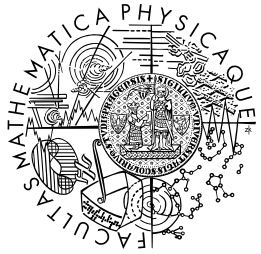


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Václav Hrouda

Automatizace generování popisů produktů pomocí neuronových jazykových modelů

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce:	Ing. Zdeněk Kasner
Studijní program:	Informatika (B0613A140006)
Studijní obor:	Informatika se specializací Databáze a web

Praha 2024

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Na tomto místě bych chtěl poděkovat vedoucímu práce Ing. Zdeňku Kasnerovi za odborné rady, podněty a připomínky týkající se zpracování práce. Dále ústavu formální a aplikované lingvistiky MFF UK za poskytnutí hardwaru potřebného pro uskutečnění experimentů a poskytnutí přístupu k velkým jazykovým modelům. Také bych chtěl poděkovat společnosti wpj s.r.o. za poskytnutí dat, na kterých byly experimenty prováděny. V neposlední řadě děkuji všem, kteří se podíleli na hodnocení vygenerovaných textů.

Název práce: Automatizace generování popisů produktů pomocí neuronových jazykových modelů

Autor: Václav Hrouda

ústav: Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Ing. Zdeněk Kasner, Ústav formální a aplikované lingvistiky

Abstrakt: Popisky produktů jsou důležitou součástí prezentace zboží v ecommerce. Tato bakalářská práce zkoumá možnosti použití jazykových modelů, založených na architektuře Transformer, ke generování popisků produktů na základě textových informací o produktech. Během práce byla použita data z reálného eshopu a byly vyzkoušeny tři různé přístupy. Fine-tuning malého modelu GPT2 small czech, využití modelu Mistral s překladem jeho vstupů a výstupů do angličtiny a přímé použití ChatGPT na českých datech. K vyhodnocení vygenerovaných textů byla použita kombinace automatických metrik a lidského hodnocení. Výsledkem je jasné pořadí těchto přístupů (ChatGPT, Mistral, GPT2 small czech) s tím, že se ukázalo, že pro použití v praxi není žádný z přístupů dostatečně spolehlivý.

Klíčová slova: generování textu z dat, jazykové modely, ecommerce

Title: Automatization of Generating Product Descriptions With Neural Language Models

Author: Václav Hrouda

institute: Institute of Formal and Applied Linguistics

Supervisor: Ing. Zdeněk Kasner, Institute of Formal and Applied Linguistics

Abstract: Product descriptions are an important part of product presentation in e-commerce. This bachelor thesis explores the possibilities of using language models based on the Transformer architecture to generate product descriptions based on textual product information. Data from a real ecommerce store was used and three different approaches were tested during the work. Fine-tuning of the GPT2 small Czech model, using the Mistral model with the translation of its inputs and outputs into English and directly using ChatGPT on the Czech data. A combination of automated metrics and human moderation was used to evaluate the generated texts. The result is a clear ranking of these approaches (ChatGPT, Mistral, GPT2 small Czech), with none proving sufficiently reliable for practical use.

Keywords: data-to-text generation, language models, ecommerce

Obsah

Úvod **3**

1 Základy neuronových sítí a jazykových modelů **5**
1.1 Historie a vývoj neuronových sítí 5
1.1.1 Ranné počátky (1940–1960) 6
1.1.2 Zima umělé inteligence (1970–1985) 6
1.1.3 Oživení a nové směry (1986–2000) 6
1.1.4 Era hlubokého učení (2000–Současnost) 6
1.2 Principy fungování jazykových modelů 7
1.3 N-gramové modely 7
1.4 Rekurentní neuronové sítě (RNN) 8
1.4.1 Architektura RNN 9
1.4.2 Paměť a kontext 9
1.4.3 Problémy s dlouhodobou závislostí 9
1.4.4 Variace RNN 9
1.5 Slovní embeddingy 10
1.5.1 Princip slovních embeddingů 10
1.5.2 Techniky pro vytváření embeddingů 10
1.5.3 Aplikace slovních embeddingů 10
1.6 Architektura Transformer 11
1.6.1 Struktura Transformeru 11
1.6.2 Mechanismus Self-Attention 12
1.6.3 Multi-Head Attention 12
1.7 Předtrénované modely 13
1.7.1 Výhody předtrénovaných modelů 13
1.7.2 Použití předtrénovaných modelů 13
1.7.3 Příklady Předtrénovaných modelů 14
1.8 Velké jazykové modely 14
1.8.1 ChatGPT 14
1.8.2 Mistral 7B 15
1.9 Jazykové modely a čeština 15

2 Návrh experimentu **17**
2.1 Data . . . 17
2.2 Postupy generování textu . . . 17
2.3 Vyhodnocení . . . 18
2.3.1 Automatické metriky . . . 18
2.3.2 Hodnocení lidskými anotátory . . . 19

3 Příprava dat a trénování modelů **21**
3.1 Infrastruktura . . . 21
3.1.1 SLURM Workload Management . . . 22
3.2 Webová aplikace . . . 23
3.2.1 Technologie . . . 25
3.2.2 Architektura . . . 27
3.2.3 Slurm joby . . . 28
3.3 Příprava dat . . . 31
3.3.1 Proces získávání dat z e-shopů . . . 31
3.3.2 Čištění a formátování dat . . . 31
3.3.3 Tvorba datasetu . . . 33
3.4 Fine-tuning modelu . . . 33
3.4.1 Tokenizace . . . 33
3.4.2 Data collator . . . 35
3.4.3 Argumenty pro trénování . . . 36
3.5 Použití velkých předtrénovaných modelů . . . 37
3.5.1 Vstup pro modely . . . 37
3.6 Mistral . . . 39
3.7 ChatGPT . . . 39

4 Hodnocení kvality výsledků **41**
4.1 Automatická evaluace . . . 41
4.1.1 Zvolené metriky . . . 41
4.1.2 Výběr nejlepšího fine-tunovaného modelu . . . 41
4.2 Lidští anotátoři . . . 42
4.2.1 Zvolené metriky . . . 42
4.3 Výsledky . . . 48

Závěr **55**

Seznam použité literatury **56**

A Webová aplikace **58**

B Ukázková data **59**

Úvod

V dnešním digitálně zaměřeném světě se e-commerce stala klíčovou součástí obchodního prostředí. Produkty nabízené v online obchodech jsou zpravidla prezentovány pomocí kombinace vizuálních prvků a textových popisů. Tyto popisy hrají zásadní roli v rozhodovacím procesu zákazníků, poskytují nezbytné informace o vlastnostech produktů a přispívají k lepšímu porozumění nabízeného zboží. Mají také výrazný vliv na SEO (Search Engine Optimization), což hraje klíčovou roli v tom, jestli produkt k zákazníkovi dostane. V kontextu rostoucího objemu online prodeje a neustále se měnící nabídky produktů však vzniká výzva v podobě efektivního a přesného generování popisů produktů.

Tradiční přístupy k vytváření těchto popisů obvykle zahrnují manuální práci, která zajišťuje vysokou kvalitu a přesnost textu. Tento proces je však časově náročný a ekonomicky nákladný, zejména v případě velkých e-shopů, které nabízejí široký sortiment produktů. Alternativním řešením může být použití automatizovaného generování popisů pomocí šablon, které však často nedosahují požadované úrovně kvality a mohou vést k generování stereotypních popisů [1].

V této práci se zaměřím na nové přístupy v oblasti generování textu, konkrétně na využití předtrénovaných neuronových jazykových modelů architektury Transformer [2]. Tyto modely, vyvinuté v posledních letech, představují pokročilé technologie v oblasti zpracování přirozeného jazyka (NLP) a umožňují generování plynulého, koherentního a relevantního textu na základě vstupních dat.

Cílem této práce je prozkoumat, zda lze tyto modely efektivně využít pro generování popisů produktů v českém e-commerce prostředí. K tomuto účelu bude proveden důkladný průzkum současných metod v oblasti NLP, se zaměřením na vícejazyčné modely a modely specificky trénované pro češtinu. V praktické části práce bude vyvinuta webová aplikace umožňující snadné stahování dat produktů z e-shopů a jejich zpracování pro potřeby generativních jazykových modelů. Pomocí této aplikace budou provedeny experimenty s cílem vyhodnotit efektivitu a přesnost generovaných popisů z hlediska jejich plynulosti, faktické přesnosti a celkové kvality.

Výsledky této práce umožní lepší přehled o tom, jaké jsou možnosti nasazení neuronových jazykových modelů v praxi českých e-commerce platforem.

Práce obsahuje nejen teoretický přehled, ale také praktické ukázky a doporučení, která mohou pomoci dalšímu vývoji a vylepšení automatizovaných systémů pro generování textu v oblasti e-commerce.

Kapitola 1

Základy neuronových sítí a jazykových modelů

V posledních desetiletích došlo v oblasti umělé inteligence (AI) a strojového učení (ML) k zásadnímu posunu, který významně ovlivnil nejen akademický svět, ale i komerční aplikace. Jedním z nejvýznamnějších milníků v této éře byl rozvoj a zdokonalení neuronových sítí a jazykových modelů. Tyto technologie se staly základem pro širokou škálu aplikací, od rozpoznávání obrazu a zpracování přirozeného jazyka (NLP) až po autonomní řízení vozidel a personalizované doporučovací systémy.

Neuronové sítě, inspirované strukturou a funkcí lidského mozku, se staly základním stavebním kamenem pro většinu moderních AI systémů [3, 4]. Od jednoduchých modelů, které byly schopné základního učení a adaptace, se neuronové sítě vyvinuly do složitých architektur schopných zpracovávat a interpretovat obrovské objemy dat s neuvěřitelnou přesností.

Paralelně s tímto vývojem došlo k nárůstu zájmu a výzkumu v oblasti jazykových modelů. Tyto modely, které se zaměřují na porozumění, generování a překlad přirozeného jazyka, využívají neuronové sítě k modelování složitých jazykových struktur a jejich významů. V posledních letech byly představeny revoluční jazykové modely, jako je GPT (Generative Pretrained Transformer) [5] nebo BERT (Bidirectional Encoder Representations from Transformers) [6], které přinesly nové možnosti v oblasti zpracování přirozeného jazyka a umožnily vytváření přesnějších, plynulých a kontextově relevantních textů.

1.1 Historie a vývoj neuronových sítí

Rozvoj neuronových sítí lze sledovat od prvních teoretických konceptů v polovině 20. století až po dnešní sofistikované systémy.

1.1.1 Ranné počátky (1940–1960)

Model Neuronu Neuronové sítě mají své kořeny v pokusech modelovat procesy lidského mozku pomocí matematiky a inženýrství. Významným milníkem byla práce McCullocha a Pittse [7], kteří představili model umělého neuronu, zjednodušenou abstrakci biologického neuronu. Tento model byl schopen provádět základní logické operace a položil základy pro pozdější vývoj neuronových sítí.

Perceptron Dalším klíčovým momentem bylo vytvoření perceptronu Frankem Rosenblattem [8]. Perceptron představoval jednoduchou, ale revoluční formu neuronové sítě, která byla schopna učení prostřednictvím posilování nebo oslabování synaptických spojení, což napodobovalo procesy učení v lidském mozku.

1.1.2 Zima umělé inteligence (1970–1985)

Tzv. „zima umělé inteligence“ nastala v 70. letech a byla charakterizována poklesem zájmu a financování v oblasti AI. Skepticismus byl způsoben přehnanými očekáváními a omezenými technologickými schopnostmi tehdejších systémů. Tento pokles pokračoval až do 80. let, kdy se objevil nový zájem o neuronové sítě.

1.1.3 Oživení a nové směry (1986–2000)

Zpětné šíření chyby (Backpropagation) Období stagnace skončilo s příchodem nových teoretických a technologických pokroků. Klíčovým momentem bylo představení algoritmu zpětného šíření chyby [9], který umožnil efektivnější trénování vícevrstvých neuronových sítí. Tento přístup umožnil sítím se učit složitější vzory a funkcionalitu.

Konvoluční neuronové sítě (CNN) Dalším důležitým krokem byl rozvoj CNN [10], které se staly základem pro mnoho aplikací v oblasti zpracování obrazu a rozpoznávání vzorů.

1.1.4 Era hlubokého učení (2000–Současnost)

V prvních letech 21. století došlo k značnému rozvoji v oblasti neuronových sítí, což vedlo k období známému jako era hlubokého učení [11]. Tento rozvoj byl podpořen především pokrokem ve výpočetní technice, zejména vývojem grafických karet (GPU), které umožnily zpracovávat větší objemy dat a trénovat složitější modely.

Rozvoj GPU a dalších výpočetních technologií umožnil zpracovávat větší objemy dat a trénovat složitější neuronové sítě. Díky tomu byly představeny revoluční modely, které umožnily dosáhnout průlomů v mnoha oblastech, včetně zpracování přirozeného jazyka, strojového vidění a autonomního řízení.

1.2 Principy fungování jazykových modelů

Jazykové modely (LM) jsou základním stavebním kamenem moderního zpracování přirozeného jazyka. Jejich hlavním úkolem je předpovídat pravděpodobnost sekvence slov ve větě nebo textu. Tato predikce je zásadní pro celou řadu aplikací, od automatického překladu po generování textu.

Nejčastější typy jazykových modelů jsou založeny na statistických metodách, jako jsou n-gramy, a neuronových sítích. N-gramové modely 1.3 pracují s pravděpodobností sekvencí slov založených na frekvenci jejich výskytu v trénovacích datech. Tyto modely však mají omezení, zejména co se týče zachycení delších kontextů a závislostí ve větách.

