = \frac{10PR}{R + 9P} \tag{4.6}$$

Dlhšie zhodné sekvencie sú zahrnuté pomocou veličiny *Penalty* 4.7 pre daný alignment.

$$Penalty = 0.5 * \left(\frac{\#chunks}{\#unigrams\_matched} \right)^3 \tag{4.7}$$

$\#chunks$ – počet zhodných n-gramov
$\#unigrams\_matched$ – počet namapovaných unigramov

Finálne skóre sa vypočíta vo vzťahu 4.8.

$$Score = Fmean * (1 - Penalty) \tag{4.8}$$

V implementácii je táto metrika integrovaná cez rozhranie knižnice evaluate od spoločnosti HuggingFace. Modul je popísaný na stránkach HuggingFace spaces[3], pričom táto implementácia vnútorne používa knižnicu nltk[4] na evaluáciu Meteor skóre. Implementácia v nltk je založená na neskôr vydanom článku **METEOR: An Automatic Metric for MT Evaluation with High Levels of Correlation with Human Judgments** [6]. Hlavné rozdiely sú v počítaní *Fmean* 4.9 a *Penalty* 4.10.

$$Fmean = \frac{P.R}{\alpha * P + (1 - \alpha) * R} \tag{4.9}$$

$$Penalty = \gamma * \left(\frac{\#chunks}{\#unigrams\_matched} \right)^{\beta} \tag{4.10}$$

Tieto rozdiely prinášajú možnosť meniť skórovanie tejto metriky na zvolenú úlohu pomocou parametrov α, β a γ. Možnosť zmeny týchto parametrov bola v implementácii využitá na precíznejšie ohodnotenie kvality generovanej dokumentácie. Parametre sú zvolené nasledovne:

- $\alpha = 0.4$, východzia hodnota je 0.9. Takto sa väčší dôraz kladie na porovnanie voči referencii.
- $\beta = 6$, východzia hodnota je 3. Zväčšenie parametru oproti východzej hodnote spôsobilo lepšie rozdelenie dobrých a zlých predikcií.
- $\gamma = 0.5$, východzia hodnota je 0.5.

Táto metrika je zapúzdrená do triedy `MeteorEvaluator`, implementovaná je v súbore `metrics.py`.

[3]Repozitár meteor metriky: https://huggingface.co/spaces/evaluate-metric/meteor
[4]Repozitár knižnice nltk: https://github.com/nltk/nltk/blob/develop/nltk

Rouge metrika na hodnotenie sumarizácie

Článok **Rouge: A package for Automatic Evalaution of Summaries** [7] predstavuje nástroje ako hodnotiť kvalitu sumarizácie textu porovnávaním vzoru oproti generovaným sumarizáciám. Rouge je akronym pre **R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation. Zahŕňa spôsoby, ako automaticky ohodnotiť kvalitu zhrnutí textov porovnávaním s referenčnými, ideálnymi sumármi vytvorenými ľuďmi. Metriky počítajú počet prekrývajúcich sa celkov, napríklad n-gramov, reťazcov alebo dvojíc slov medzi generovanými a ľudsky tvorenými sumármi. Rouge zahŕňa štyri metriky: Rouge-N, Rouge-L, Rouge-W a Rouge-S.

Rouge-N je podiel medzi počtom zhodných n-gramov a počtom n-gramov v referenčnom výsledku. Je to podobné ako v BLEU, až na to, že tam je v menovateli počet n-gramov generovaného textu. Teda Rouge-N sa viaže viac na referenčný text, kdežto BLEU sa zameriava na presnosť v rámci generovaného textu. Rouge-N sa pre danú dĺžku n-gramov vypočíta podľa vzťahu 4.11. V praxi sa prevažne používa Rouge-1 a Rouge-2, dlhšie n-gramy sa nehľadajú.

$$\text{Rouge-N} = \frac{\#matched\_n\text{-}grams}{\#reference\_n\text{-}grams} \tag{4.11}$$

$\#matched\_n\text{-}grams$ – počet zhodných n-gramov
$\#reference\_n\text{-}grams$ – počet n-gramov v referencii

Rouge-L hľadá najdlhšie súvislé sekvencie zhodných n-gramov medzi generovaných textom a referenciou. Skóre na úrovni viet sa počíta podľa vzťahu 4.15.

$$R_{lcs} = \frac{LCS(X,Y)}{m} \tag{4.12}$$

$$P_{lcs} = \frac{LCS(X,Y)}{n} \tag{4.13}$$

$$\beta = R_{lcs}/R_{lcs} \tag{4.14}$$

$$F_{lcs} = \frac{(1+\beta^2)R_{lcs}P_{lcs}}{R_{lcs}+\beta^2 Plcs} \tag{4.15}$$

X – referenčný text
Y – generovaný text
$LCS(X,Y)$ – dĺžka najdlhšieho spoločného podreťazca medzi X a Y
m – dĺžka reťazca X
n – dĺžka reťazca Y
F_{lcs} – výsledné skóre Rouge-L

Z uvedených rovníc vyplýva, že Rouge-L bude mať hodnotu 1, ak $X = Y$, a naopak 0 v prípade $LCS(X,Y) = 0$.

Rouge-W je váhovaná najdlhšia spoločná sekvencia. Rozdiel medzi Rouge-L je zrejmý na príklade:

- X : [ABCDEFG]
- Y : [ABCDHIK]

- z : [AHBKCID]

X je referenčný reťazec, Y a Z sú generované reťazce. Podľa Rouge-L majú reťazce Y aj Z oproti X rovnaké skóre, keďže ten ignoruje kontext. Rouge-W sa zameriava len na neprerušené postupnosti, preto by reťazec Y mal výrazne vyššie hodnotenie.

Rouge-S hľadá zhodné bigramy medzi generovaným a referenčným textom s tým, že medzi dvoma unigramami môžu byť iné, ktoré sa vynechajú. Dôležitý parameter pri tejto metrike je, aká veľká môže byť maximálna medzera v zhodných bigramoch. Výhoda oproti BLEU je, že nepotrebuje neprerušenú postupnosť, ale stále zachytí slovosled.

Táto metrika je integrovaná cez rozhranie knižnice evaluate od spoločnosti HuggingFace. Modul je popísaný na stránkach HuggingFace spaces[5]. Evaluate integruje Rouge vyvíjané spoločnosťou Google Research, repozitár implementácie je dostupný na Githube[6]. V tejto knižnici sú implementované len metriky pre Rouge-1, Rouge-2, Rouge-L a Rouge-Lsum. Evaluáciu je možné špecifikovať nasledovnými parametrami:

- `rouge_types` – Špecifikuje, ktoré skóre majú byť vypočitané. Východzia hodnota je list: Rouge-1, Rouge-2, Rouge-L, Rouge-Lsum.
- `use_stemmer` – Použitie vstavaných metód na odseknutie prípon. Východzia hodnota `False`.
- `use_aggregator` – Ak je nastavená na `True`, skóre sa agreguje do jedného, inak vráti skóre pre každú vzorku. Východzia hodnota je `True`.

Túto metriku zapúzdruje trieda `RougeEvaluator`, implementovaná je v súbore `metrics.py`.

Hodnotenie podobnosti textu na základe podobnosti dĺžky

Generovaná dokumentácia by mala mať približne rovnakú dĺžku ako referencie. Ostatné metriky pri hodnotení už z princípu zarátajú do hodnotenia aj pomer dĺžok, chcel som ale vedieť exaktne ohodnotiť a penalizovať príliš krátke alebo dlhé predikcie.

V module `metrics.py` som implementoval triedu `LengthEvaluator`. Táto trieda hodnotí dĺžku generovaného textu voči referencii nasledovne. Pre každú funkciu sú dva referenčné docstringy, Y_1 a Y_2, a model vygeneruje jeden docstring Z. Z toho sa vypočítajú medze dĺžky vygenerovaného docstringu nasledovne:

$$len\_interval\_start = min(len(Y_1), len(Y_2)) * low\_length\_threshold \quad (4.16)$$

$$len\_interval\_end = max(len(Y_2), len(Y_2)) * upper\_length\_threshold \quad (4.17)$$

Podľa toho sa potom pre daný docstring vypočíta skóre 4.18.

$$Score = \begin{cases} 1 & len(Z) \in \langle len\_interval\_start, len\_interval\_end \rangle \\ 1 - penalty & len(Z) \notin \langle len\_interval\_start, len\_interval\_end \rangle \end{cases} \quad (4.18)$$

Skóre sa pre každý docstring vypočíta samostatne. Hlavné skóre je aritmetický priemer, ako rozšírenie však modul vypočíta aj rozptyl.

Zo vzťahov popisujúcich skórovanie vyplývajú tri parametre, ktorými je možné konfigurovať hodnotenie:

[5]Repozitár Rouge metriky na huggingFace: https://huggingface.co/spaces/evaluate-metric/rouge

[6]Repozitár implementácie Rouge Metriky: https://github.com/google-research/google-research/tree/master/rouge

- `low_length_threshold` – Činiteľ pre úpravu dolnej hranice. Nastavený na 0.7.
- `upper_length_threshold` – Činiteľ pre úpravu hornej hranice. Nastavený na 1.3.
- `penalty` – Penalizácia. Nastavená na 0.5.