S rozvojem hlubokého učení se staly populárními neuronové jazykové modely (viz sekce 1.4, 1.5, a 1.6). Tyto modely využívají neuronové sítě k modelování složitých vzorců v datech a dokážou lépe zachytit kontext a nuance v jazyce. Jedním z klíčových příspěvků v této oblasti je model Transformer, který byl představen v roce 2017 [2] a poskytuje výkonný mechanismus pro modelování závislostí v dlouhých textových sekvencích.

1.3 N-gramové modely

N-gramové modely představují jeden z tradičních přístupů v oblasti jazykového modelování. Tyto modely jsou základem pro pochopení vývoje a funkcí jazykových modelů, zejména v kontextu statistického zpracování jazyka.

Základní principy n-gramových modelů N-gram je sekvence n po sobě jdoucích slov v textu. Například věta „Jdu do obchodu“ obsahuje n-gramy různých délek: unigramy („Jdu“, „do“ a „obchodu“), bigramy („Jdu do“ a „do obchodu“) a trigramy („Jdu do obchodu“). N-gramové modely predikují pravděpodobnost následujícího slova ve větě na základě předchozích $n-1$ slov. Tato pravděpodobnost je vypočítána z frekvence výskytu dané sekvence slov v trénovacích datech.

Výpočet pravděpodobností V n-gramovém modelu je pravděpodobnost slova w_n v kontextu jeho předcházejících slov $w_1, w_2, \ldots, w_{n-1}$ dána poměrem frekvence výskytu této n-gramové sekvence k frekvenci výskytu

$(n-1)$-gramové sekvence. Například, pravděpodobnost slova „obchodu“ (w_n) po slovech „Jdu do“ (w_{n-2}, w_{n-1}) v trigramovém modelu by byla vypočítána jako:

$$P(w_n|w_{n-2}, w_{n-1}) = \frac{\text{Count}(w_{n-2}, w_{n-1}, w_n)}{\text{Count}(w_{n-2}, w_{n-1})}$$

Kde $P(w_n|w_{n-2}, w_{n-1})$ je podmíněná pravděpodobnost výskytu slova w_n po sekvenci w_{n-2}, w_{n-1}, $\text{Count}(w_{n-2}, w_{n-1}, w_n)$ je frekvence výskytu trigramu v datech a $\text{Count}(w_{n-2}, w_{n-1})$ je frekvence výskytu bigramu. V ukázkovém příkladu by to bylo:

$$P(\text{"obchodu"}|\text{"Jdu"}, \text{"do"}) = \frac{\text{Count}(\text{"Jdu"}, \text{"do"}, \text{"obchodu"})}{\text{Count}(\text{"Jdu"}, \text{"do"})}$$

Tento vzorec ilustruje princip výpočtu pravděpodobnosti v n-gramových modelech, kde se frekvence konkrétních sekvencí slov využívá k odhadu pravděpodobnosti následujícího slova v daném kontextu.

Omezení n-gramových modelů Hlavní omezení n-gramových modelů spočívá v jejich nezbytné závislosti na velikosti n. S rostoucí délkou n-gramů se zvyšuje přesnost modelu v zachycení kontextu, ale současně roste množství potřebných dat pro efektivní trénování. Modely s krátkými n-gramy (např. bigramy) mají tendenci nezachycovat dostatečný kontext, což vede k nepřesnostem ve výsledcích. Na druhou stranu, modely s dlouhými n-gramy (např. pentagramy) trpí problémem datové řídkosti, kdy mnoho možných n-gramů se v trénovacích datech vůbec neobjeví.

Způsoby řešení omezení Existují různé metody pro zvládnutí omezení n-gramových modelů, jako je například vyhlazování a zpětná diskontace. Vyhlazování pomáhá řešit problém datové řídkosti tím, že rozděluje část pravděpodobnosti od často se vyskytujících n-gramů k méně častým, čímž se snižuje přílišná závislost na četných n-gramech. Zpětná diskontace zase umožňuje modelu lépe se vyrovnat s neznámými slovy nebo n-gramy, které se v trénovacích datech nevyskytují.

N-gramové modely, přestože jsou poměrně primitivní v porovnání s moderními neuronovými jazykovými modely, stále představují důležitý základ pro pochopení základních principů jazykového modelování.

1.4 Rekurentní neuronové sítě (RNN)

Rekurentní neuronové sítě (RNN) představují třídu neuronových sítí, které jsou navrženy pro zpracování sekvenčních dat, jako jsou časové řady, řeč nebo text.

Základní myšlenkou RNN je využití skrytých stavů (angl. hidden states) pro zachování určité formy paměti o předchozích vstupech. Tato vlastnost umožňuje RNN zpracovávat informace v kontextu, což je zásadní pro úlohy, kde je důležitý časový nebo sekvenční kontext dat.

1.4.1 Architektura RNN

Tradiční neuronová síť zpracovává každý vstup nezávisle, bez ohledu na předchozí nebo následující data. Naproti tomu RNN obsahuje smyčky, které umožňují informacím cirkulovat v síti. V RNN je výstup z jedné vrstvy předáván zpět do stejné vrstvy. Tento proces se opakuje v časových krocích, přičemž každý krok představuje jedno sekvenční pozorování v datech.

1.4.2 Paměť a kontext

Klíčovým prvkem RNN je její schopnost udržet „paměť“. V každém kroku má síť přístup nejen k aktuálnímu vstupu, ale také k informacím, které byly zpracovány v předchozích krocích. Toto umožňuje RNN zahrnovat kontext při rozhodování a reagování na nové informace, což je zvláště užitečné pro úlohy jako je predikce dalšího slova v textu nebo pochopení celkového významu věty.

1.4.3 Problémy s dlouhodobou závislostí

Přestože RNN teoreticky mohou zachytit dlouhodobé závislosti v sekvencích dat, v praxi se často setkávají s problémy, jako je ztráta paměti a obtížnost učení s dlouhými sekvencemi. Tento problém, známý jako „vanishing gradient problem“, vede k obtížnosti při trénování standardních RNN na zachycení dlouhodobých závislostí v datech.

1.4.4 Variace RNN

Pro překonání těchto problémů byly vyvinuty pokročilé varianty RNN, jako jsou Long Short-Term Memory (LSTM; [12]) a Gated Recurrent Units (GRU; [13]). Tyto architektury zavádějí speciální struktury, které pomáhají modelu učit se, které informace si ponechat v paměti a které odstranit, čímž zlepšují schopnost modelu zachytit dlouhodobé závislosti.

V souhrnu RNN představují důležitou třídu neuronových sítí pro zpracování sekvenciálních dat a mají širokou škálu aplikací v různých oblastech, včetně přirozeného zpracování jazyka, rozpoznávání řeči a časových řad.

1.5 Slovní embeddingy

Slovní embeddingy představují způsob, jakým jsou slova reprezentována v oblasti NLP. Jedná se o techniku převádění slov a frází z přirozeného jazyka do vektorů číselných hodnot, které jsou schopné zachytit jejich významy, nuance a vztahy v kontextu.

1.5.1 Princip slovních embeddingů

Tradičně byla slova v počítačovém zpracování reprezentována jako diskrétní symboly, což však neumožňovalo zachytit jejich sémantické vlastnosti nebo vztahy mezi nimi. Slovní embeddingy řeší tento problém tím, že každé slovo nebo frázi převádějí do vysokodimenzionálního vektorového prostoru. V tomto prostoru jsou slova s podobnými významy blíže k sobě, což umožňuje modelům zachytit a využívat sémantické informace.

1.5.2 Techniky pro vytváření embeddingů

Existují různé metody pro vytváření slovních embeddingů, přičemž mezi nejpopulárnější patří:

Word2Vec [14] Metoda vyvinutá společností Google, která vytváří embeddingy pomocí neuronových sítí. Word2Vec používá dva hlavní modely, Skip-Gram a CBOW (Continuous Bag of Words), pro generování vektorových reprezentací slov.

GloVe (Global Vectors) [15] Tato metoda kombinuje techniky maticové faktorizace a lokálního kontextového okna pro efektivní výpočet vektorových reprezentací slov.

FastText [16] Vyvinutý Facebookem, rozšiřuje Word2Vec o schopnost zpracovávat části slov, což umožňuje lépe zpracovávat neznámá slova a morfologii slov.

1.5.3 Aplikace slovních embeddingů

Slovní embeddingy najdou uplatnění v řadě úloh v NLP, včetně klasifikace textu, strojového překladu, rozpoznávání entit a analýzy sentimentu. Díky schopnosti zachytit sémantický význam slov umožňují embeddingy vytvářet modely, které jsou schopné lépe rozumět a interpretovat lidský jazyk.

Celkově slovní embeddingy představují zásadní inovaci v zpracování přirozeného jazyka a hrají klíčovou roli v moderních NLP aplikacích a modelech strojového učení.

1.6 Architektura Transformer

Transformer, představený ve studii *Attention Is All You Need* [2] v roce 2017, představuje zásadní inovaci v oblasti zpracování přirozeného jazyka. Tento model znamenal odchod od tradičních sekvenciálních architektur, jako jsou rekurentní neuronové sítě a konvoluční neuronové sítě, a zavedl nový přístup zaměřený na mechanismus self-attention.

Transformer a jeho variace, jako je GPT (Generative Pretrained Transformer) a BERT (Bidirectional Encoder Representations from Transformers), představují významný posun v oblasti jazykových modelů. Tyto modely využívají techniku zvanou „self-attention“, která umožňuje modelu zvážit různé části vstupní sekvence při predikci dalšího slova. Tím se významně zlepšuje schopnost modelů generovat koherentní a kontextově relevantní text [5, 6].

1.6.1 Struktura Transformeru

Transformer se skládá ze dvou hlavních částí: enkodéru a dekodéru. Každá z těchto částí obsahuje několik vrstev, které jsou sestaveny z modulů self-attention a feed-forward neuronových sítí. Celá struktura je zobrazena na obrázku 1.1.

Enkodér

Enkodér Transformeru se skládá z řady identických vrstev. Každá vrstva obsahuje dvě hlavní komponenty:

Multi-Head Self-Attention Mechanism Umožňuje enkodéru se zaměřit na různé části vstupní sekvence. Tento mechanismus umožňuje modelu lépe zachytit kontext slov v rámci celé věty nebo textového segmentu.

Pointwise Feed-Forward Network Jedná se o plně propojenou (dense) neuronovou síť, která transformuje výstup z attention mechanismu. Každá pozice v sekvenci je zpracovávána odděleně a nezávisle.

Dekodér

Dekodér také obsahuje několik vrstev a každá vrstva má tři hlavní komponenty:

Masked Multi-Head Self-Attention Podobně jako enkodér, ale se zavedením masky, která zabraňuje toku informací od slov dále v textu. To zajistí, že při predikci daného slova má model k dispozici pouze předcházející slova.

Multi-Head Attention Tato komponenta přijímá výstupy z enkodéru, díky čemuž umožňuje dekodéru zaměřit se na relevantní části vstupní sekvence během generování výstupu.

Pointwise Feed-Forward Network Stejný jako v enkodéru.

1.6.2 Mechanismus Self-Attention

Mechanismus self-attention v architektuře Transformer představuje sofistikovaný způsob, jakým model zpracovává význam a vztahy slov v textové sekvenci. Na rozdíl od tradičních sekvenciálních metod, jako jsou RNN a LSTM, které zpracovávají text postupně slovo za slovem, self-attention umožňuje každému slovu v sekvenci zohlednit význam ostatních slov v sekvenci. Tento proces umožňuje modelu získat komplexní přehled o kontextu a vztazích mezi slovy, což vede k lepšímu porozumění celkové struktury a významu věty.

1.6.3 Multi-Head Attention

Multi-head attention rozšiřuje koncept self-attention tím, že rozděluje proces pozornosti do několika samostatných „hlav“. Každá z těchto hlav se zaměřuje na různé aspekty informací ve větě. Tímto způsobem je model schopen paralelně zpracovávat různé druhy vztahů a významů ve větě, jako jsou syntaktické struktury a sémantické kontexty. Tato multifunkčnost zvyšuje schopnost modelu rozpoznávat složité vzory a nuance v jazyce, čímž značně zlepšuje jeho výkon při úlohách zpracování přirozeného jazyka.

Tok dat v Transformeru

Enkodér přijímá vstupní sekvenci slov, která je nejprve převedena na vektorové reprezentace (slovní embeddingy). Tyto embeddingy jsou pak transformovány prostřednictvím série self-attention a feed-forward vrstev enkodéru, čímž vzniká kontextově obohacená reprezentace vstupní sekvence. Každý embedding je také posunut o specifickou hodnotu, která odpovídá pozici slova ve větě (position encoding), aby model dokázal zachovat informace o sekvenčním uspořádání slov.

Výstupy enkodéru jsou následně předány do dekodéru. Dekodér využívá těchto kontextově bohatých reprezentací společně s vlastními embeddingy pro generování výstupní sekvence. Tyto výstupní embeddingy odpovídají dosud generovaným slovům ve výstupní sekvenci a jsou postupně aktualizovány s každým novým vygenerovaným slovem.

Dekodér kombinuje informace z enkodéru a své vlastní predikce, aby vygeneroval další slovo ve výstupní sekvenci. Pro tento proces využívá masked self-attention mechanismus, který zabrání modelu vidět nevygenerovaná slova ve výstupu. Tímto způsobem je každé nově generované slovo ovlivněno jak kontextem vstupní sekvence, tak již vygenerovaným textem.