Hodnotenie syntaxe dokumentácie pomocou gramatiky

Vygenerovaná dokumentácia je takmer nepoužiteľná ak, nedodržiava stanovenú konvenciu (syntax). V module `metrics.py` som implementoval triedu `GrammarEvaluator`. Hodnotenie funguje nasledovne. Evaluator sa pokúsi spracovať danou gramatikou vygenerovaný docstring. Ak je docstring syntakticky správny podľa gramatiky, tak udelí skóre 1, inak udelí skóre $= 1 - penalty$. Takto sa ohodnotí každý docstring, hlavné skóre je aritmetický priemer, ako rozšírenie modul vypočíta aj rozptyl.

Táto metrika má dva parametre:

- `penalty` – Penalizácia. Nastavená na 0.5.
- `grammar_file` – Súbor popisujúci gramatiku. Gramatika je v prílohe A.

Sémantické porovnanie podobnosti vzoru a predikcií

Predstavené metriky Meteor, SacreBLEU a Rouge hodnotia generovaný text na základe hľadania rovnakých n-gramov v texte. Avšak nerozumejú textu ako takému, nezachytávajú sémantiku. Existujú však spôsoby, ako sémanticky porovnávať texty.

Robustný a dobre preskúmaný spôsob spočíva v transformovaní vstupu na n-dimenzionálny vektor, takzvaný **embedding**. Embedding efektívne kóduje sémantiku vstupu do daného vektora. Vstup je teda kódovaný ako bod v n-dimenzionálnom priestore. Táto skutočnosť sa dá využiť, pretože sémanticky rovnaké vstupy budú v priestore bližšie ako vstupy ktoré majú rozdielny význam. Teda vzdialenosť embeddingov je priamo úmerná tomu ako veľmi sú si významovo podobné [13].

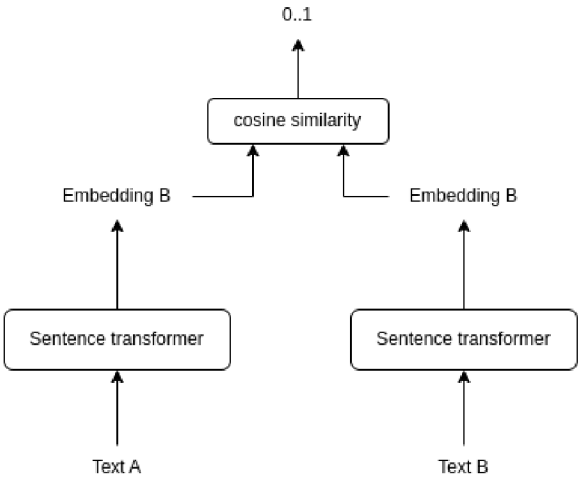
V praxi sa tento princíp implementuje aj pomocou autoenkodérových modelov Transformer, ktoré vyextrahujú embeddingy. Embedingy potom porovnáme napríklad pomocou kosínusovej vzdialenosti [12], principiálne je to naznačené na obrázku 4.1. Sentence transformers, na obrázku 4.1, sú modely vyvinuté špeciálne na extrakciu embeddingov z textu.

Trieda implementujúca túto metriku je `EmbeddingSimilarityEvaluator` v module `metrics.py`. Využíva knižnicu `sentence-transformers`[7]. Princíp fungovania je nasledovný. Najprv sa vyextrahujú všetky embeddingy z referenčných a generovaných textov. Potom sa matica s embeddingami generovaných docstringov upraví tak, aby zodpovedala dimenzii referenčných embeddingov. Inak povedané, zvýši sa dimenzia nakopírovaním každého riadku, pretože pre každý docstring existujú dve referencie. Následne sa na daných maticiach vypočíta kosínusová vzdialenosť. Zo všetkých vzdialeností sa nakoniec vypočíta aritmetický priemer a rozptyl.

Táto trieda má nasledujúce konfigurovateľné parametre:

- `n_workers` – v prípade použitia CPU, koľko jadier použiť, východzie 1.
- `device` – Aké zariadenie použiť na výpočet(GPU, CPU), východzie CPU.
- `model_name` – Aký model použiť na inferenciu, použitý all-MiniLM-L6-v2.

[7] Oficiálna stránka sentence-transformers: https://sbert.net/index.html.



Obrázek 4.1: Principiálne fungovanie sémantického porovnávania dvoch textov.

4.2 Architektúra spôsobu merania

Architektúru nástroja na kvantitatívne hodnotenie výkonu nielen adaptovaných modelov som navrhol tak aby podliehala nasledovným kritériám.

- Možnosť konfigurácie pomocou súboru s nastaveniami.
- Modulárnosť, možnosť jednoduchého pridania metrík.
- Možnosť automaticky vytvárať správy s vyhodnotením.

Konceptuálny návrh implementovaného riešenia je načrtnutý na obrázku 4.2.

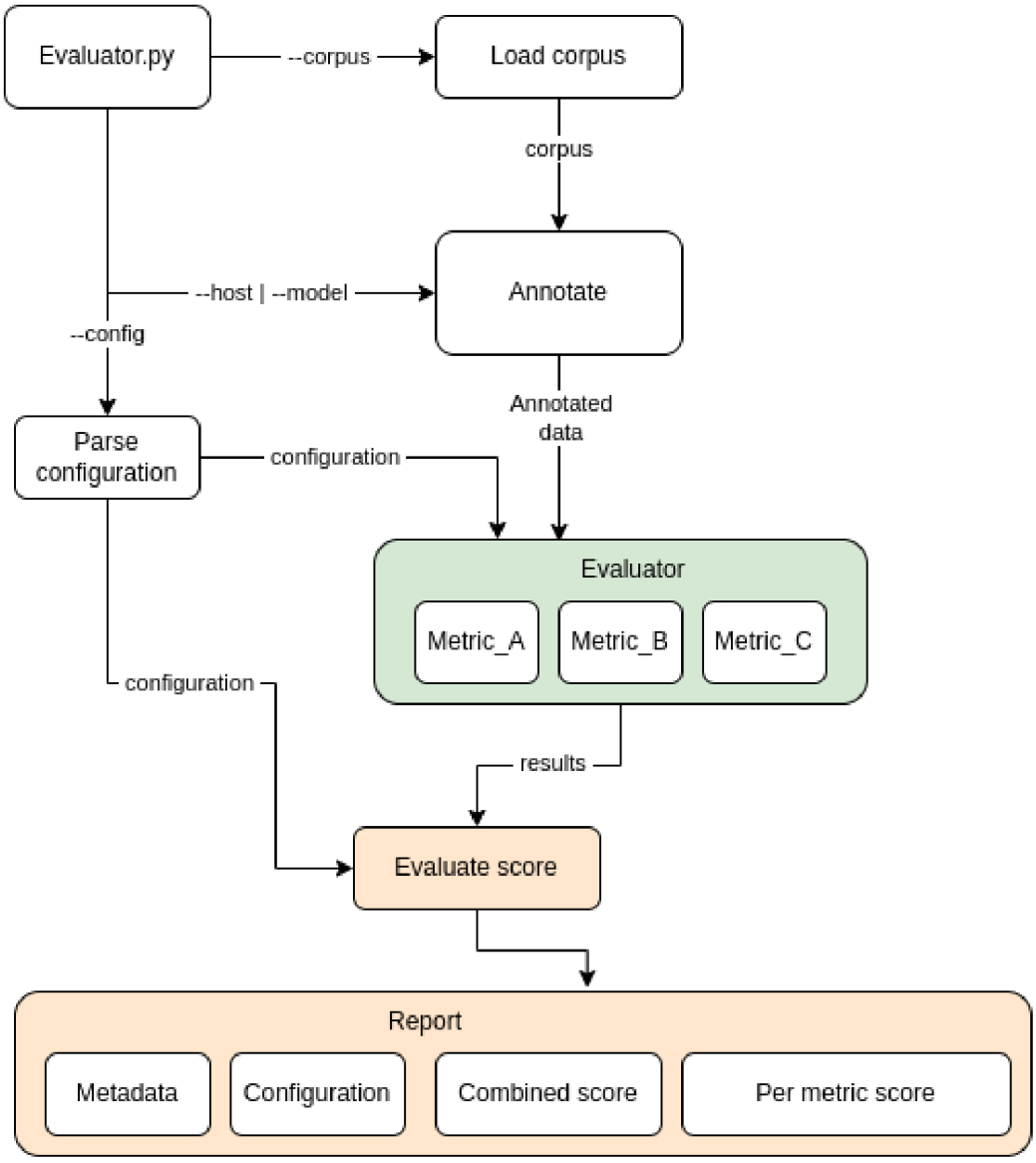
Program na evaluáciu spolu s jeho modulmi a príslušenstvom je v zložke projektu `Evaluation`. V nasledujúcich odstavcoch sú postupne popísané jednotlivé bločky v schéme, ich princíp činnosti a ako na seba nadväzujú.

Evaluator.py je vstupný program, spracováva argumenty z príkazového riadku:

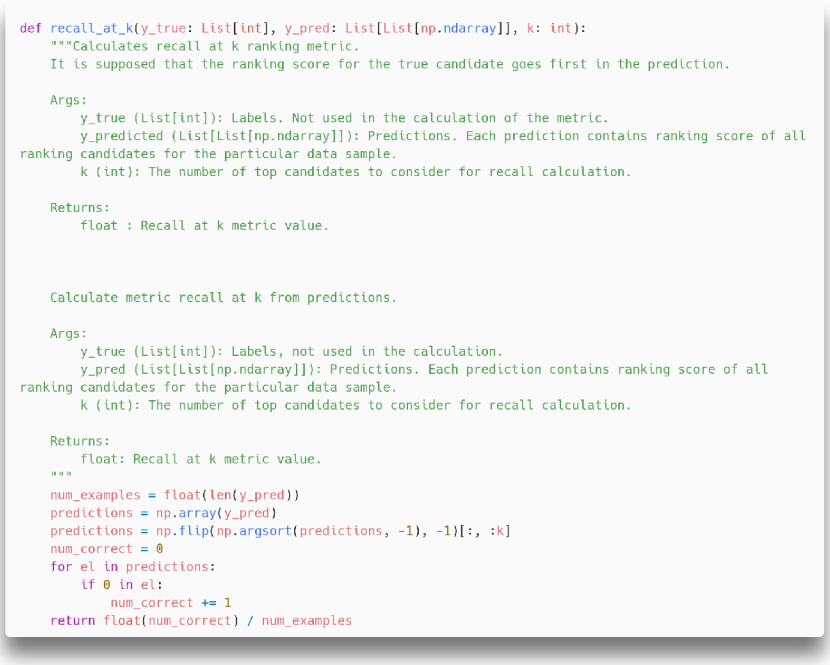
- `host` alebo `model` – Entita poskytujúca generovanie dokumentácie k zadanej funkcii
- `corpus` – Dataset obsahujúci python funkcie s ukážkovými anotáciami.
- `config` – Konfigurácia v json formáte.
- `n_workers` – Niektoré časti programu môžu pracovať vo viacerých vláknach, týmto je možné nastaviť presný počet, východzia hodnota je 1.

Load corpus je implementovaný v module `corpus_processor.py`, načíta korpus, dataset, a spracuje ho do `pandas.Dataframe`. Ukážková položka v datasete je na obrázku 4.3. Funkcia je anotovaná dvoma docstringami, oddelenými troma prázdnymi riadkami. Pri načítavaní sa pomocou knižnice `ast`[8] oddelí dokumentácia od funkcie. Z funkcie sa odstráni dokumentácia tak, aby bola vhodná pre generátor.

[8]dokumentácia dostupná na: https://docs.python.org/3/library/ast.html



Obrázek 4.2: Konceptuálna schéma programu na hodnotenie modelov



```python
def recall_at_k(y_true: List[int], y_pred: List[List[np.ndarray]], k: int):
    """Calculates recall at k ranking metric.
    It is supposed that the ranking score for the true candidate goes first in the prediction.

    Args:
        y_true (List[int]): Labels. Not used in the calculation of the metric.
        y_predicted (List[List[np.ndarray]]): Predictions. Each prediction contains ranking score of all
ranking candidates for the particular data sample.
        k (int): The number of top candidates to consider for recall calculation.

    Returns:
        float : Recall at k metric value.



    Calculate metric recall at k from predictions.

    Args:
        y_true (List[int]): Labels, not used in the calculation.
        y_pred (List[List[np.ndarray]]): Predictions. Each prediction contains ranking score of all
ranking candidates for the particular data sample.
        k (int): The number of top candidates to consider for recall calculation.

    Returns:
        float: Recall at k metric value.
    """
    num_examples = float(len(y_pred))
    predictions = np.array(y_pred)
    predictions = np.flip(np.argsort(predictions, -1), -1)[:, :k]
    num_correct = 0
    for el in predictions:
        if 0 in el:
            num_correct += 1
    return float(num_correct) / num_examples
```

Obrázek 4.3: Ukážková funkcia z evaluačného korpusu. Zdroj: dataset Code_Search_Net.

Annotate v schéme popisuje časť, ktorá funkciám na vstupe vygeneruje dokumentáciu. Je implementovaná v súbore `code_annotation.py`. Pre väčšiu flexibilitu je možné použiť na anotovanie priamo model, jeho štruktúra je popísaná v kapitole 5.1 alebo sa môže dotazovať na server, jeho rozhranie je definované v kapitole 5.3.1. Trieda pre anotovanie dotazovaním na http server je implementovaná multivláknovo. Dataset sa rozdelí medzi vlákna a každé vlákno anotuje len svoju časť, na konci sa časti spoja.

Parse configuration spracuje konfiguráciu. Konfigurácia použitá na hodnotenie modelov je v súbore `configuration.json`, v práci zobrazená v prílohe na obrázku B.1. Položka `evaluators` obsahuje pole metrík s ich konfiguráciou. Jedna položka, zobrazená na obrázku 4.4, obsahuje názov triedy implementujúcej danú metriku a parametre špecifické pre danú metriku. Podľa konfigurácie sa inštanciuje trieda s danými parametrami v evaluátore.

Evaluator je trieda zapúzdrujúca metriky, je implementovaná v súbore `metrics.py` spolu s triedami implementujúcimi vybrané metriky. Jednotlivé metriky sú triedy implementujúce rozhranie `MetricEvalutorI`. Evaluátor postupne nad anotovaným datasetom vyhodnocuje všetky inicializované triedy, pričom ich skóre vkladá do kontajnera `EvaluateScore`.

Evaluate score reprezentuje triedu, ktorá zbiera všetky výsledky z merania, konfiguráciu a vytvorí správu. Je implementovaný triedou `EvaluateScore`, ktorá je v súbore `metrics.py`.

Report symbolizuje správu vo formáte markdown. Táto správa obsahuje:

- **Metadata** – dátum a čas merania.
- **Configuration** – Konfiguráciu podľa, ktorej sa meralo.
- **Combined score** – Kombinované skóre zo všetkých metrík, priemer, rozptyl, produkt a suma.

Obrázek 4.4: Ukážka konfigurácie jednej metriky evaluátora.

- **Per metric Score** – Skóre jednotlivých metrík, každá metrika môže mať rozličné položky, ktoré budú v tejto časti napísané.

4.3 Vyhodnotenie modelov

V tejto sekcii sa využije vyvinutý nástroj na hodnotenie kvality modelov. Vyhodnotia sa nielen adaptované modely, ale pre porovnanie aj bežne dostupné modely, gpt-3.5-turbo-0125 a llama-3-8b. Porovnanie umožní objektívne nahliadnuť na schopnosť generovať dokumentáciu adaptovaných rádovo menších modelov.

4.3.1 Vyhodnotenie finetuningu modelu Salesforce/codet5p-220m

Adaptácia jedného z najmenších modelov z kolekcie `Salesforce/codet5p` s 220 miliónmi parametrov priniesla úspech. Subjektívne môžem zhodnotiť, že schopnosť generovať dokumentáciu aj napriek malej veľkosti je dostatočná.

Model bol trénovaný v precíznosti `float32` na 20-tisíc dátach. Z experimentov vyplynulo, že už pri zmenšenom korpuse dosahuje model dobré výsledky a výrazne to skracuje čas trénovania.

Z ukážky na obrázku 4.5 vidno, že model dobre pomenoval parametre a dokumentácia sémanticky sedí s kódom. Ale chýba popis možných výnimiek a dátových typov argumentov.

Pri evaluácii model dosiahol celkové skóre `0.5908`. Detailné hodnotenie je v prílohe C.1. Z detailného hodnotenia je vidieť nasledovné skutočnosti:

- Model nie vždy dodržiava referenčnú dĺžku textu, niekedy generuje kratšie texty ako by mal.
- Dodržiava konvencie štýlu dokumentácie.
- Má dostatočnú zhodu podľa metrík porovnávajúcich n-gramy.
- Sémanticky sa generované texty vo veľkej miere zhodujú s referenčnými.

```
def parse_deckspawn_metainfo(protobuf: bytes, version: int) -> dict:
  """Parse a DeckSpawnProto message into a dictionary.

    Args:
        protobuf: The DeckSpawnProto message to parse.
        version: The version of the DeckSpawnProto to parse.

    Returns:
        A dictionary containing the parsed DeckSpawnProto message.
    """
    deck = DeckSpawnProto()
    deck.ParseFromString(protobuf)
    error = {'error': 'Deck ({deck}) metainfo incomplete, deck must have a name.'.format(deck=deck.name)}
    if deck.name == '':
        raise InvalidDeckMetainfo(error)
    if deck.version != version:
        raise InvalidDeckVersion({'error', 'Deck version mismatch.'})
    return {'version': deck.version, 'name': deck.name, 'issue_mode': deck.issue_mode,
'number_of_decimals': deck.number_of_decimals, 'asset_specific_data': deck.asset_specific_data}
```

Obrázek 4.5: Ukážka funkcie anotovanej modelom `codet5p-220m-docstring`.

Adaptovaný model je pomenovaný `codet5p-220m-docstring`, dostupný na huggingface hub [9].

4.3.2 Vyhodnotenie finetuningu modelu Salesforce/codet5p-770m

Pre zhodnotenie vplyvu veľkosti modelu na kvalitu generovaného textu bol adaptovaný aj väčší model `Salesforce/codet5p-770m`.

Model bol trénovaný v precíznosti `float16` na 20-tisíc dátach. S podobnými parametrami ako 220 miliónový model.

Pri evaluácii dosiahol model celkové skóre `0.5978`, čo je malé zlepšenie oproti 220 miliónovému modelu. Detailné hodnotenie je v prílohe C.2. Z detailného hodnotenia môžeme porovnať s menším modelom:

- Lepšie dodržiava dĺžku referenčných dát.
- Častejšie nedodržiava syntax konvencie.
- Dosahuje lepšie skóre v metrikách, ktoré porovnávajú n-gramy.
- Sémantické hodnotenie je o stotinu menšie, čo je zanedbateľné.