Tato sofistikovaná interakce mezi enkodérem a dekodérem, spolu s využitím slovních a pozicových embeddingů, umožňuje Transformeru efektivně generovat přesné a kontextově relevantní výstupy, které zohledňují jak vstupní sekvenci, tak dosud generovaný obsah.

Na závěr je výstup vyprodukovaný dekodérem převeden na tzv. logit vektor pomocí lineární vrstvy, což je jednoduchá plně propojená neuronová síť. Tento vektor má stejnou délku jako počet možných slov, které je model schopen generovat. Poslední vrstva Softmax převede logit vektor na vektor pravděpodobností pro jednotlivá slova. Jedno ze slov (záleží na nastavení modelu) s největší pravděpodobností je poté vybráno jako finální výstup.

1.7 Předtrénované modely

Předtrénované modely jsou modely neuronových sítí, které byly již předem natrénovány na rozsáhlém datasetu, často obsahujícím širokou škálu jazykových dat. Tato data mohou zahrnovat knihy, články, webové stránky a další formy psaného textu. Hlavním účelem předtrénování je naučit model základním jazykovým strukturám, slovní zásobě a kontextuálním vztahům.

1.7.1 Výhody předtrénovaných modelů

Předtrénované modely přinášejí několik významných výhod:

Zlepšení výkonu Modely natrénované na rozsáhlých a diverzifikovaných datech jsou schopné lépe rozumět a generovat přirozený jazyk.

Úspora času a zdrojů Vývojáři a výzkumníci mohou použít předtrénované modely jako výchozí bod pro specifické úlohy, což výrazně snižuje potřebu dlouhého a nákladného trénování od začátku.

Flexibilita Předtrénované modely mohou být dotrénovány (fine-tuned) pro konkrétní aplikace, což umožňuje jejich široké využití v různých oblastech.

1.7.2 Použití předtrénovaných modelů

Použití předtrénovaných modelů obvykle zahrnuje dva hlavní kroky:

1. **Předtrénování:** Model je nejprve natrénován na obrovském datasetu. Tento proces se zaměřuje na učení se obecných jazykových vzorců a struktur.
2. **Fine-tuning:** Po předtrénování je model dotrénován na menším, specifickém datasetu. Během trénování se model upravuje tak, aby lépe vyhovoval konkrétním požadavkům nebo úlohám.

1.7.3 Příklady Předtrénovaných modelů

Několik známých příkladů předtrénovaných modelů zahrnuje:

BERT (Bidirectional Encoder Representations from Transformers) Vyvinutý Googlem, BERT efektivně chápe kontext slov ve větě pomocí bidirekcionálního trénování, což umožňuje lepší pochopení významu slov v různých kontextech.

GPT (Generative Pretrained Transformer) Vytvořený OpenAI, GPT modely excelují v generování koherentního a relevantního textu, a jsou především trénovány pro účely generování textu s vysokou mírou přesnosti a kontextové relevanci.

BART (Bidirectional and Auto-Regressive Transformers) BART, vyvinutý Facebookem, kombinuje výhody bidirekcionálního zpracování BERTu a generativních schopností GPT. Je to efektivní model pro úlohy, jako je sumarizace textu a oprava chyb ve větách.

1.8 Velké jazykové modely

Velké jazykové modely představují nejnovější vývoj v oblasti strojového učení a zpracování přirozeného jazyka. Tyto modely, jako jsou ChatGPT od OpenAI a Mistral [17] od MistralAI, představují špičku v technologii umělé inteligence a nabízejí nástroje pro generování textu, automatické odpovídání na dotazy nebo tvorbu kreativního obsahu.

1.8.1 ChatGPT

ChatGPT, vyvinutý společností OpenAI, je jedním z nejvýznamnějších příkladů velkých jazykových modelů. Tento model je založen na architektuře Transformer a je navržen tak, aby generoval koherentní a relevantní odpovědi v konverzačním stylu. ChatGPT je trénován na rozsáhlých datových sadách, které zahrnují různé formy komunikace, od běžných rozhovorů po odborné texty, což mu umožňuje flexibilně reagovat na širokou škálu dotazů a témat.

Hlavními vlastnostmi ChatGPT jsou:

Kontextové porozumění Schopnost modelu chápat a reagovat na komplexní dotazy založené na poskytnutém kontextu.

Kreativní generování textu Možnost generovat originální a relevantní texty, od technických odpovědí po kreativní psaní.

Flexibilita v aplikacích Využití v širokém spektru aplikací, včetně chatbotů, asistentů zákaznické podpory a edukačních nástrojů.

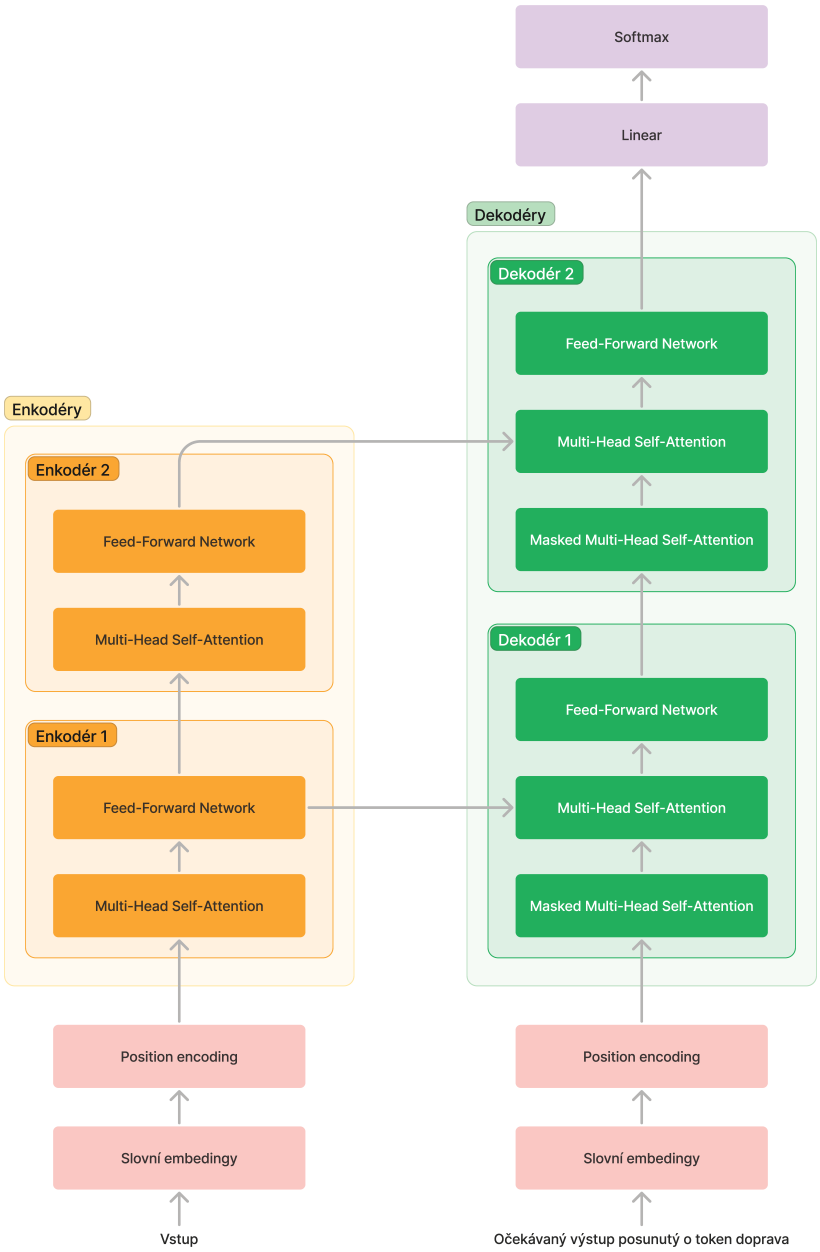
1.8.2 Mistral 7B

Mistral 7B [17], vyvinutý společností Mistral AI, je jazykový model, který se vyznačuje efektivitou a výkonem při menším počtu parametrů (7.3 miliardy). Přesto překonává větší modely jako Llama 2 13B a Llama 1 34B v různých benchmarkových testech. Díky použití Grouped-query attention a Sliding Window Attention pro efektivnější zpracování delších sekvencí.

Model Mistral 7B je k dispozici pod licencí Apache 2.0, což umožňuje jeho široké využití a adaptaci.

1.9 Jazykové modely a čeština

Jelikož je trénování velkých jazykových modelů výpočetně náročný a drahý proces, tak velké modely natrénované přímo pro češtinu neexistují. K dispozici jsou pouze varianty GPT-2 small (viz [18] nebo [19]). Velké modely jako Mistral nebo ChatGPT zvládají generovat české texty, jelikož byly trénovány na vícejazyčném datasetu. V těchto datasetech ale není zastoupení češtiny tak velké jako angličtiny. Kvůli tomu jsou anglické výsledky těchto modelů lepší.



Obrázek 1.1 Schéma architektury Transformer

Kapitola 2

Návrh experimentu

V této kapitole se zaměříme na návrh experimentů. Cílem je prozkoumat, jak různé metody generování textu ovlivňují kvalitu a přesnost popisů produktů v e-commerce. V práci se nachází porovnání tří různých přístupů: finetuning stávajícího jazykového modelu pro data z českého e-shopu, využití kombinace překladů a modelu Mistral pro zpracování textů, a přímé použití modelu ChatGPT na data v českém jazyce. K vyhodnocení vygenerovaných textů je použita kombinace automatických metrik a hodnocení lidskými anotátory.

2.1 Data

V práci byla použita data z reálného e-shopu. Dataset obsahuje informace o 57683 produktech. Každý z těchto produktů obsahuje:

- název produktů
- zkrácený popis produktu
- seznam parametrů produktu (např. barva, rozměry, kategorie produktu, …)
- seznam variant produktu (např. různé velikosti oblečení)
- kompletní popis produktu

2.2 Postupy generování textu

Fine-tuning malého modelu V rámci práce jsem se zaměřili na finetuning modelu GPT-2 Small CzechCS[1], který je natrénován pro český jazyk [18].

[1] spital/gpt2-small-czech-cs

Proces finetuningu spočívá v dalším trénování již předtrénovaného modelu na datech konkrétního e-shopu. Cílem je dosáhnout vyšší přesnosti a relevanci generovaného textu specifický e-shop.

Překlad a model Mistral Tento přístup zahrnuje překlad vstupních dat z češtiny do angličtiny, generování textu pomocí modelu Mistral a následný překlad zpět do češtiny. Tato metoda umožňuje využít pokročilých anglických modelů pro generování textu v jazyce, pro který nemusí být dostupné nativní modely.

ChatGPT Tato metoda zahrnuje přímé generování popisů produktů pomocí modelu ChatGPT na základě českých dat bez nutnosti překladu. Tato metoda zkoumá schopnost modelu ChatGPT generovat relevantní a kvalitní texty v češtině.

2.3 Vyhodnocení

Vyhodnocení vygenerovaných textů nám umožňuje posoudit účinnost a přesnost různých metod generování textu. Pro tento účel byly použity jak automatické metriky, tak hodnocení lidskými anotátory.

2.3.1 Automatické metriky

Automatizované metriky nám poskytují rychlé kvantitativní hodnocení kvality generovaných textů. Automatizovaných metrik existuje celá řada. V této sekci popíšeme metriky použité v této práci pro hodnocení a srovnání různých metod generování textu.

BLEU (Bilingual Evaluation Understudy)

Popis Tato metrika byla původně vyvinuta pro hodnocení kvality strojového překladu, ale je také užitečná pro hodnocení dalších forem generovaného textu. Je založena na srovnání n-gramů generovaného textu s n-gramy z jednoho nebo více referenčních textů. Vysoké skóre BLEU naznačuje, že generovaný text se blíží kvalitě lidského referenčního materiálu.

Výpočet BLEU skóre se vypočítá jako geometrický průměr n-gramové přesnosti, penalizovaný za příliš krátké generované texty. Přesnost n-gramů se vypočítává jako poměr počtu správných n-gramů generovaného textu ku celkovému počtu n-gramů v generovaném textu. Pro penalizaci krátkosti se používá tzv. *brevity penalty* (BP).

ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

Popis: ROUGE je sada metrik určená pro hodnocení automatických souhrnů textu. ROUGE měří míru shody (overlap) mezi generovaným textem a referenčním textem, s důrazem na recall. Recall v kontextu ROUGE znamená schopnost generovaného textu obsáhnout co nejvíce relevantních informací z referenčních textů. Jinými slovy, recall hodnotí, kolik z informací uvedených v referenčních textech se objevuje v generovaném textu. Vyšší recall znamená, že generovaný text úspěšně zachycuje většinu obsahu referenčních textů, což je důležité při hodnocení textu, který má obsahovat informace založené na vstupních datech jako je popis produktu.

Výpočet: Existuje několik variant ROUGE, jako například ROUGE-N (porovnává n-gramy), ROUGE-L (založeno na nejdelší společné podposloupnosti), a ROUGE-S (založeno na sekvenci slov s mezerami). Výpočet ROUGE-N se například provádí jako poměr počtu shodujících se n-gramů v generovaném textu a referenčním textu ku počtu n-gramů v referenčním textu.

METEOR (Metric for Evaluation of Translation with Explicit Ordering)

Popis: METEOR je metrika pro hodnocení strojových překladů, která zohledňuje synonyma a parafraze pro flexibilnější porovnání sémantické a syntaktické shody mezi generovaným textem a referencem.

Výpočet: METEOR využívá mapování mezi slovy generovaného textu a referencí, zahrnující synonyma a parafraze. Skóre se vypočítává na základě shody (precision) a úplnosti (recall) tohoto mapování, s vyvážením těchto dvou aspektů.