Na obrázku 4.6 vidno reálne zlepšenie anotovania, preto by sa dalo očakávať väčšie skóre evaluácie. Zlepšenie sa však neprejavilo na celom datasete, preto rozdiely v kvantitatívnom meraní nie sú až také veľké.

Adaptovaný model je pomenovaný `codet5p-770m-docstring`, dostupný na huggingface hub[10].

[9] https://huggingface.co/juraj-juraj/codet5-220m-docstring
[10] https://huggingface.co/juraj-juraj/codet5p-770m-docstring

```python
def parse_deckspawn_metainfo(protobuf: bytes, version: int) -> dict:
    """Parses a DeckSpawn metainfo from a given protobuf string and version.

    Args:
        protobuf (bytes): The protobuf string containing the DeckSpawn metainfo.
        version (int): The version of the DeckSpawn metainfo to parse.

    Returns:
        dict: A dictionary containing the parsed information about the Deck, including version, name, issue mode,
number of decimals, and asset specific data.

    Raises:
        InvalidDeckMetainfo: If the Deck metainfo is incomplete, or if the Deck name is empty.
        InvalidDeckVersion: If the Deck version does not match.
    """
    deck = DeckSpawnProto()
    deck.ParseFromString(protobuf)
    error = {'error': 'Deck ({deck}) metainfo incomplete, deck must have a name.'.format(deck=deck.name)}
    if deck.name == '':
        raise InvalidDeckMetainfo(error)
    if deck.version != version:
        raise InvalidDeckVersion({'error', 'Deck version mismatch.'})
    return {'version': deck.version, 'name': deck.name, 'issue_mode': deck.issue_mode, 'number_of_decimals':
deck.number_of_decimals, 'asset_specific_data': deck.asset_specific_data}
```

Obrázek 4.6: Ukážka funkcie anotovanej modelom `codet5p-770m-docstring`.

4.3.3 Vyhodnotenie finetuningu modelu allenai/led-base-16384

Finetuning modelu `allenai/led-base-16384` bol náročnejší ako predchádzajúce dva modely a priniesol komplikácie. Nedarilo sa ho natrénovať tak, aby dosahoval použiteľné výsledky. Skúšal som adaptáciu pomocou vlastného skriptu, čo nedosahovalo želané výsledky. Následne som skúsil použiť skript na finetuning od samotného autora modelu, no model dosahoval podobné výsledky.

Trénovanie prebiehalo na precíznosti `float16` a na 30000 funkciách.

Z ukážky na obrázku 4.7 vidno, že model nezdokumentoval všetky argumenty funkcie ani možné výnimky.

Pri evaluácii model dosiahol celkové skóre `0.4742`. Detailné vyhodnotenie je v prílohe C.3. Z vyhodnotenia vyplývajú nasledovné závery:

- Model do istej miery dodržiava dĺžky generovaných komentárov.
- Model vo veľa prípadoch nedodržiava konvencie štýlu.
- Hodnotenie podľa zhody n-gramov je podpriemerné.
- Sémantické porovnanie je tiež podpriemerné.

Ručná kontrola vygenerovaných dokumentácií podporuje výsledky merania. Viackrát model nedodržiaval konvenciu dokumentácie. Taktiež niektoré docstringy sa textovo prelínali so vstupom, vstup sa objavoval na výstupe.

Adaptovaný model som pomenoval `led_base_16384_docstring`, je dostupný na huggingface hub [11].

[11] https://huggingface.co/juraj-juraj/led_base_16384_docstring

```python
def parse_deckspawn_metainfo(protobuf: bytes, version: int) -> dict:
    """Generate docstring to python function:

    Parse a deck metainfo and return a dictionary containing information about the deck, including version, name,
issue mode, number of decimals, and asset specific data.

    Args:
        protobuf (bytes): The path to the deck file to be parsed. Defaults to bytes, which is the version of the deck
to be used for parsing.

    Returns:
      dict: A dictionary containing the deck information including the deck version and issue mode for the deck.
    """
    deck = DeckSpawnProto()
    deck.ParseFromString(protobuf)
    error = {'error': 'Deck ({deck}) metainfo incomplete, deck must have a name.'.format(deck=deck.name)}
    if deck.name == '':
        raise InvalidDeckMetainfo(error)
    if deck.version != version:
        raise InvalidDeckVersion({'error', 'Deck version mismatch.'})
    return {'version': deck.version, 'name': deck.name, 'issue_mode': deck.issue_mode, 'number_of_decimals':
deck.number_of_decimals, 'asset_specific_data': deck.asset_specific_data}
```

Obrázek 4.7: Ukážka anotovanej funkcie pomocou modelu `led_base_16384_docstring`.

4.3.4 Vyhodnotenie konkurenčného modelu gpt-3.5-turbo-0125

Ako konkurečné riešenie na porovnanie schopností je vhodný model `gpt-3.5-turbo-0125` od spoločnosti OpenAI. Tento model bol meraný cez verejné API. Parametre tohto modelu nie sú verejne známe. Podľa dostupných informácií má model 20 miliárd parametrov [14]. Tento model je definovaný cez vlastné rozhranie predstavené v časti 5.1.

Z obrázku 4.8 je vidno, že model dodržal syntax štýlu a správne popísal všetky parametre a výnimky,

Pri evaluácii dosiahol skóre `0.5554`, detailné skóre je v prílohe C.4. Z detailných výsledkov sa dá vyhodnotiť:

- Model dodržiava dĺžky podľa referencií.
- Model často nedodržiava syntax konvencie.
- Skóre podľa podobnosti n-gramov je na dobrej úrovni.
- Sémanticky je generovaná dokumentácia dostatočne podobná referenčnej.

```python
def parse_deckspawn_metainfo(protobuf: bytes, version: int) -> dict:
    """Parse the metainfo of a deck from a Protocol Buffers encoded message.

    Args:
        protobuf (bytes): The Protocol Buffers encoded message containing the deck metainfo.
        version (int): The expected version of the deck metainfo.

    Raises:
        InvalidDeckMetainfo: If the deck metainfo is incomplete and missing a name.
        InvalidDeckVersion: If the deck version does not match the expected version.

    Returns:
        dict: A dictionary containing the parsed deck metainfo, including version, name, issue mode,
number of decimals, and asset specific data."""
    deck = DeckSpawnProto()
    deck.ParseFromString(protobuf)
    error = {'error': 'Deck ({deck}) metainfo incomplete, deck must have a name.'.format(deck=deck.name)}
    if deck.name == '':
        raise InvalidDeckMetainfo(error)
    if deck.version != version:
        raise InvalidDeckVersion({'error', 'Deck version mismatch.'})
    return {'version': deck.version, 'name': deck.name, 'issue_mode': deck.issue_mode,
'number_of_decimals': deck.number_of_decimals, 'asset_specific_data': deck.asset_specific_data}
```

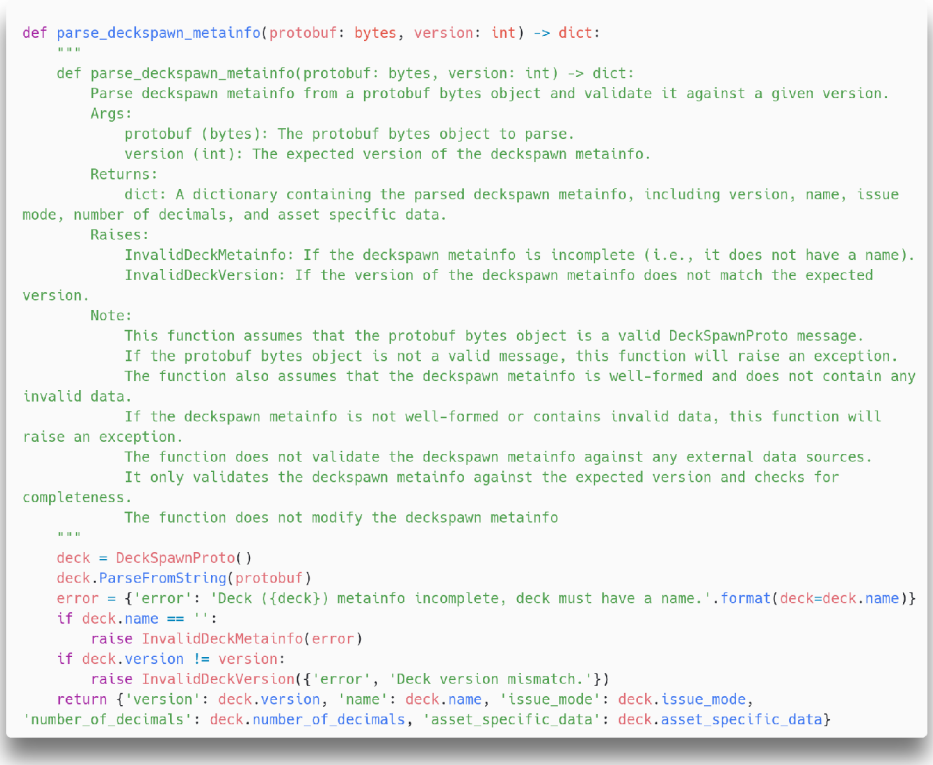
Obrázek 4.8: Ukážka funkcie anotovanej pomocou modelu `gpt-3.5-turbo-0125`.

4.3.5 Vyhodnotenie konkurenčného modelu llama-3-8b-instruct

Ako vhodný kandidát na porovnanie modelov je taktiež vhodný model `llama-3-8b-instruct`, ktorý cez verejné API ponúka spoločnosť Predibase. Tento model je podľa popisu okrem iného vhodný aj na generovanie dokumentácie. Model je použitý cez vlastné rozhranie definované v časti 5.1.

Model dosiahol skóre `0.3663`. Toto skóre je oproti iným modelom podpriemerné, pretože model generuje zbytočne dlhé dokumentácie. Takto dlhé dokumentácie nie sú nutné, pretože popisujú zrejmé veci a popis zachádza až k popisu implementácie. Detailné skóre je v prílohe C.5. Z detailných výsledkov sú zrejmé nasledovné skutočnosti:

- Model generuje dokumentáciu inej dĺžky ako je referenčná.
- Model nedodržuje syntaktickú konvenciu štýlu.
- Následkom nedodržovania dĺžky majú metriky na porovnávanie n-gramov zlé skóre.
- Sémantická presnosť je na priemernej úrovni.



```python
def parse_deckspawn_metainfo(protobuf: bytes, version: int) -> dict:
    """
    def parse_deckspawn_metainfo(protobuf: bytes, version: int) -> dict:
        Parse deckspawn metainfo from a protobuf bytes object and validate it against a given version.
        Args:
            protobuf (bytes): The protobuf bytes object to parse.
            version (int): The expected version of the deckspawn metainfo.
        Returns:
            dict: A dictionary containing the parsed deckspawn metainfo, including version, name, issue
mode, number of decimals, and asset specific data.
        Raises:
            InvalidDeckMetainfo: If the deckspawn metainfo is incomplete (i.e., it does not have a name).
            InvalidDeckVersion: If the version of the deckspawn metainfo does not match the expected
version.
        Note:
            This function assumes that the protobuf bytes object is a valid DeckSpawnProto message.
            If the protobuf bytes object is not a valid message, this function will raise an exception.
            The function also assumes that the deckspawn metainfo is well-formed and does not contain any
invalid data.
            If the deckspawn metainfo is not well-formed or contains invalid data, this function will
raise an exception.
            The function does not validate the deckspawn metainfo against any external data sources.
            It only validates the deckspawn metainfo against the expected version and checks for
completeness.
            The function does not modify the deckspawn metainfo
    """
    deck = DeckSpawnProto()
    deck.ParseFromString(protobuf)
    error = {'error': 'Deck ({deck}) metainfo incomplete, deck must have a name.'.format(deck=deck.name)}
    if deck.name == '':
        raise InvalidDeckMetainfo(error)
    if deck.version != version:
        raise InvalidDeckVersion({'error', 'Deck version mismatch.'})
    return {'version': deck.version, 'name': deck.name, 'issue_mode': deck.issue_mode,
'number_of_decimals': deck.number_of_decimals, 'asset_specific_data': deck.asset_specific_data}
```

Obrázek 4.9: Ukážka funkcie anotovanej modelom `llama-3-8b-instruct` od spoločnosti Predibase.

Kapitola 5

Implementácia nástroja poskytujúceho intuitívne použitie modelov

Adaptovanie generatívneho modelu na špecifickú úlohu rieši len časť problému. Akokoľvek dobre navrhnutý a výkonný model bez vhodného API alebo UI je nepoužiteľný. Preto je dôležité navrhnúť aj systém, ktorý bude umožňovať využívanie modelu.

V tejto kapitole sa pozornosť prenesie z vývoja modelu na jeho nasadenie. Využijú sa už predstavené modely tak, aby prinášali pridanú hodnotu aj bežným užívateľom. Ďalšie prinesú poznatky ako používať modely cez príkazový riadok či cez HTTP komunikáciu a ako je možné rozšíriť funkcionalitu týchto nástrojov o ďalšie modely.

Implementácia tohto celku je v zložke projektu `server`, v texte tejto časti budú už len odkazy na konkrétne zdrojové súbory, ktoré sú len v rámci tejto zložky.

5.1 Architektúra zapúzdrenia modelu

Hlavným cieľom integrácie modelu bolo vytvoriť entitu s jednoduchým rozhraním fungujúcu nezávisle na vnútornej logike. Navrhnuté riešenie preto poskytuje maximálnu voľnosť čo sa týka vnútornej funkcionality modelu.

Základná štruktúra zapúzdrenia modelu je nasledovná. Je zložka v súborovom systéme počítača. Táto zložka musí obsahovať skript `__init__.py` a `model.py`. Init súbor je prázdny, z dôvodu aby Python interpret rozoznal zložku ako platný modul. `Model.py` obsahuje logiku modelu. Implementuje rozhranie `ModelI` 5.1, definované je v súbore `model_loader.py` .

Konštruktor je definovaný tak, aby každý model prijímal ľubovoľné argumenty. V implementácii je možné potom genericky inštanciovať modely s tým, že modely prijímajú aj argumenty, ktoré nepotrebujú, ale iné modely by ich potrebovali(napr. možnosť použitia grafickej karty). V nasledujúcich častiach sú predstavené modely, pričom niektoré vedia využiť grafickú kartu a iné modely takú možnosť nemajú a tento argument ignorujú.

Metóda `generate` je určená na generovanie predikcií. Jej vstup je jedna definícia funkcie aj s prototypom. Výstup je vygenerovaný docstring. To, ako konkrétne model vygeneruje dokumentáciu, je čisto na implementácii. Teda model sám nemusí generovať, ako je ukázané v nasledujúcich častiach, model môže fungovať ako proxy na model poskytovaný cez API na internete alebo môže len generovať statický popis pre ladiace účely.

Nasledujúce časti sú venované implementovaným modelom.

```python
class ModelI(Protocol):

    def __init__(self, *args, **kwargs) -> None: ...
    def generate(self, code: str) -> str: ...
```

Obrázek 5.1: Python rozhranie modelu

ct2_codet5p_220m_docstring a ct2_codet5p_770m_docstring

Integrácia adaptovaných modelov codet5p. Model je spustený pomocou `CTranslate2`. Zložka obsahuje súbory modelov prevedené do formátu vhodného pre runtime CTranslate2 v zložke `data`. Konfigurácia, v súbore `config.json`, špecifikuje meno modelu, prefix dotazu a kvantizáciu pre beh na procesore a na grafickej karte. Samotný model sa načíta pri konštrukcii z dátovej zložky a počas celého behu je v operačnej alebo grafickej pamäti.

led_base_16384_docstring

Zapúzdruje adaptovaný model `led_base_16384_docstring`. Samotný model neobsahuje binárne dáta, tie sa pri spustení stiahnu z internetu. Ako behové prostredie používa knižnicu `transformers`. Konfigurácia obsahuje názov modelu na huggingface hub a prefix dotazu pre generovanie.

gpt_online_generator

Tento model funguje len ako proxy na OpenAI servery poskytujúce model `gpt-3.5-turbo-0125`. Využíva knižnicu openai na jednoduchú a spoľahlivú komunikáciu. Model má pevne nastavenú rolu a dotaz je makro, do ktorého sa doplní text funkcie.

Rola pre gpt model je nastavená nasledovne:
`You are a helpful assistant to create google-style docstrings for python functions.`
Dotaz na anotáciu potom vyzeral nasledovne:

```
Create google-style dosctring for function:
<code>
Return only docstring without quotes
```

Kde `<code>` je nahradené za konkrétnu funkciu.

predibase_model

Model funguje ako proxy na servery spoločnosti Predibase. Táto spoločnosť ponúka API rozhranie pre modely trénované na riešenie úloh v oblasti programovania. Konfigurácia obsahuje názov práve používaného modelu z repertoára. Nakonfigurovaný a odskúšaný je model `llama-3-8b-instruct`.

Na generovanie dokumentácie používa model nasledovnú šablónu:

```
Write an appropriate docstring for the following Python function.
Return only generated docstring without quotes.
#Function:
<code>
Generated docstring:
```

Tento dotaz vychádza z ukážky v blogu[1], je upravený tak, aby model vracal len docstring. V makre sa nahradí `<code>` za vloženú funkciu.

5.2 Pridanie dokumentácie do existujúceho kódu

Docstring dokumentácia je súčasť abstraktného syntaktického stromu Python programu. Python knižnica `ast` zdrojový kód spracuje do syntaktického stromu. Táto knižnica poskytuje nástroj ako z funkcií, metód alebo modulov extrahovať docstringy. Docstring je vždy prvý uzol v tele a má dátovy typ `Str` alebo `Constant` s vnútorným dátovým typom `str`.

Vzhľadom na to, že `ast` neposkytuje funkcionalitu pre pridávanie docstringov, požadovanú funkcionalitu som vyvinul v module `docstring_transformer.py`. Názov funkcie je `set_docstring` a funkcia pridá ako prvý prvok tela funkcie výraz s konštantou obsahujúcou požadovaný text. V prípade ak už funkcia má dokumentáciu, je možné ju prepísať ak to užívateľ dovolí prepínačom `overwrite`.

5.3 Rozhranie pre anotáciu cez http rozhranie

Pre čo najlepšiu dostupnosť natrénovaných modelov som implementoval http server poskytujúci rozhranie pre anotáciu zdrojových súborov. Návrh servera podliehal požiadavkám na efektívne poskytovanie možností modelov a zároveň aby maximálne využíval dostupný výpočtový výkon.

Server dokáže načítať a využiť modely popísané v časti 5.1. V ďalších častiach bude popísané akým spôsobom sú modely inštanciované v systéme.