2.3.2 Hodnocení lidskými anotátory

Přestože automatické metriky, jako jsou BLEU, ROUGE nebo METEOR, poskytují rychlé a objektivní hodnocení kvality generovaného textu, mají zásadní omezení. Tyto metriky často porovnávají generovaný text s předem stanovenou referencí a hodnotí shodu na základě lexikální přesnosti. Tento přístup však nedokáže zcela zachytit sémantickou a kontextovou hloubku jazyka. Například, fráze s odlišnými slovy, ale se stejným významem, mohou být automatickými metrikami hodnoceny nízce, přestože jsou lingvisticky validní a relevantní.

Lidské hodnocení přináší do procesu validace generovaného textu odlišnou perspektivu. Hodnotitelé jsou schopni posoudit nejen lexikální a gramatickou správnost, ale také sémantickou koherenci, kontextovou přiměřenost a celkovou

čitelnost textu. To je obzvláště důležité v případech, kde je kontext složitý nebo kde je důležitá kreativní či emotivní složka textu.

Existují různé přístupy, jak provádět lidské hodnocení generovaného textu. Jednou z metod je použití Likertovy škály, kde anotátoři hodnotí různé aspekty textu, jako je přirozenost, koherence a přesnost, na stupnici od velmi nízké po velmi vysokou. Další přístupem je porovnávací hodnocení, kde anotátoři porovnávají několik variant textu a vybírají nejlepší podle daných kritérií. Důležité je také zajistit, aby hodnocení bylo prováděno skupinou různorodých anotátorů, aby se předešlo subjektivním zkreslením a zajištěna byla reprezentativnost výsledků.

Kapitola 3

Příprava dat a trénování modelů

Tato kapitola obsahuje detailní popis jednotlivých přístupů pro generování popisků. Zpočátku představuje dostupný výpočetní výkon a infrastrukturu pro spouštění experimentů. Dále se věnuje procesu přípravy dat pro experimenty. Následuje popis tří přístupů, které jsem pro generování textu použil:

1. Fine-tuning modelu GPT-2-small-cs
2. Použití modelu Mistral s překladem
3. Přímé použití ChatGPT

3.1 Infrastruktura

Pro trénování modelů umělé inteligence je klíčovým hardwarovým prvkem grafický procesor (GPU). Zásadní vlastností GPU je jejich schopnost provádět paralelní výpočty, což je nezbytné pro efektivní zpracování algoritmů strojového učení, zejména při trénování hlubokých neuronových sítí. Kromě GPU existují i specializované alternativy[1], jako jsou Tensor Processing Units (TPU), které jsou optimalizovány pro specifické úlohy v AI. Tyto komponenty, spolu s dostatečnou kapacitou paměti, tvoří základ pro efektivní trénování AI modelů.

Pro fine-tuning jazykových modelů jsem měl příležitost využívat studentský cluster AIC[2]. Tento cluster je vybaven grafickými kartami. Je strukturován do dvou uzlů (nodů), přičemž každý z nich má instalovány různé typy grafických karet. Specifikace grafických karet na jednotlivých nodech jsou představeny v tabulkách 3.1 a 3.2, které poskytují přehled o hardwarovém vybavení, s nímž

[1] Tyto alternativy nejsou běžně dostupné. Buďto je používají pouze firmy, které si je vyvinuly pro vlastní potřeby, nebo je pronajímají cloudoví poskytovatelé.

[2] `https://aic.ufal.mff.cuni.cz/`

jsem pracoval. Vzhledem k omezením clusteru jsem byl schopen využít vždy pouze jednu grafickou kartu současně.

Název GPU	Celková paměť (GiB)
GPU 0: NVIDIA GeForce RTX 3090	24.0
GPU 1: NVIDIA GeForce GTX 1080 Ti	11.0
GPU 2: NVIDIA GeForce GTX 1080 Ti	11.0
GPU 3: NVIDIA GeForce GTX 1080 Ti	11.0
GPU 4: NVIDIA GeForce GTX 1080 Ti	11.0
GPU 5: NVIDIA GeForce GTX 1080 Ti	11.0
GPU 6: NVIDIA GeForce GTX 1080 Ti	11.0

Tabulka 3.1 Informace GPU na nodu 1. Tato tabulka uvádí dostupné GPU a jejich celkovou paměť v GiB.

Název GPU	Celková paměť (GiB)
GPU 0: NVIDIA GeForce RTX 2080 Ti	11.0
GPU 1: NVIDIA GeForce RTX 2080 Ti	11.0
GPU 2: NVIDIA GeForce RTX 2080 Ti	11.0
GPU 3: NVIDIA GeForce RTX 2080 Ti	11.0
GPU 4: NVIDIA GeForce RTX 2080 Ti	11.0
GPU 5: NVIDIA GeForce RTX 2080 Ti	11.0
GPU 6: NVIDIA GeForce RTX 2080 Ti	11.0
GPU 7: NVIDIA GeForce RTX 2080 Ti	11.0

Tabulka 3.2 Informace GPU na nodu 2. Tato tabulka uvádí dostupné GPU a jejich celkovou paměť v GiB.

3.1.1 SLURM Workload Management

Na studentském clusteru je používán SLURM[3] (Simple Linux Utility for Resource Management) jako systém pro správu zatížení. SLURM je otevřený software, který slouží jako efektivní a flexibilní nástroj pro správu pracovních úloh a zdrojů na clusteru.

Princip SLURM spočívá v alokaci zdrojů a plánování úloh na clusteru. Umožňuje uživatelům odesílat a spravovat úlohy (joby) na clusteru, které mají rezervovaný předem specifikované výpočetní prostředky. SLURM poskytuje mechanismy

[3] `https://slurm.schedmd.com/overview.html`

pro rezervaci zdrojů, plánování úloh, monitorování stavu úloh a správu front úloh.

Pro spuštění úlohy na clusteru s použitím SLURM uživatelé vytvářejí skripty, které definují požadavky na zdroje (například počet CPU, množství paměti RAM, typ a počet GPU) a příkazy nebo skripty, které mají být provedeny. Tyto skripty jsou poté odeslány do systému SLURM pomocí příkazu `sbatch`. Po odeslání skriptu SLURM automaticky zařadí úlohu do fronty a po alokaci potřebných zdrojů spustí úlohu na clusteru. Zde je příklad takového skriptu.

```
#!/bin/bash

#SBATCH --job-name=python_gpu_job    # Název úlohy
#SBATCH --output=result_%j.txt       # Výstupní soubor (stdout)
#SBATCH --error=error_%j.txt         # Soubor s chybami (stderr)
#SBATCH --cpus-per-task=4            # Počet CPU na úlohu
#SBATCH --mem=64GB                   # Požadovaná paměť
#SBATCH --time=1:00:00               # Časový limit: 1 hodina
#SBATCH --gres=gpu:2                 # Požadavek na 2 GPU

# Spuštění Python skriptu
python3 /path/to/your/script.py
```

SLURM také nabízí nástroje pro monitorování stavu odeslaných úloh a správu fronty. Příkazy jako `squeue` pro zobrazení stavu fronty a `scontrol` pro zobrazení informací o odeslaných úlohách umožňují uživatelům sledovat průběh a výsledky svých výpočtů.

3.2 Webová aplikace

V rámci této bakalářské práce jsem vytvořil webovou aplikaci, jejímž cílem je usnadnit proces evidování a správy různých aspektů spojených s trénováním a evaluací jazykových modelů. Aplikace je součástí zdrojových kódů přiložených k této práci A. Aplikace umožňuje spravovat:

- E-shopy, ze kterých je možné stahovat data o produktech.
- Data stažená z eshopů.
- Datasety připravené pro trénování a inferenci.
- Vygenerované popisky produktů.

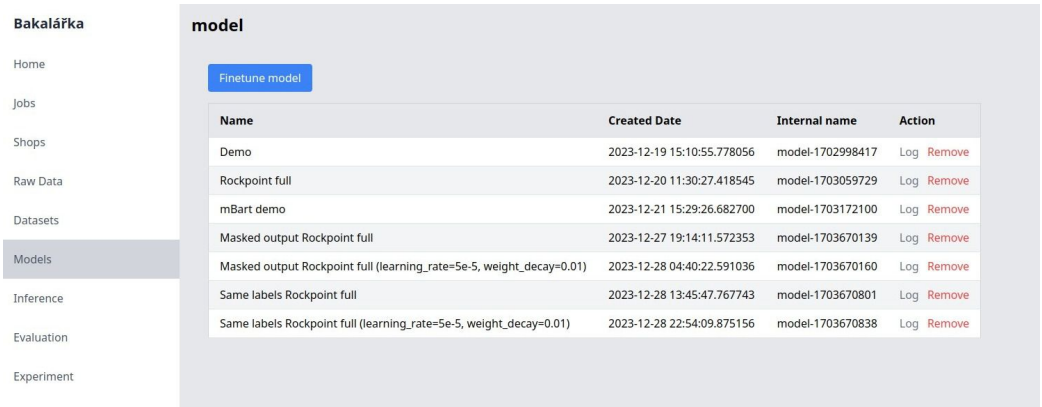
- Výsledky vyhodnocených experimentů, hodnotících kvalitu generovaných popisků.
- SLURM joby spuštěné z webové aplikace.

Dále aplikace poskytuje funkce pro spouštění různých skriptů, které jsou spouštěny prostřednictvím SLURM jobů na výpočetním clusteru. Uživatelé mohou v reálném čase sledovat výstup těchto jobů přímo v uživatelském rozhraní aplikace, což značně usnadňuje monitorování a správu výpočetních úloh. Tyto skripty zahrnují:

- Stažení dat z e-shopu.
- Přípravu datasetu.
- Spuštění fine-tuningu modelu.
- Spuštění inference na natrénovaných modelech.
- Evaluaci automatických metrik pro generované popisky.
- Vytvoření experimentů pro hodnocení generovaných popisků lidskými anotátory.

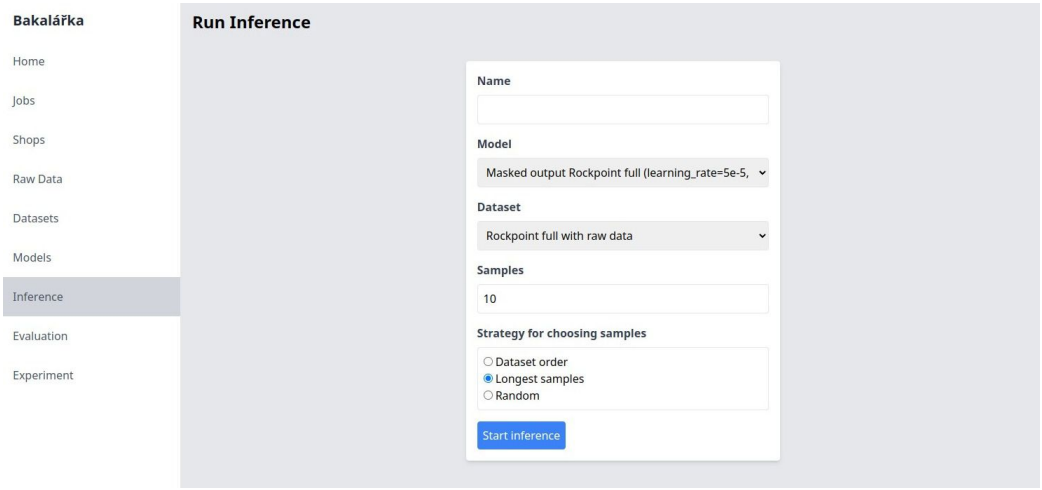
Screenshoty aplikace

Seznam natrénovaných modelů, který lze vidět na Obrázku 3.1, poskytuje funkce pro mazání modelů a zobrazení logů SLURM jobů, ve kterých byly modely trénovány. Tento přístup je implementován i pro ostatní evidované entity v aplikaci.



Obrázek 3.1 Seznam natrénovaných modelů

Jak ukazuje Obrázek 3.2, formulář umožňuje spouštět inferenci modelů nad vybranými datasety. Po odeslání formuláře je úloha odeslána do SLURM clusteru a skript pro inferenci je spuštěn, přičemž uživatel je přesměrován na stránku s živým výstupem z daného jobu. Tento mechanismus je aplikován i pro další skripty spouštěné na clusteru.



Obrázek 3.2 Formulář pro spuštění inference pro vybraný model a dataset.

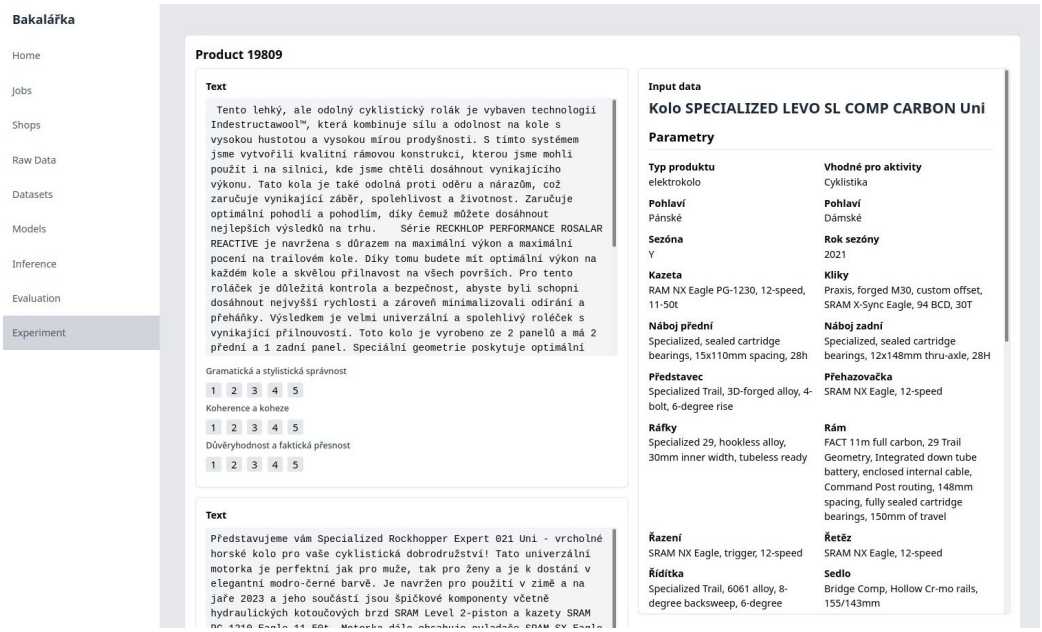
Na Obrázku 3.3 je zobrazeno rozhraní pro hodnocení vygenerovaných popisků anotátory. Popisky jsou zobrazeny v náhodném pořadí pro zachování anonymizace modelů, zatímco na pravé straně jsou prezentována vstupní data, která slouží k ověření faktické přesnosti popisků.

Obrázek 3.4 zobrazuje výstup ze SLURM jobu, který je aktualizován v reálném čase. Ze scriptu, který běží v jobu, je možné posílat speciální příkazy, které interaktivně ovlivňují uživatelské rozhraní. Příkladem jsou real-time aktualizace progress baru a grafu, které jsou zobrazeny na tomto obrázku.

3.2.1 Technologie

V této sekci představím technologie, které byly použity při vývoji webové aplikace. Aplikace je založena na mikro webovém frameworku Flask, napsaném v jazyce Python. Flask v základu neposkytuje mnoho funkcionalit. Podporuje však rozšíření, pomocí kterých je možné framework dovybavit funkcemi, které bychom očekávali od plnohodnotného webového frameworku.

Jako databázový systém byl zvolen SQLite, což je lehká relační databáze, která na rozdíl od běžných databází nepotřebuje samostatný serverový proces. Jedná se pouze o relativně malou knihovnu, která se stará o komunikaci mezi aplikací a jedním souborem na disku, ve kterém je celá databáze uložena.



Obrázek 3.3 Rozhraní pro hodnocení vygenerovaných textů anotátory.

Flask jsem rozšířil o několik balíčků, mezi nejdůležitější patří:

Flask SQLAlchemy: Tento balíček poskytuje podporu pro SQLAlchemy, což je ORM (Object-Relational Mapping) nástroj pro Python. Umožňuje efektivní a bezpečnou manipulaci s databází pomocí Python objektů a dotazů napsaných pomocí volání funkcí na těchto objektech.

Flask Migrate: Flask Migrate je rozšíření, které poskytuje podporu pro migrace databází. Umožňuje snadno provádět změny v databázových schématech pomocí Python kódu a sledovat historii těchto změn.

Flask WTF (WTForms): Toto rozšíření poskytuje integraci s WTForms, knihovnou pro generování a validaci formulářů ve Flasku. Umožňuje snadnou tvorbu formulářů a validaci uživatelských vstupů, což je nezbytné pro interaktivní aspekty webové aplikace.

Flask SocketIO: Flask SocketIO poskytuje Flask aplikacím podporu pro WebSockets, což umožňuje real-time komunikaci mezi klientem a serverem. Je to klíčové pro funkce, jako je zobrazování výstupu SLURM jobů v reálném čase.

3.2.2 Architektura

Aplikace je rozdělena do několika modulů. Níže je prezentována zjednodušená struktura adresářů aplikace s popisem jednotlivých modulů:

```
.
|-- config
|-- data
|-- dataCreation
|   |-- formatters
|   +-- loaders
|-- entities
|-- finetuning
|   |-- callbacks
|   |-- dataCollator
|   +-- tokenizerStrategy
|-- jobs
|-- migrations
|-- pages
|   |-- dataLoading
|   |-- dataset
⋮   ⋮
|   +-- model
|-- static
|-- templates
|   +-- pages
+-- utils
```

config Obsahuje konfigurační soubory a nastavení aplikace. Nachází se zde nastavení Flasku, propojení s databází, nastavení WebSocketu a některé globální prvky jako struktura hlavního menu.

data Slouží pro ukládání používaných dat aplikace (data stažená z e-shopu, natrénované modely, výsledky experimentů, atd.).

dataCreation Zahrnuje nástroje pro formátování a načítání dat, které jsou nezbytné pro přípravu a práci s datasety.

entities Definuje entity používané v aplikaci. Jednotlivé třídy v tomto modulu definují strukturu databáze.

finetuning Obsahuje moduly související s fine-tuningem modelů.

- Callbacks obsahuje callbacky používané při trénování, které se starají o posílání příkazů do webového rozhraní a například aktualizují progress-bar.
- DataCollator obsahuje různé přístupy k vytváření dávek (batches) během tréninku.
- TokenizerStrategy obsahuje různé přístupy k tokenizaci dat před trénováním.

jobs Zahrnuje skripty a definice úloh (jobs), které se spouštějí na výpočetním clusteru.

migrations Obsahuje skripty pro migraci databáze.

pages Obsahuje definice a logiku jednotlivých stránek webového rozhraní.

static Skladuje statické soubory, jako jsou CSS a JavaScript, které jsou potřebné pro frontend aplikace.

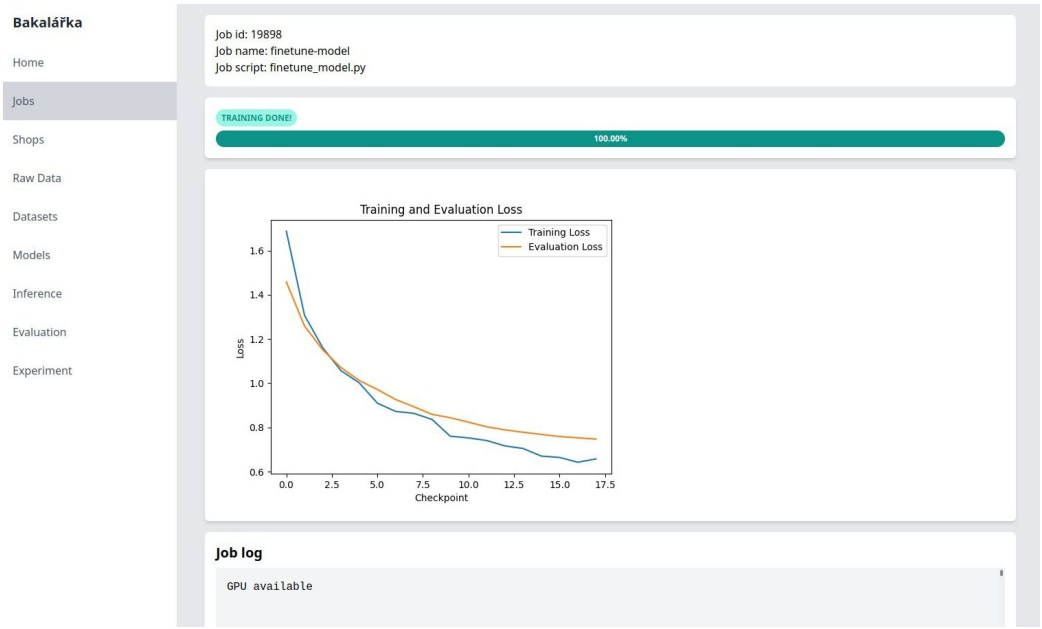
templates Obsahuje šablony pro stránky, které definují strukturu a vzhled uživatelského rozhraní.

utils Poskytuje pomocné nástroje a funkce pro různé účely v rámci aplikace.

3.2.3 Slurm joby

Jelikož pro spouštění SLURM jobů přímo z Python kódu neexistuje nativní podpora, je nutné skript odeslat do clusteru pomocí příkazu `sbatch`, stejně jako z terminálu. Výstup z jobu je poté ukládán do předem specifikovaného souboru. Pro zjednodušení spouštění a monitorování jobů jsem vytvořil mechanismus, který se o tyto úkony stará. Spustit úlohu na clusteru je tak možné pomocí několika málo řádků kódu (viz. Kód 1).

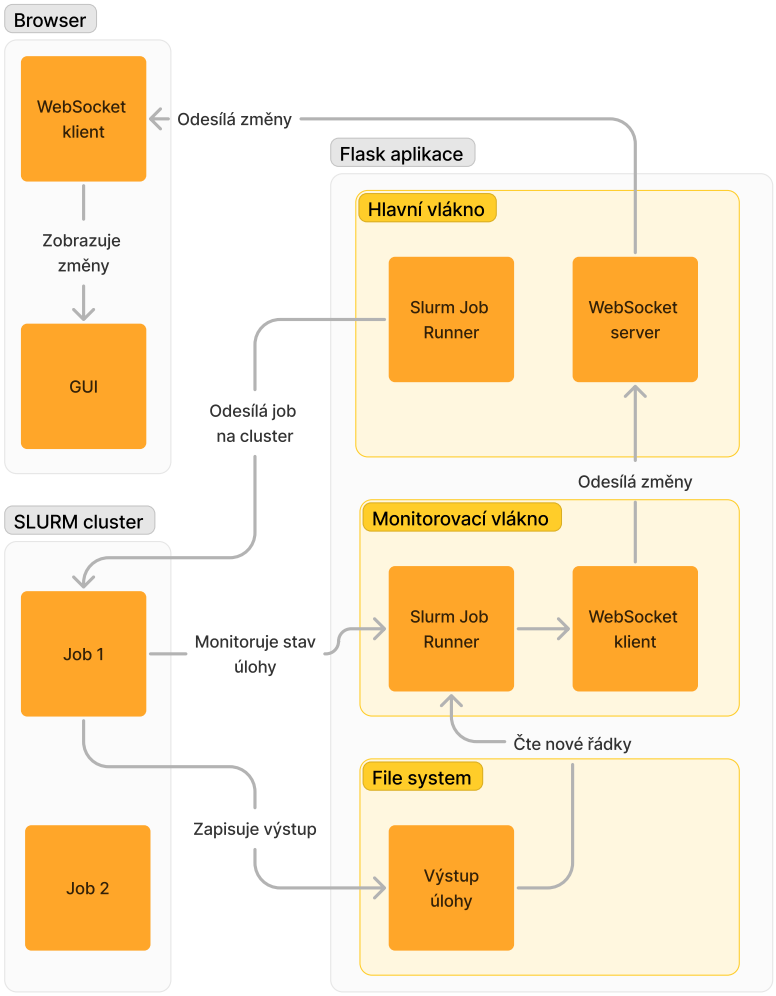
Tento kód vytvoří `SlurmJobRunner` objekt, který se stará o odeslání skriptu se správnými argumenty na cluster. Při vytváření objektu je možné specifikovat požadované zdroje jako GPU, CPU nebo paměť. Následně je pomocí metody `submit_job` úloha odeslána na cluster a zařazena do fronty pro spuštění. Nakonec `start_job_monitoring` spustí samostatné vlákno, které se stará o monitorování stavu úlohy, čtení výstupního souboru a odesílání změn na frontend pomocí WebSocketu. Nákres 3.5 názorně ukazuje celý mechanismus.



Obrázek 3.4 Výstup z SLURM jobu s interaktivními prvky.

Výpis kódu 1 Ukázka spuštění SLURM jobu z Python kódu

```
job = SlurmJobRunner(
    job_name='finetune-model',
    job_script='finetune_model.py',
    gpus=1
)
job_id = job.submit_job(
    form.dataset.data,
    form.name.data,
    f'model-{int(time.time())}'
)
job.start_job_monitoring()
```



Obrázek 3.5 Struktura spouštění SLURM jobů z Python kódu

3.3 Příprava dat

Příprava dat představuje klíčovou fázi ve vývoji modelů strojového učení, včetně těch zaměřených na zpracování přirozeného jazyka. Kvalita a forma dat mají přímý vliv na účinnost a spolehlivost výsledného modelu. Tato kapitola popisuje procesy a kroky, které byly provedeny pro přípravu dat před trénováním modelů, včetně získávání, čištění a formátování dat, a následné tvorby datasetů.

3.3.1 Proces získávání dat z e-shopů

Data pro trénování modelů byla získávána z e-shopů postavených na platformě WPJShop[4]. Ta poskytuje GraphQL API, ze kterého je možné načíst data o produktech. Ke stahování slouží třída `WPJDataLoader`, která obsahuje metodu `get_data_generator`. Tato metoda vrací generátor, který poskytuje jednotlivé produkty jako obyčejné slovníky. Třída se stará o komunikaci s API. Komunikace probíhá po blocích o velikosti 100 produktů, což zajišťuje efektivní přenos dat. Ve webové aplikaci je tato třída použita pro vytváření JSON souborů s produkty.

3.3.2 Čištění a formátování dat

Data načtená z e-shopu jsou strukturovaná ve formátu JSON. Tento formát není ideálním vstupem pro model, jelikož obsahuje relativně velké množství znaků, které hrají pouze strukturální roli. Navíc při tokenizaci je většina režijního textu tokenizována po znacích, což vede ke zbytečnému zvětšování vstupů. Jelikož jsou modely, které je možné spustit na dostupném hardware, relativně malé, v některých případech je nutné příliš dlouhé vstupy oříznout, a režijní tokeny tak zabírají místo sémanticky důležitým.