Framework pre budovanie http servera

Výber správneho frameworku je dôležitý krok v procese vývoja servera pre aplikácie strojového učenia. FastAPI vyniká ako excelentná voľba. Je to výkonný, moderný a efektívny Python framework, ktorý som zvolil ako ideálnu možnosť.

Jedným z mnohých dôvodov, prečo si vybrať FastAPI, je natívne používanie knižnice `pydantic` pre jednoduché kontrolovanie správnosti prichádzajúcich požiadavkov. Ďalšou charakteristickou črtou FastAPI je jeho jednoduchá syntax, ktorá znižuje zložitosť vývoja a čas ladenia. FastAPI má vstavanú serializáciu, ktorá pomáha konvertovať zložité dátové typy na primitívy json. Umožňuje rýchle overenie vstupu a výstupu, čím sa znižuje priestor pre chyby a zvyšuje sa robustnosť.

[1] https://predibase.com/blog/generate-high-quality-docstring-with-fine-tuned-codellama-7b

5.3.1 Popis API

V nasledujúcich častiach sú popísané endpointy, ktoré ponúka server pre anotáciu kódu a správu.

`POST /annotate_code/` – Endpoint popisuje celý súbor, zdrojový kód. Spracovanie dotazu je blokujúce, teda klient čaká a drží otvorené spojenie kým server neodpovie s hotovou úlohou. Dotaz je definovaný triedou `AnnotateRequest`, ukážka serializovaného dotazu je v objekte 1. Položka `overwrite_docstrings` špecifikuje, či má generátor prepísať už existujúce docstringy v zdrojovom kóde. Endpoint vráti ako odpoveď json s jednou položkou `result`, ktorá obsahuje anotovaný zdrojový kód. V prípade, že je plná fronta požiadaviek, server vráti chybový stav s číslom 429 a správou, že server je vyťažený.

Implementačne sa pri spracovaní požiadavky zadá úloha do fronty s typom `annotate_code`. Viac o implementácii spracovania požiadaviek je v časti 5.3.2.

```
{
    "code": <zdrojový kód>,
    "overwrite_docstrings": False
}
```

Obj 1: Ukážka serializovaného dotazu `AnnotateRequest`.

`POST /generate_docstring/` – Endpoint prijímajúci jednu funkciu, ktorej vygeneruje docstring a vráti len ten. Spracovanie je rovnaké ako pri endpointe `/annotate_code/`, s rozdielom že úloha má typ `generate_docstring`. Dotaz je typu `AnnotateRequest` 1, návratová hodnota je rovnaká ako pri `/annotate_code/`. `Overwrite_docstrings` sa v tomto prípade ignoruje.

`GET /info/` – Endpoint slúži na získanie konfiguračných informácií, zo spusteného servera. Vráti informácie obsiahnuté v aktuálne načítanej a použitej konfigurácii servera. Neposkytuje informácie z konfigurácie načítaného modelu.

5.3.2 Architektúra spracovávania požiadavkov

Server je implementovaný ako FastAPI aplikácia. FastAPI sa stará o smerovanie konkrétnych požiadaviek na správne funkcie, ktoré ich obslúžia a klientovi vráti požadovanú odpoveď. Server funguje vďaka FastAPI asynchrónne, čo je výhodné pre ďalšie spracovanie.

Pri návrhu architektúry som riešil problém zdroja(tj. modelu) s výlučným prístupom. Jeden model môže naraz spracovávať jednu úlohu, pričom jedna úloha môže zabrať rádovo sekundy. Pre účely škálovania a maximálneho využitia hardware je výhodné mať naraz viac modelov a naraz spracovávať viac požiadaviek. Z toho vyvstávajú problémy, ktorému modelu priradiť aktuálnu úlohu, ktorý je voľný a ako priradiť výsledok úlohy správnemu klientovi.

Server stavia na architektúre nezávislých `workerov`, kde je každý nezávislá jednotka s vlastným modelom a prístupom k zdieľanej fronte. Implementácia je ako multivláknová aplikácia založená na Python knižnici `threading`.

Worker je objekt, ktorý zapúzdruje a obsluhuje model. Pri konštrukcii inštanciuje triedu modelu, ktorá je ako parameter konštruktora a uloží si referenciu na zdieľanú frontu. Implementačne je worker špecializácia triedy `threading.Thread` inštanciovaná ako daemon vlákno. Worker potom beží v pozadí, počas celého behu servera a aynchrónne čaká kým sa pridá do fronty úloha. Akonáhle sa objaví úloha, vyberie ju z fronty. Samotná úloha

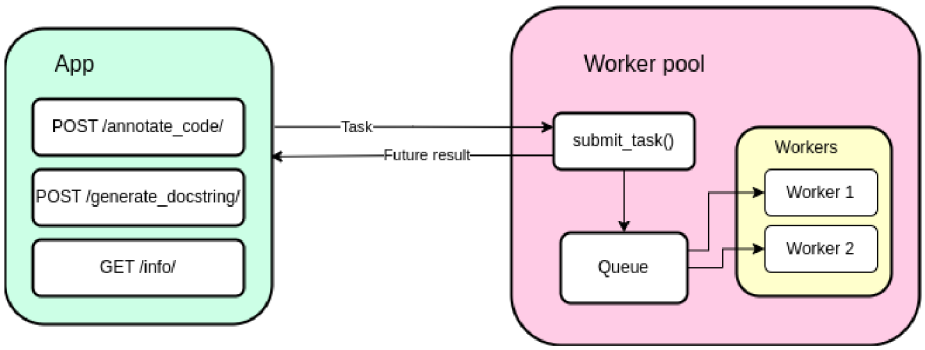
pozostáva z `GeneralTask` a `asyncio.Future`. Podľa typu úlohy, anotácia zdrojového kódu alebo generovanie dokumentácie len ku konkrétnej funkcii, spracuje a s výsledkom notifikuje konkrétny čakajúci endpoint pomocou `asyncio.Future`. Ak počas spracovania nastane neočakávaná chyba, výnimka sa propaguje cez `asyncio.Future`. Implementuje ho trieda `AnnotateWorker` v súbore `server.py`.

Worker pool, obrázok 5.2, obsluhuje zdieľanú frontu a inicializuje workerov pri štarte servera. Pomocou jeho metódy `submit_task` sa vkladajú úlohy do fronty, ktorá je zdieľaná s workermi. Pri zadávaní úlohy sa predpokladá s dvoma možnými výsledkami:

- Fronta má kapacitu na ďalšiu úlohu. V tom prípade metóda vráti objekt `asyncio.Future`, ktorý sa spolu so zadaním úlohy vloží do fronty.
- Kapacita fronty je na maxime, v tom prípade sa vyvolá výnimka informujúca o vyčerpanosti zdroja.

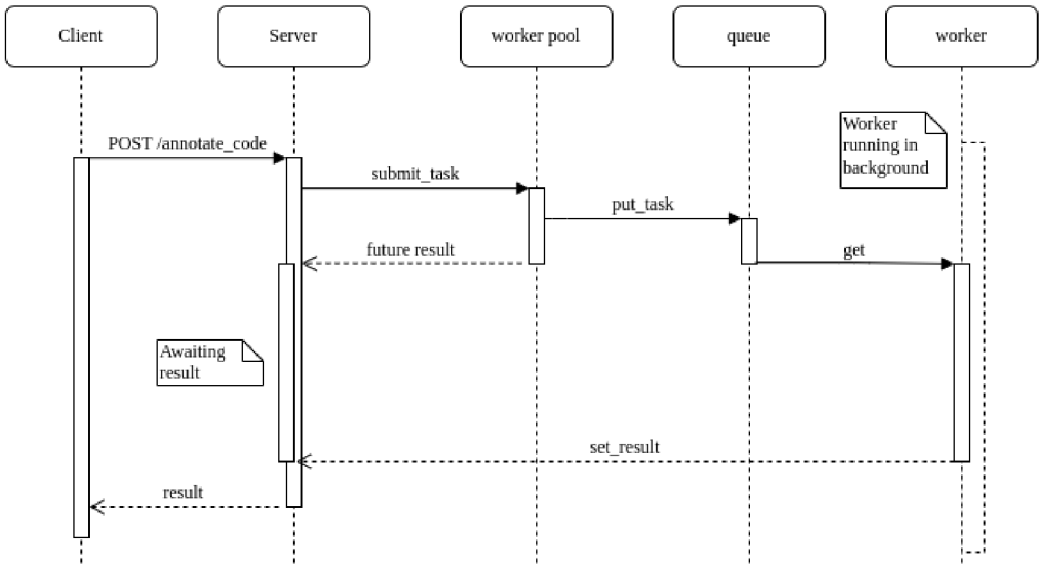
Implementácia fronty v Pythone zaručuje atomické pridávanie a odoberanie, teda operácie s ňou sú bezpečné z hľadiska viacvláknovej aplikácie. Objekt `worker_pool` je dostupný ako globálna premenná.

App, obrázok 5.2, znázorňuje top-level objekt http servera. Je to trieda fastapi frameworku zodpovedná za prijímanie requestov a ich smerovanie. Pod ňu sa registrujú jednotlivé endpointy. Pri prijatí konkrétneho requestu sa z endpointu zadá úloha do worker_poolu a následne sa čaká na výsledok. V prípade neúspechu úlohy sa zachytí vzniknutá výnimka, ktorá sa zašle klientovi.



Obrázek 5.2: Zjednodušený pohľad na architektúru servera.