Bylo tedy nutné strukturovaná data zpracovat do formátu vhodného pro jazykové modely. V úvahu přicházely dvě varianty.

[4] `https://www.wpj.cz/`

1. **Oddělení dat pomocí speciálního znaku** Takto formátovaný text by ze vstupních dat vytvořil jeden řetězec, ve kterém by byly jednotlivé parametry vstupních dat odděleny nějakým speciálním znakem, například `|`. Výsledný text by vypadal následovně:

```
název produktu|krátký popis|
parametr_1: hodnota parametru_1|...|
parametr_n: hodnota parametru_n|
jmenovka varianty_1: název varianty_1|...|
jmenovka varianty_n: název varianty_n|
ukázkový popis
```

   Tento přístup nakonec nebyl použit, jelikož při prvních pokusech trénování modelu vracel text, který byl úplně nečitelný. U druhého přístupu k takovému chování nedošlo a kvůli úspoře času a omezeným výpočetním kapacitám nebyl tento přístup dále zkoušen.

2. **Vytvoření čitelné věty** Tento přístup je podobný prvnímu. Rozdíl je v tom, že jsou vstupní data oddělována pomocí slov volených tak, aby se výsledný text dal přečíst jako jeden smysluplný kus textu.

```
Produkt {název produktu}, s krátkým
popisem: {krátký popis} s parametry:
{parametr_1}: {hodnota parametru_1}, ...,
{parametr_n}: {hodnota parametru_n} a variantami
{jmenovka varianty_1}: {název varianty_1}, ...,
{jmenovka varianty_n}: {název varianty_n} má
popis: {ukázkový popis}
```

Transformace strukturovaných dat do textu nebyla jedinou úpravou. Ukázkové popisy často obsahují nějaké HTML a jiné speciální znaky. Tento nadbytečný text zabírá, stejně jako režijní znaky JSONu, místo ve vstupu pro model. Před použitím dat tedy došlo k jejich odstranění.

O celý proces se stará třída `ReadableStringFormatter`. Ta pro své fungování potřebuje nějaký data loader, ze kterého čerpá data. Aplikace obsahuje, kromě data loaderu pro načítání dat z API zmíněného v sekci 3.3.1, ještě `DiskDataLoader`, který načítá data z JSON souboru uloženého na disku, a `DirectDataLoader`, který načítá data z paměti.

Aplikace obsahuje ještě několik dalších formatterů, které mají stejné API, ale jsou používány pouze pro účely webové aplikace.

3.3.3 Tvorba datasetu

Pro předzpracování dat pro trénování byla použita knihovna Transformers [20], která usnadňuje používání a fine-tuning předtrénovaných modelů nejen pro NLP úlohy. Tato knihovna používá pro interakci s modely datasety z knihovny Datasets.

Z webové aplikace je možné spustit skript, který tento dataset z naformátovaných a očištěných dat vytvoří. Tento skript vytvoří kolekci datasetů, z nichž každý slouží k jinému účelu.

train Obsahuje 80 % vstupních dat. Slouží k fine-tuningu modelu.

validation Obsahuje 10 % vstupních dat. Používá se v průběhu trénování pro výběr nejlepšího checkpointu.

test Obsahuje 10 % vstupních dat. Používá se pro finální hodnocení modelu automatickými metrikami a lidskými anotátory.

3.4 Fine-tuning modelu

Tato sekce se zabývá specifikami procesu fine-tuningu, který přizpůsobuje předtrénovaný model datům připraveným v sekci 3.3. Zahrnuje popis tokenizace, tvorby dávek (anglicky batches) a výběru hyperparametrů pro trénování. To jsou kroky, které je nezbytné udělat před samotným trénováním modelu.

K samotnému trénování je pak použita již zmíněná knihovna Transformers. Ta umožňuje spustit proces trénování pomocí několika málo řádek kódu (viz. Kód 2).

3.4.1 Tokenizace

Tokenizace je proces, při kterém se vstupní text převede do numerické reprezentace, která slouží jako vstup pro model. Je dán slovník předem připravených částí slov, tzv. subwordů, z nichž každý má svůj identifikátor (token id). Vstupní text je rozdělen na tyto části a převeden do pole tokenů.

Způsob, jakým jsou části vybírány a jaké tokeny jsou pro ně použity, se pro různé modely liší. Je ovšem důležité, aby při fine-tuningu modelu byl použit stejný mechanismus jako při prvotním trénování. Knihovna Transformers tuto konzistenci zajišťuje pomocí třídy `AutoTokenizer`, která umožňuje použít odpovídající tokenizer pro konkrétní model.

Jelikož má model omezenou kapacitu, není možné tokenizovat celý text najednou. Je tedy nutné rozdělit ho do bloků, které maximální délku nepřesahují. Délkou vstupu je zde myšlen počet tokenů po tokenizaci.

Výpis kódu 2 Ukázka spuštění trénování modelu pomocí knihovny Transformers

```
model = AutoModelForCausalLM
            .from_pretrained(model_checkpoint)

training_args = TrainingArguments(
    output_dir=model_path,
    evaluation_strategy="epoch",
    num_train_epochs=1,
    learning_rate=2e-5,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["valid"],
)

trainer.train()
```

Model	Prům. tok. (CZ)	EN	Poměr tok. (CZ)	EN
spital/gpt2-small-czech-cs	172.22	191.45	0.3288	0.3947
facebook/mbart-large-50	156.75	135.93	0.2961	0.2793
bert-base-uncased	210.42	124.87	0.3977	0.2572

Tabulka 3.3 Srovnání tokenizátorů. Tato tabulka uvádí srovnání průměrného počtu tokenů a poměru tokenů vůči netokenizovanému textu (tzn. průměrný počet znaků na token) pro různé modely. Data byla vypočítána na 5768 textech, kde anglický text byl překladem českého.

Obecně je nevýhodou českého textu, že slova bývají častěji rozdělena do více tokenů, takže je průměrná délka vstupu pro text stejné délky vyšší, než v angličtině. Použitím modelu, jehož tokenizer byl trénován na českých datech, se můžeme této nevýhody zbavit nebo dokonce dosáhnout kratšího vstupu u českého textu (viz. Tabulka 3.3).

Připravený dataset již obsahuje rozdělení po produktech. Je ovšem potřeba se vypořádat s případy, kdy je připravený text o produktu delší než maximální délka modelu. To lze realizovat různými přístupy.

Oříznutí (truncation) V tomto případě je část tokenů, která přesahuje maximální délku modelu, oříznuta. Výhodou tohoto přístupu je jeho jednoduchost. Nevýhodou je, že model pro zvolený formát vstupního textu nevidí celý ukázkový popis.

Spojování textu Jako jednotlivé trénovací vstupy nejsou použity texty o jednotlivých produktech. Prvně jsou texty spojeny do jednoho dlouhého bloku textu, následně jsou tokenizovány a rozděleny do bloků o maximální délce. Tento přístup zachovává všechna data, ale pro model může být těžší odlišit jednotlivé produkty, jelikož výsledné bloky mohou být přerušeny například v půlce slova.

Posuvné okénko Pro vyvážení nevýhod ořezávání textu by bylo možné dlouhé případy rozdělit na více částí. Pokud by se to udělalo nejjednodušším způsobem, mohlo by stále být pro model těžké rozeznat jednotlivé produkty. Tomu by bylo možné předejít rozdělením na větší množství příkladů. Každý z nich by měl fixní délku a jeho začátek a konec by byl vždy posunut o fixní počet tokenů. Tím by se ovšem výrazně zvýšil počet příkladů v datasetu a prodloužilo by to trénování.

V práci byl zvolen první přístup kvůli jednoduchosti implementace.

Vypořádání se s příliš dlouhými vstupy není jediná věc, kterou je nutné před trénováním vyřešit. Při trénování model potřebuje, aby byly všechny vstupy stejně dlouhé. V případě spojování textu je tento problém téměř vyřešen, ale u ostatních je třeba délku dorovnat. To se řeší přidáním speciálních tokenů (padding) na konec kratších příkladů. Tyto tokeny nehrají při trénování žádnou roli.

Aby model poznal, jestli má mít daný token nějaký význam, tak při tokenizaci vzniká ještě jedno pole zvané attention mask. To má délku rovnou počtu tokenů a obsahuje 1 na místě, kde je token, který má model považovat za vstup, a 0 na místě, které má být ignorováno.

3.4.2 Data collator

Při trénování model běžně neprochází příklady jednotlivě, ale v dávkách, které obsahují více příkladů najednou. Počet příkladů v dávce je omezen hlavně kapacitou paměti grafických karet, na kterých je model trénován. V případě GPU dostupných na studentském clusteru se ovšem do paměti vešly maximálně dávky o velikosti 2.

O přípravu dávek se stará třída běžně nazývaná data collator. Tato třída může provést finální úpravy dávek pro model. Například zde může probíhat přidávání paddingu.

Pro správnou funkci trénovacího cyklu je potřeba, aby dataset kromě tokenizovaného vstupu a attention mask obsahoval ještě štítky (labels). To jsou tokeny, se kterými model porovnává svoje predikce, na základě kterých počítá loss funkci, jejíž výsledek slouží k upravování vah modelu. Přidání těchto štítků je jednou z rolí data collatoru.

Štítky je možné opět tvořit více způsoby. V této práci byly vyzkoušeny dva přístupy.

Stejné štítky Tento přístup je jednoduchý. Jedná se pouze o zkopírování vstupních tokenů. Jeho nevýhodou je, že se model učí modelovat i vstupní tokeny, což nadbytečně zatěžkává kapacitu modelu.

Maskování vstupu Část textu, která obsahuje vstupní data jako jméno produktu a jeho varianty, dostane štítek s hodnotou -100[5]. To je speciální hodnota, která modelu říká, že v průběhu počítání loss funkce nemá brát tyto tokeny v úvahu. Tím je zajištěno, že model upravuje svoje váhy pouze na základě vygenerovaných popisků.

3.4.3 Argumenty pro trénování

Proces trénování je možné ovlivnit několika argumenty, které mají vliv na výsledek.

počet epoch Určuje, kolikrát model během trénování uvidí každý příklad. Během jedné epochy projde modelem každý příklad z trénovacího datasetu jednou. Pokud je epoch hodně, tak to může vést k tomu, že se model naučí specificky na trénovací data, ale na datech, které neviděl, nemá dobré výsledky. Tomuto stavu se říká nadměrné přizpůsobení (overfitting). Větší počet epoch má také zásadní vliv na délku trénování.

velikost dávky (batch size) Určuje, kolik příkladů najednou model zpracuje během jednoho kroku. Větší dávky urychlují trénování, ale zároveň zvyšují nároky na paměť. Batch size je tedy ideální volit co největší, ale zároveň tak, aby nedošlo k vyčerpání paměti.

learning rate Určuje, jak velké změny v nastavení vah bude model provádět po každém kroku. Moc malá learning rate znamená, že se model bude učit pomalu a ani po velkém počtu kroků se nemusí výrazně zlepšit. Naopak moc velká learning rate může způsobit, že model bude skákat od extrému k extrému a nikdy se mu nepodaří najít optimum.

ostatní režijní parametry Třída `Trainer` z knihovny transformers umožňuje nastavit ještě řadu parametrů, které nemají přímý vliv na trénování. Mezi tyto parametry patří:

- Adresář, do kterého bude model uložen.

[5] `https://huggingface.co/docs/transformers/main/en/internal/modeling_utils#transformers.modeling_tf_utils.TFCausalLanguageModelingLoss`

- Volba strategie pro hodnocení modelu a výběr nejlepšího modelu. Model je průběžně testován na validačním datasetu a pokud by začalo docházet k overfittingu, tak je vybrán model, který dosáhl nejlepších výsledků na validačním datasetu.
- Kolik modelů má být během trénování uloženo. Během trénování je možno ukládat tzv. checkpointy. To je stav modelu po předem zvoleném počtu kroků nebo epoch.
- Způsob reportování průběžných výsledků a jak často se to má dít. Výchozí volbou je progress bar, který vypisuje hlášení na standardní výstup. Tuto volbu jsem vypnul a upravil tak, aby posílala příkazy webové aplikaci.

Můj trénovací dataset měl velikost 46146 produktů. Pro trénování jsem zvolil velikost dávky 2, jelikož to bylo maximum, co se vešlo do paměti dostupného GPU. Počet epoch jsem pro finální experimenty zvolil 4. S tímto nastavením trval fine-tuning jednoho modelu přibližně 12 hodin. Learning rate jsem původně nechal na výchozí hodnotě 5e-5. Jelikož během tréninku nezačalo docházet k overfittingu a trénovací i validační loss neustále klesal (viz graf 3.6), tak jsem pro další pokusy zvýšil learning rate na 1e-3.