Obrázok 5.3 názorne popisuje sekvenciu ako sa môže spracovať jeden request od klienta. V popisovanom prípade je jeden worker, ktorý beží v pozadí a do momentu príchodu požiadavku nespracováva žiadne požiadavky.



Obrázek 5.3: Ukážka sekvenčného diagramu spracovania požiadavky od klienta.

Parametre servera sú konfigurovateľné pomocou súboru `server_configuration.json`, meno súbora je pevne stanovené. Konfiguračný súbor obsahuje nasledovné položky:

- `model_dir` – Priečinok obsahujúci model popísaný v časti 5.1.
- `device` – Zariadenie použité na výpočet, môže byť `cpu` alebo `cuda` pre gafické karty od Nvidia.
- `workers` – Počet workerov. Viac workerov môže efektívnejšie využiť hardware. Ich maximálny počet je typicky daný veľkosťou operačnej pamäte.
- `queue_size` – Veľkosť fronty v počte zadaných úloh.
- `log_level` – Úroveň logovania serveru pri behu.

5.3.3 Uživateľské rozhranie pre anotovanie kódu

V rámci systému je popísaný server iba jedna časť riešenia problému. Druhá časť je ako ho využiť. Vďaka jednoduchému API je možné využiť rôzne voľne dostupné nástroje na http komunikáciu, napríklad Postman [2], curl a iné.

Avšak implementoval som aj jednoduchého klienta, ktorý vie z príkazového riadku sprístupniť možnosti servera. Je implementovaný v skripte `client.py`.

Pre rýchle a jednoduché anotovanie jednotlivých súborov som implementoval aj aplikáciu s rozhraním v príkazovom riadku. Táto aplikácia nepotrebuje bežiaci server. Načíta zadaný model, doplní dokumentáciu a súbor uloží do zadaného súbora. Je implementovaná v skripte `cli_app.py`.

[2] https://www.postman.com/

Kapitola 6

Záver

Cieľom tejto bakalárskej práce bolo predovšetkým adaptovať vhodné modely tak, aby k nedokumentovanému zdrojovému kódu v jazyku Python generovali dokumentáciu, a vyvinúť systém na ich využitie na bežnom hardware.

Jednou z prvých výziev bolo nájdenie vhodného datasetu na fine-tuning vybraných modelov. Vytvorený dataset je podmnožina datasetu code_search_net. Zo spomenutého datasetu bola vybratá len časť s funkciami písanými v programovacom jazyku Python. Vylúčením funkcií, ktorých dokumentácia nespĺňala konvenciu Google Style a rozšírením datasetu generovaním dokumentácie pomocou gpt-3.5-turbo-0125 pre nedokumentované funkcie, vznikol dostatočne veľký dataset na finetuning jazykových modelov.

Pomocou pripraveného datasetu boli adaptované modely. Ako vhodné boli vybrané modely postavené na architektúre T5 a Longformer. Z architektúry T5 konkrétne modely upravené spoločnosťou Salesforce, codet5p-220m a codet5p-770m. Z architektúry longformer bol vybraný model predprivanený Allen inštitútom pre AI, konkrétne led_base_16384.

Dôležitým aspektom tejto práce bol vývoj hodnotiaceho nástroja na objektívne posúdenie výkonnosti modelu. Využíva zavedené metriky ako SacreBLEU, Meteor a Rouge, spolu s vlastnými metrikami. Vlastné metriky boli implementované za účelom posúdenia dodržovania konvencie a dĺžky generovaného textu. Hodnotenie dodržiavania konvencie dokumentácie je kontrolované pomocou gramatiky popisujúcej skúmaný štýl. V prípade tejto práce dodržiavanie konvencie Google Style.

Kvantitaívne hodnotenie prinieslo náhľad na výkonnosť modelov a takisto umožnilo porovnanie s dostupnými väčšími modelmi. Ako konkurenčné modely pre porovnanie boli zvolené gpt-3.5-turbo-0125 od OpenAI a llama-3-8b, ktorý ponúka spoločnosť Predibase. Oba konkurenčné modely sú podľa popisu autorov vhodné aj na úlohy týkajúce sa programovania. Na účely hodnotenia boli použité pomocou verejného API.

Oba modely codet5p – 220 aj 770 miliónov parametrov – vykazovali dobré skóre. V úlohe generovania dokumentácie, na ktorú boli adaptované, prekonali oba konkurenčné, o magnitúdu väčšie modely. Úroveň generovanej dokumentácie je na použiteľnej úrovni. Väčší z modelov, codet5p-770m, očakávane prekonáva menší, kde dokáže častejšie odvodiť aj dátové typy a výnimky, ktoré môže funkcia vyvolať. Model led_base_16384 nedosiahol také vysoké skóre a dokumentácia ním generovaná nie je použiteľnej kvality.

Poslednou úlohou bola implementácia servera, ktorý sprístupňuje možnosti modelov cez ľahko použiteľné API. Server je navrhnutý tak, aby horizontálnym škálovaním dokázal maximálne využiť potenciál hardware. Rozšíriteľnosť riešenia podčiarkuje aj možnosť servera pracovať s ľubovoľným modelom navrhovanej architektúry.

Literatura

[1] Banerjee, S. a Lavie, A. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In: *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. Ann Arbor, Michigan: Association for Computational Linguistics, Jun 2005, s. 65–72. Dostupné z: https://www.aclweb.org/anthology/W05-0909.

[2] Beltagy, I., Peters, M. E. a Cohan, A. *Longformer: The Long-Document Transformer*. 2020.

[3] Loper, E. *Automatic API Documentation Generation for Python* [online]. 2008 [cit. 2024-04-18]. Dostupné z: https://epydoc.sourceforge.net/.

[4] Google. *Comments and Docstrings* [online]. 2024 [cit. 2024-04-18]. Dostupné z: https://google.github.io/styleguide/pyguide.html#38-comments-and-docstrings.

[5] Husain, H., Wu, H.-H., Gazit, T., Allamanis, M. a Brockschmidt, M. *CodeSearchNet Challenge: Evaluating the State of Semantic Code Search*. 2020.

[6] Lavie, A. a Agarwal, A. METEOR: An Automatic Metric for MT Evaluation with High Levels of Correlation with Human Judgments. In: Callison Burch, C., Koehn, P., Fordyce, C. S. a Monz, C., ed. *Proceedings of the Second Workshop on Statistical Machine Translation*. Prague, Czech Republic: Association for Computational Linguistics, Jun 2007, s. 228–231. Dostupné z: https://aclanthology.org/W07-0734.

[7] Lin, C.-Y. ROUGE: A Package for Automatic Evaluation of Summaries. In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, červenec 2004, s. 74–81. Dostupné z: https://aclanthology.org/W04-1013.

[8] authors, N. *Style guide* [online]. 2024 [cit. 2024-04-18]. Dostupné z: https://numpydoc.readthedocs.io/en/latest/format.html.

[9] Papineni, K., Roukos, S., Ward, T. a Zhu, W.-J. Bleu: a Method for Automatic Evaluation of Machine Translation. In: Isabelle, P., Charniak, E. a Lin, D., ed. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, JUL 2002, s. 311–318. DOI: 10.3115/1073083.1073135. Dostupné z: https://aclanthology.org/P02-1040.

[10] David Goodger, G. v. R. *PEP 257 – Docstring Conventions* [online]. 2001 [cit. 2024-04-18]. Dostupné z: https://peps.python.org/pep-0257/.

[11] Post, M. A Call for Clarity in Reporting BLEU Scores. In: Bojar, O., Chatterjee, R., Federmann, C., Fishel, M., Graham, Y. et al., ed. *Proceedings of the Third Conference on Machine Translation: Research Papers*. Brussels, Belgium: Association for Computational Linguistics, Oct 2018, s. 186–191. DOI: 10.18653/v1/W18-6319. Dostupné z: https://aclanthology.org/W18-6319.

[12] Sanseviero, O. *Deep Dive into Cross-encoders and Re-ranking* [online]. 2024 [cit. 2024-04-27]. Dostupné z: https://osanseviero.github.io/hackerllama/blog/posts/sentence_embeddings2.

[13] Sanseviero, O. *Everything you wanted to know about sentence embeddings (and maybe a bit more)* [online]. 2024 [cit. 2024-04-27]. Dostupné z: https://osanseviero.github.io/hackerllama/blog/posts/sentence_embeddings.

[14] Singh, M., Cambronero, J., Gulwani, S., Le, V., Negreanu, C. et al. *CodeFusion: A Pre-trained Diffusion Model for Code Generation*. 2023.

[15] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L. et al. *Attention Is All You Need*. 2017.

[16] Wang, Y., Le, H., Gotmare, A. D., Bui, N. D. Q., Li, J. et al. *CodeT5+: Open Code Large Language Models for Code Understanding and Generation*. 2023.