3.5 Použití velkých předtrénovaných modelů

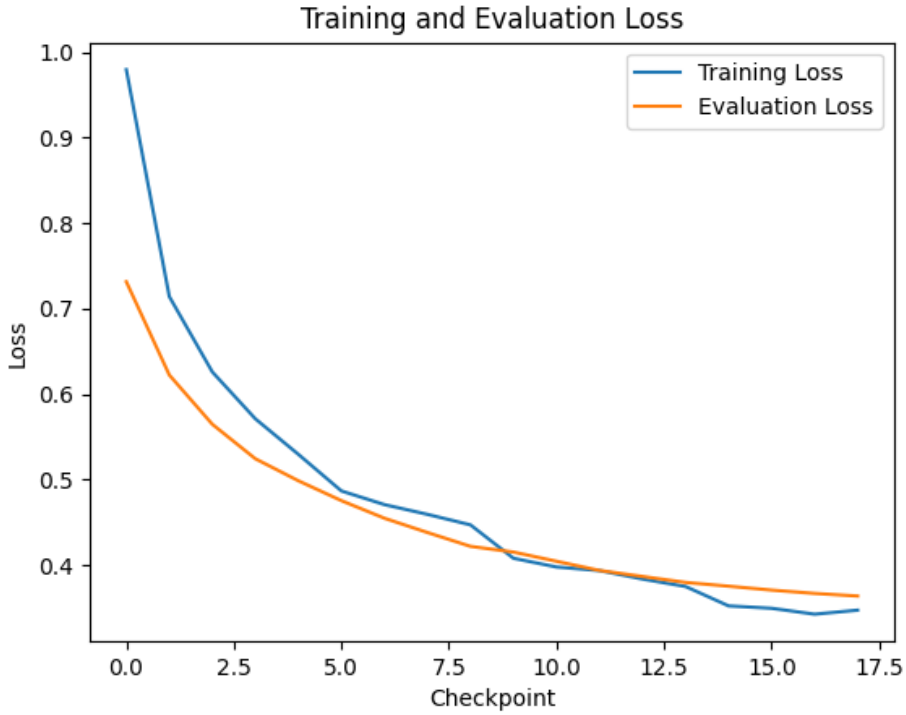
Pro porovnání s modely dotrénovanými na datech konkrétního e-shopu byly provedeny experimenty pomocí LLM takzvaným zero-shot přístupem. Tento přístup spočívá v tom, že model není nijak trénován na konkrétních datech a je použit pouze k vygenerování výsledků.

Jelikož jsou tyto modely příliš velké na to, aby bylo možné je spustit na dostupném hardware, přistupuji k nim pouze pomocí API. Pro porovnání jsem zvolil dva takové modely: Mistral (mistralai/Mistral-7B-Instruct-v0.1[6]) a ChatGPT (gpt-3.5-turbo-1106).

3.5.1 Vstup pro modely

Vzhledem k tomu, že tyto modely nejsou trénovány na našich datech, je nutné jim poskytnout instrukce, ve kterých popíšeme, co od nich očekáváme. Těmto

[6] `https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.1`



Obrázek 3.6 Vývoj loss funkce během trénování modelu s výchozí learning rate.

instrukcím se říká prompt. Pro oba modely jsem zvolil stejnou strukturu.

```
Níže jsou informace o produktu. Na základě těchto informací
vygeneruj popis produktu vhodný pro použití na e-shopu.
‘‘‘
{Text s informacemi, který byl použit při finetuningu
bez vzorového popisu (tzn. končící „má popis:"}
‘‘‘
```

3.6 Mistral

Mistral produkuje nejlepší výsledky v angličtině, proto jsem před jeho použitím přeložil prompt do angličtiny. K překladu jsem použil překladač vyvinutý na ÚFAL MFF UK, který používá model CUBBITT [21] a je hostovaný na infrastruktuře[7]. Vygenerovaný text je v angličtině, a proto jsem ho po vygenerování přeložil zpět do češtiny. Celý proces je znázorněn na diagramu 3.7.

Model mi byl zpřístupněný pomocí API, které je postaveno pomocí Text generation web UI[8]. Model běží na výpočetním clusteru LRC (Linguistic Research Cluster)[9]

3.7 ChatGPT

Pro vstup v češtině dokáže ChatGPT produkovat obstojný výstup. Proto pro jeho využití nebylo potřeba vstup překládat do angličtiny. Vygenerované výsledky pro český vstup jsou též v češtině, takže použití tohoto modelu je z implementačního hlediska velice jednoduché. Stačí pouze použít API[10], na které se pošle připravený prompt (viz 3.5.1) a jako výsledek se vrátí vygenerovaný text.

Použití API je zpoplatněno. Platí se za počet vstupních i výstupních tokenů. Pro verzi gpt-3.5-turbo-1106, kterou jsem používal, je cena 0.001 USD za 1000 vstupních tokenů a 0.002 USD za 1000 výstupních tokenů. Pro zjištění počtu tokenů je možné použít knihovnu Tiktoken[11]. V průběhu experimentu jsem vygeneroval 5768 popisků produktu s průměrně 219 vstupními tokeny a 349 výstupními tokeny. To při kurzu 22.5 za dolar odpovídá 0.02 Kč za jeden vygenerovaný popis.

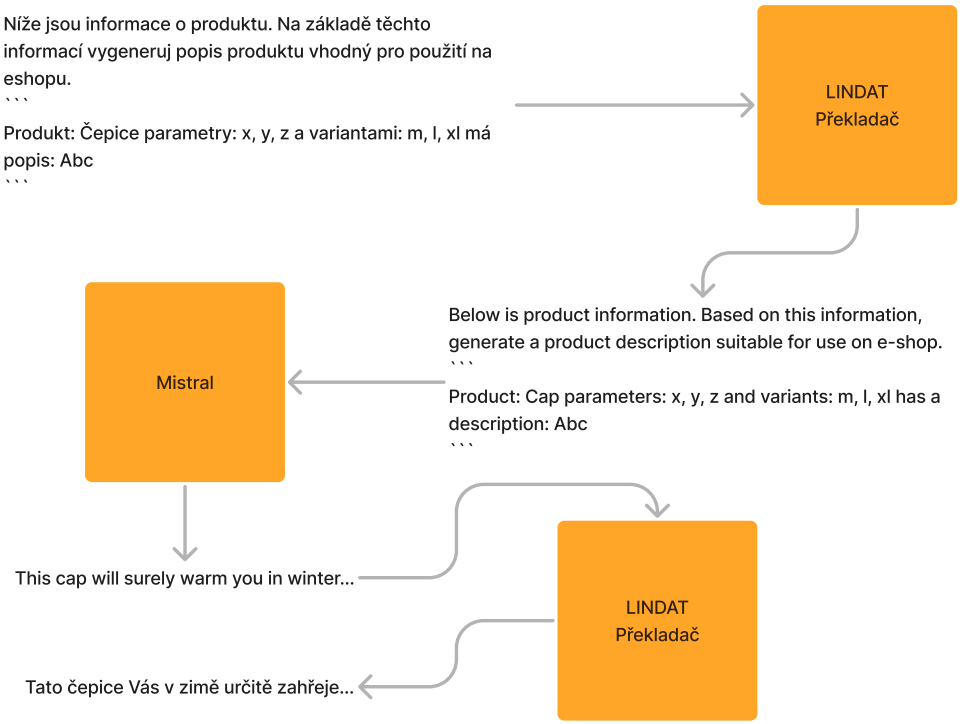
[7] https://lindat.mff.cuni.cz/

[8] https://github.com/oobabooga/text-generation-webui?tab=readme-ov-file#text-generation-web-ui

[9] https://wiki.ufal.ms.mff.cuni.cz/slurm

[10] https://platform.openai.com/docs/introduction

[11] https://github.com/openai/tiktoken



Obrázek 3.7 Pipelina pro použití modelu Mistral s překladem promptu do angličtiny a vygenerovaného textu zpět do češtiny.

Kapitola 4

Hodnocení kvality výsledků

Tato kapitola se věnuje testování a hodnocení kvality výsledků generovaných různými metodami. Kvalitativní posouzení generovaných textů je nezbytné pro porovnání efektivity různých přístupů.

4.1 Automatická evaluace

Automatizované metriky nám poskytují rychlé kvantitativní hodnocení kvality generovaných textů. V této sekci podrobněji popíšeme metriky použité pro hodnocení a srovnání různých metod generování textu.

4.1.1 Zvolené metriky

Pro vyhodnocení byli použity metriky popsané v sekci 2.3.1.

- BLEU
- METEOR
- ROUGE-1 (varianta ROUGE založena na porovnávání unigramů)
- ROUGE-L (varianta ROUGE založena na nejdelší společné podposloupnosti)

4.1.2 Výběr nejlepšího fine-tunovaného modelu

Hodnocení lidskými anotátory je náročný proces, proto bylo pro jejich hodnocení nutné vybrat nejlepší model, který byl dotrénován na datech z e-shopu. Anotátoři

tak srovnávali pouze tři rozdílné přístupy a ne různé dotrénované modely. K výběru nejlepšího dotrénovaného modelu byly použity pouze automatické metriky. K výpočtu metrik jsem použil knihovnu NLG Metricverse[1].

Do výběru se dostaly 4 fine-tunované modely (viz sekce 3.4.2 a 3.4.3).

1. Stejné štítky s výchozími hyperparametry
2. Maskované štítky s výchozími hyperparametry
3. Stejné štítky s learning rate 1e-3
4. Maskované štítky s learning rate 1e-3

Na základě vypočítaných metrik byl pro hodnocení lidskými anotátory zvolen model 2. Důvodem je, že měl největší skóre ve 3 ze 4 případů (viz grafy 4.1, 4.2, 4.3 a 4.4).

4.2 Lidští anotátoři

Lidští anotátoři hodnotili vygenerované popisky ve webové aplikaci. Na jedné stránce byly zobrazeny vždy 3 vygenerované popisky stejného produktu. Každý popisek byl z jiného přístupu ke generování.

1. ChatGPT
2. Mistral s překladem
3. Nejlepší fine-tunovaný model

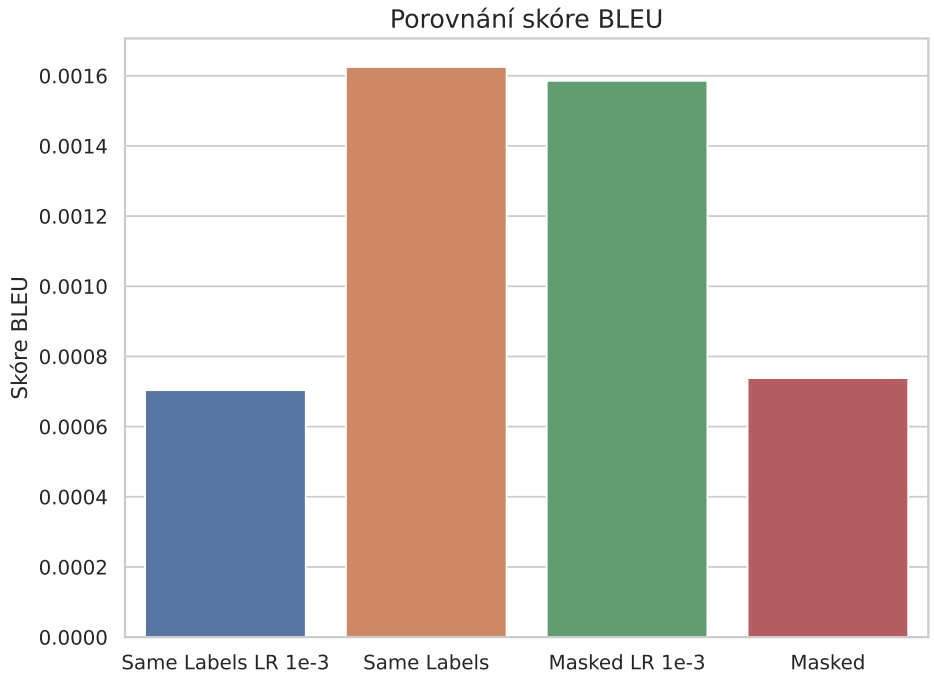
Tyto ukázky byly zobrazovány v náhodném pořadí, aby nebylo zřejmé, z jakého modelu text pochází, a hodnocení tak bylo co nejobjektivnější. Vedle popisků byly zobrazeny také vstupní data pro kontrolu přesnosti tvrzení v textu. Rozhraní, ve kterém anotátoři texty hodnotili je zobrazeno na obrázku 3.3.

4.2.1 Zvolené metriky

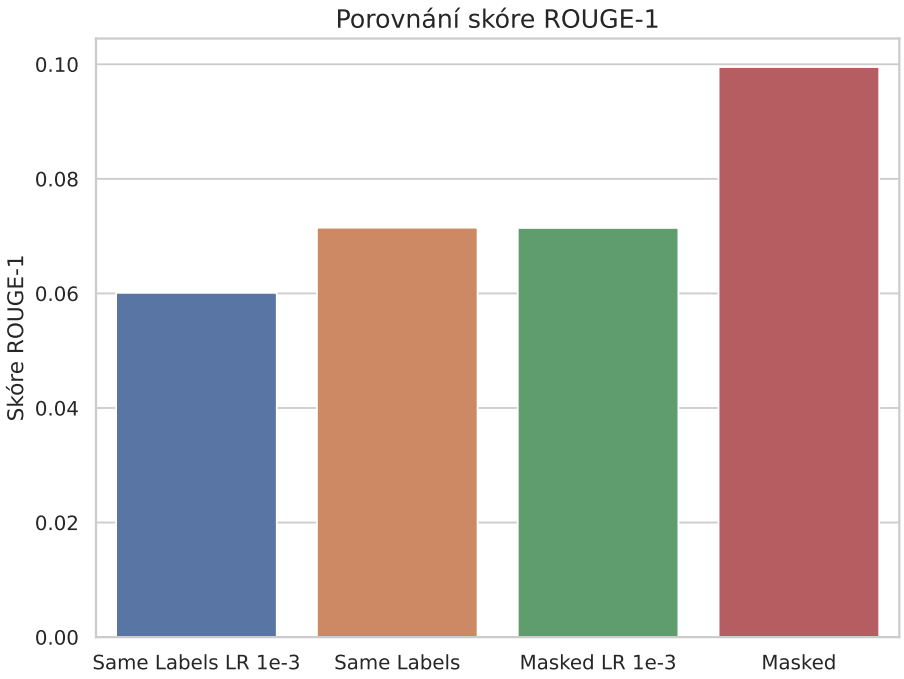
Lidští anotátoři hodnotili text pomocí následujících metrik[2]. Popis těchto metrik s významem jednotlivých stupňů škály měli anotátoři k dispozici po celou dobu hodnocení.