Príloha A

Gramatiky popisujúce štýly dokumentácie

Gramatika popisujúca Google Style dokumentovanie

```
start: LINE+ parameters NEWLINE*

parameters: (args_sec NEWLINE* [returns_sec NEWLINE*] [raises_sec])|
    (args_sec NEWLINE* raises_sec NEWLINE* returns_sec)
raises_sec.1: RAISES_KW NEWLINE+ ARG_PAIR+
returns_sec: RETURNS_KW NEWLINE+ ANYTHING
args_sec: ARGS_KW NEWLINE+ ARG_PAIR+

ARG_PAIR.1: DWORD [" "] [TYPE_SPEC] [" "] ":" /./+ [NEWLINE]
LINE: /./* NEWLINE
LINE_N: /./+ [NEWLINE]
ANYTHING: (/./|NEWLINE)+
RETURNS_KW: ("Returns"i | "Return"i) [" "] [":"]
RAISES_KW: ("Raises"i | "Throws"i | "Raise"i | "Throw"i) [" "] [":"]
ARGS_KW: ("Args"i | "Arguments"i | "Parameters"i | "Params"i) [" "] ":"
TYPE_SPEC: "(" DWORD ")"

DIGITLETTER: LETTER|DIGIT|"_"
DWORD: DIGITLETTER+
NEWLINE: (CR? LF)

%import common.LETTER
%import common.WS
%import common.DIGIT
%import common.CR
%import common.LF

// Disregard spaces in text
%ignore WS
```

Gramatika popisujúca NumpyDoc štýl dokumentácie

```
start: LINE+ NEWLINE* parameters NEWLINE*

parameters.1: section+

section.1: DWORD NEWLINE "-"+ NEWLINE+ SENTENCE+ [NEWLINE]
SENTENCE: /./+ [NEWLINE]
LINE: /./+ NEWLINE

DIGITLETTER: LETTER|DIGIT
DWORD: DIGITLETTER+

%import common.LETTER
%import common.NEWLINE
%import common.DIGIT
%import common.WS

// Disregard spaces in text
%ignore WS
```

Příloha B

Konfigurácia evaluácie modelov pre generovanie dokumentácie

B.1 Konfigurácia evaluátora

```
{ "evaluators": [
        {
            "class": "LengthEvaluator",
            "params": {
                "length_penalty": 0.5,
                "low_length_treshold": 0.7,
                "upper_length_treshold": 1.3}
        },
        {
            "class": "GrammarEvaluator",
            "params":{
                "penalty": 0.5,
                "grammar_file": "google_style_grammar.txt"}
        },
        {
            "class": "SacreBleuEvaluator",
            "params":{
                "smooth_method": "add-k",
                "smooth_value": 10,
                "tokenize": "intl",
                "lowercase": false,
                "force": false,
                "use_effective_order": false}
        },
        {
            "class": "RougeEvaluator",
            "params": {
                "use_stemmer": false,
                "use_aggregator": true}
        },
        {
            "class": "MeteorEvaluator",
            "params": {
                "alpha": 0.4,
                "beta": 6,
                "gamma": 0.5}
        },
        {
            "class": "EmbeddingSimilarityEvaluator",
            "params":{
                "model": "all-MiniLM-L6-v2",
                "device": "cuda"}
} ] }
```

Obj 2: Konfigurácia evaluátora použitá pre hodnotenie modelov.

Příloha C

Detailné vyhodnotenie testovaných modelov

C.1 Metriky modelu `codet5p-220m-docstring`


```
"length_evaluator": {
 "score": 0.7777777777777778,
 "average": 0.7777777777777778,
 "variance": 0.06172839506172841 },
"grammar_evaluator": {
 "score": 0.9333333333333333,
 "average": 0.9333333333333333,
 "variance": 0.028888888888888895 },
"sacrebleu_evaluator": {
 "score": 0.2236285345443915,
 "counts": [1376, 842.0, 563.0, 381.0],
 "totals": [1825, 1790.0, 1745.0, 1700.0],
 "precisions": [75.3972602739726, 47.039106145251395,
  32.26361031518625, 22.41176470588235],
 "bp": 0.5588256178680294,
 "sys_len": 1825,
 "ref_len": 2887 },
"rouge_evaluator": {
 "rouge1": 0.4808070527174314,
 "rouge2": 0.23713362879634822,
 "rougeL": 0.41565444640359983,
 "rougeLsum": 0.46847079672338365,
 "score": 0.3778650426391265 },
"meteor_evaluator": {
 "meteor": 0.5092927045284931,
 "score": 0.5092927045284931 },
"embedding_similarity_evaluator": {
 "score": 0.7233037352561951,
 "average": 0.7233037352561951,
 "variance": 0.015652503818273544 }
```


Obj 3: Celkové výsledky adaptovaného modelu codet5p-220m-docstring.

C.2 Metriky modelu `codet5p-770m-docstring`


```
"length_evaluator": {
 "score": 0.8777777777777778,
 "average": 0.8777777777777778,
 "variance": 0.04617283950617283 },
"grammar_evaluator": {
 "score": 0.7333333333333333,
 "average": 0.7333333333333333,
 "variance": 0.06222222222222223 },
"sacrebleu_evaluator": {
 "score": 0.3292854766774483,
 "counts": [1876, 1136.0, 801.0, 582.0],
 "totals": [2899, 2864.0, 2819.0, 2774.0],
 "precisions": [64.71196964470506, 39.66480446927374,
  28.41433132316424, 20.980533525594808],
 "bp": 0.9362387241470598,
 "sys_len": 2899,
 "ref_len": 3090 },
"rouge_evaluator": {
 "rouge1": 0.5075649978694565,
 "rouge2": 0.25809217044886285,
 "rougeL": 0.42562419617541614,
 "rougeLsum": 0.49094242373797004,
 "score": 0.39709378816457847 },
"meteor_evaluator": {
 "meteor": 0.518481354749034,
 "score": 0.518481354749034 },
"embedding_similarity_evaluator": {
 "score": 0.7314227819442749,
 "average": 0.7314227819442749,
 "variance": 0.02060026489198208 }
```


Obj 4: Celkové výsledky adaptovaného modelu codet5p-770m-docstring.

C.3 Metriky modelu `led_base_16384_docstring`


```
"length_evaluator": {
 "score": 0.8,
 "average": 0.8,
 "variance": 0.060000000000000005 },
"grammar_evaluator": {
 "score": 0.6222222222222222,
 "average": 0.6222222222222222,
 "variance": 0.04617283950617283 },
"sacrebleu_evaluator": {
 "score": 0.17059913748713942,
 "counts": [2137, 992.0, 548.0, 335.0],
 "totals": [4690, 4655.0, 4610.0, 4565.0],
 "precisions": [45.56503198294243, 21.310418904403868,
  11.887201735357918, 7.338444687842278],
 "bp": 1.0,
 "sys_len": 4690,
 "ref_len": 3646 },
"rouge_evaluator": {
 "rouge1": 0.41388964188559374,
 "rouge2": 0.1655422847955223,
 "rougeL": 0.28659548084121034,
 "rougeLsum": 0.39433286640010506,
 "score": 0.2886758025074421 },
"meteor_evaluator": {
 "meteor": 0.3551420749146566,
 "score": 0.3551420749146566 },
"embedding_similarity_evaluator": {
 "score": 0.6085264086723328,
 "average": 0.6085264086723328,
 "variance": 0.01986384019255638 }
```


Obj 5: Celkové výsledky adaptovaného modelu led_base_16384_docstring.

C.4 Metriky modelu gpt-3.5-turbo-0125


```
"length_evaluator": {
 "score": 0.8666666666666667,
 "average": 0.8666666666666667,
 "variance": 0.04888888888888891 },
"grammar_evaluator": {
 "score": 0.6666666666666666,
 "average": 0.6666666666666666,
 "variance": 0.05555555555555555 },
"sacrebleu_evaluator": {
 "score": 0.2550705061990907,
 "counts": [1951, 1043.0, 657.0, 434.0],
 "totals": [3482, 3447.0, 3402.0, 3357.0],
 "precisions": [56.03101665709362, 30.258195532346967,
  19.312169312169313,12.928209711051535],
 "bp": 1.0,
 "sys_len": 3482,
 "ref_len": 3261 },
"rouge_evaluator": {
 "rouge1": 0.4774974654203554,
 "rouge2": 0.22591534161484572,
 "rougeL": 0.373232126531229,
 "rougeLsum": 0.4557188584352525,
 "score": 0.3588816445221434 },
"meteor_evaluator": {
 "meteor": 0.44481077604306435,
 "score": 0.44481077604306435 },
"embedding_similarity_evaluator": {
 "score": 0.740044891834259,
 "average": 0.740044891834259,
 "variance": 0.015778005123138428 }
```


Obj 6: Celkové výsledky modelu gpt-3.5-turbo-0125 od spoločnosti OpenAI.

C.5 Metriky modelu `llama-3-8b-instruct`


```
"length_evaluator": {
 "score": 0.5,
 "average": 0.5,
 "variance": 0.0 },
"grammar_evaluator": {
 "score": 0.6222222222222222,
 "average": 0.6222222222222222,
 "variance": 0.04617283950617284 },
"sacrebleu_evaluator": {
 "score": 0.0653127232500231,
 "counts": [2546, 1038.0, 552.0, 302.0],
 "totals": [12534, 12499.0, 12454.0, 12409.0],
 "precisions": [20.31274932184458, 8.304664373149851,
  4.432310904127188, 2.433717463131598],
 "bp": 1.0,
 "sys_len": 12534,
 "ref_len": 4163 },
"rouge_evaluator": {
 "rouge1": 0.2694109942182218,
 "rouge2": 0.10445118387795685,
 "rougeL": 0.1823305014121964,
 "rougeLsum": 0.25619677144061803,
 "score": 0.18539755983612502 },
"meteor_evaluator": {
 "meteor": 0.17866241357266302,
 "score": 0.17866241357266302 },
"embedding_similarity_evaluator": {
 "score": 0.6460006833076477,
 "average": 0.6460006833076477,
 "variance": 0.0178722832351923 }
```


Obj 7: Celkové výsledky modelu llama-3-8b-instruct od spoločnosti Predibase.