[1] `https://github.com/disi-unibo-nlp/nlg-metricverse`

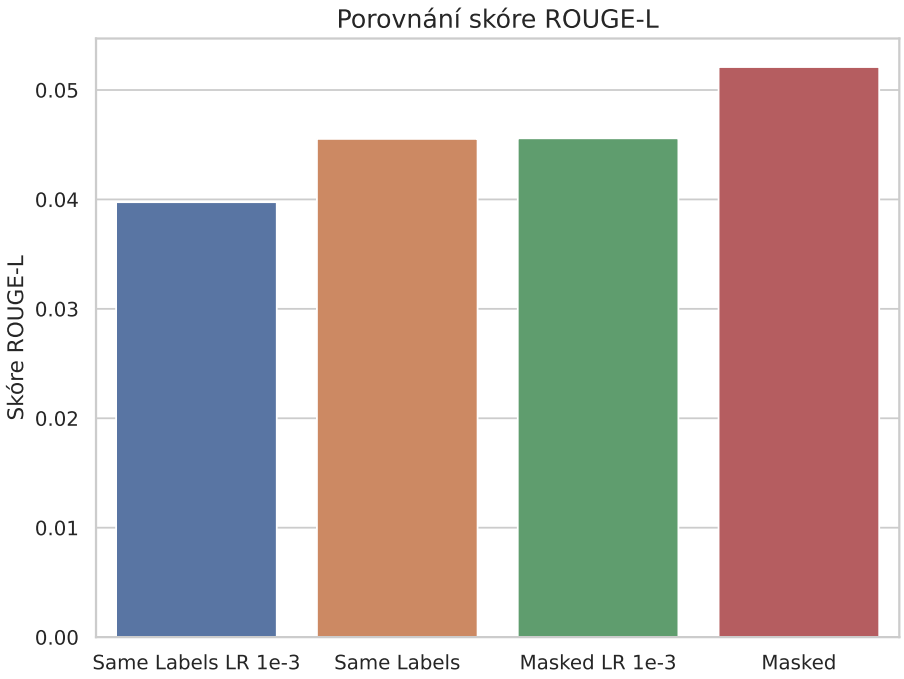
[2] Pro tvorbu definic jednotlivých bodů škály zvolených metrik byly použity nástroje umělé inteligence.



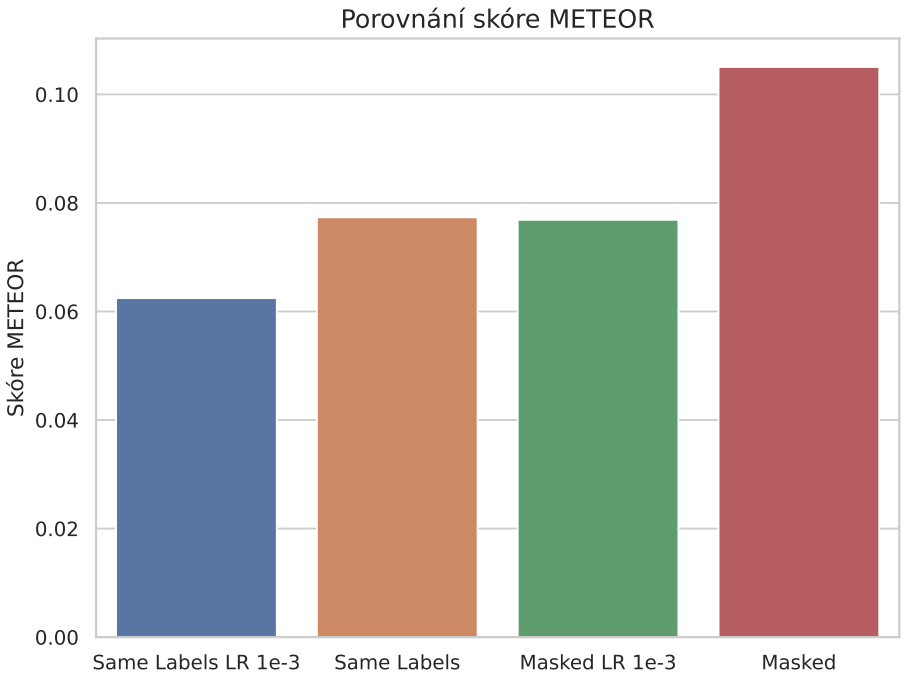
Obrázek 4.1 Porovnání BLEU skóre pro 4 fine-tunované modely



Obrázek 4.2 Porovnání skóre ROUGE-1 pro 4 fine-tunované modely



Obrázek 4.3 Porovnání skóre ROUGE-L pro 4 fine-tunované modely



Obrázek 4.4 Porovnání skóre METEOR pro 4 fine-tunované modely

Gramatická a Stylistická Správnost

Metrika *Gramatická a Stylistická Správnost* hodnotí, jak dobře text dodržuje pravidla gramatiky, pravopisu, interpunkce a stylistické konzistence.

1 bod: Velmi nízká Text obsahuje mnoho gramatických chyb a stylistických nedostatků. Styl je nekonzistentní a nevhodný pro daný kontext.

2 body: Nízká Text má mnoho gramatických chyb a stylistických nedostatků. Styl je nejednotný a často neodpovídá standardům pro daný druh textu.

3 body: Střední Text obsahuje několik gramatických nebo stylistických chyb, ale ty neovlivňují celkové porozumění. Styl je obecně vhodný pro daný kontext.

4 body: Vysoká Text je velmi dobře napsán s minimálními gramatickými nebo stylistickými chybami. Styl psaní je vhodný a konzistentní.

5 bodů: Velmi vysoká Text je gramaticky bezchybný a stylisticky vynikající. Styl psaní je přesně přizpůsobený kontextu a cílovému publiku.

Koherence a Koheze

Metrika *Koherence a Koheze* hodnotí, jak logicky a plynule text postupuje od jedné myšlenky k druhé. Zohledňuje logické spojení myšlenek, plynulost přechodů, celkovou srozumitelnost textu a schopnost vyhnout se nadbytečným informacím, které by mohly narušit kohezi.

1 bod: Velmi nízká Text je velmi fragmentovaný a nekonzistentní. Myšlenky jsou chaotické, často nedokončené, s náhodnými nebo nepřítomnými přechody, a obsahují nadbytečné informace.

2 body: Nízká Text obsahuje určitou míru logického spojení, ale je narušen nesrovnalostmi, nedostatky v logice a přechodech, a přítomností nadbytečných detailů.

3 body: Střední Text je obecně koherentní s několika logickými skoky nebo nepřesnostmi. Přechody mohou být někdy nejasné a mohou obsahovat několik nadbytečných informací.

4 body: Vysoká Text je dobře strukturovaný s logickými a plynulými přechody mezi myšlenkami. Obsahuje minimální nesrovnalosti a nadbytečné informace jsou omezené.

5 bodů: Velmi vysoká Text je vynikající v koherenci a kohezi s přirozenými přechody. Nabízí čtenáři jasný a ucelený narativ bez zbytečných detailů.

Důvěryhodnost a Faktická Přesnost

Metrika *Důvěryhodnost a Faktická Přesnost* hodnotí, nakolik jsou informace v textu pravdivé a založené vstupních datech.

1 bod: Velmi nízká Text obsahuje mnoho faktických chyb a zavádějících informací. Je těžce ověřitelný a vyžaduje rozsáhlé úpravy.

2 body: Nízká Text obsahuje několik faktických nepřesností, které vedou k nesprávnému pochopení tématu. Vyžaduje ověření a opravy.

3 body: Střední Většina informací je správná, ale existují menší chyby nebo nepřesnosti. Text je obecně spolehlivý, ale vyžaduje úpravy.

4 body: Vysoká Text je velmi spolehlivý s minimálními faktickými chybami. Informace jsou dobře založené na vstupních datech.

5 bodů: Velmi vysoká Text je bezchybný a důvěryhodný. Všechny informace přesně odpovídají vstupním datům.

4.3 Výsledky

Výsledky automatických metrik (viz grafy 4.5, 4.6, 4.7 a 4.8) i hodnocení lidských anotátorů (viz graf 4.9) ukazují celkem jednoznačné pořadí zvolených přístupů.

1. ChatGPT
2. Mistral
3. Fine-tunovaný GPT-2-small-cs

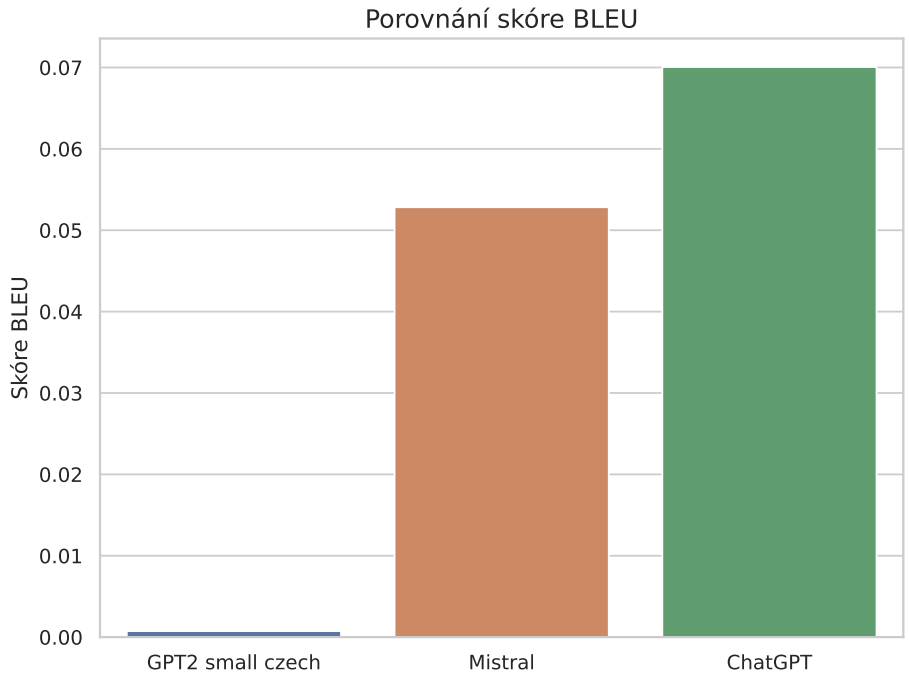
Co se týče automatických metrik, tak u BLEU skóre můžeme vidět téměř nulové hodnoty u fine-tunovaného modelu. U ostatních metrik jsou výsledky vyrovnanější.

Manuálního hodnocení se účastnilo dohromady 8 lidských anotátorů a ohodnoceno bylo 190 produktů. Výsledky vypadají na první pohled poměrně vyrovnaně, s tím, že velké jazykové modely (Mistral a ChatGPT) jsou na tom trochu lépe. Celkově jsou ale výsledné hodnoty okolo středu škály a žádná z hodnocených metrik zásadním způsobem nevybočuje.

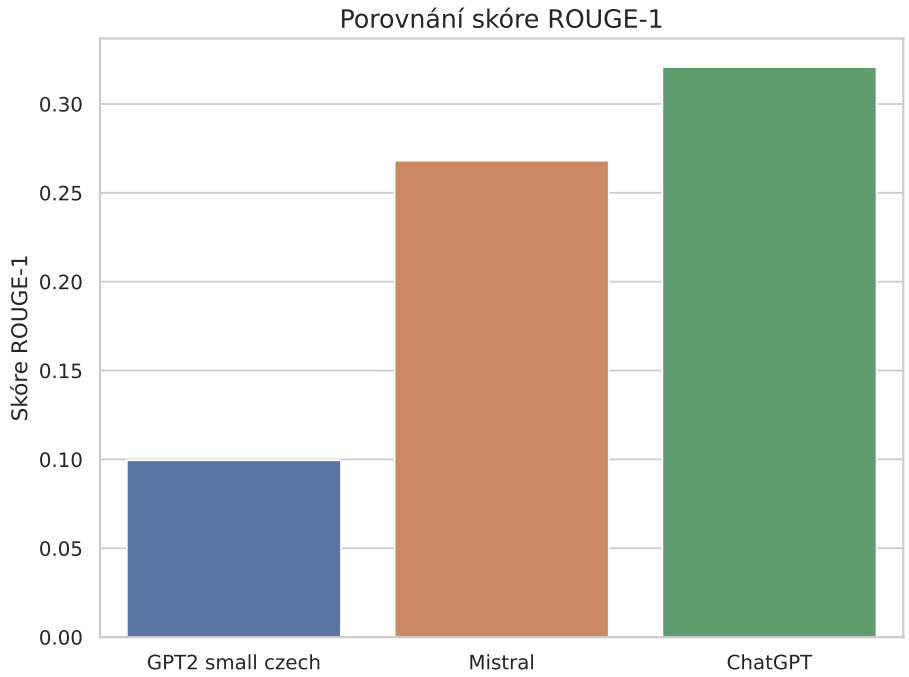
Subjektivně bych řekl, že vygenerované texty z velké většiny nejsou použitelné pro přímé použití na e-shopu. U produktů, které mají hodně vstupních parametrů (např. cyklistická kola), by byly texty s mírnou korekcí použitelné. U produktů s méně parametry však byly popisky často zavádějící, opakovaly se

stejné fráze nebo byly příliš obecné. Takovéto popisky vypadaly podobně jako by byly vytvořeny pomocí šablon, s rozdílem, že není zaručena faktická přesnost.

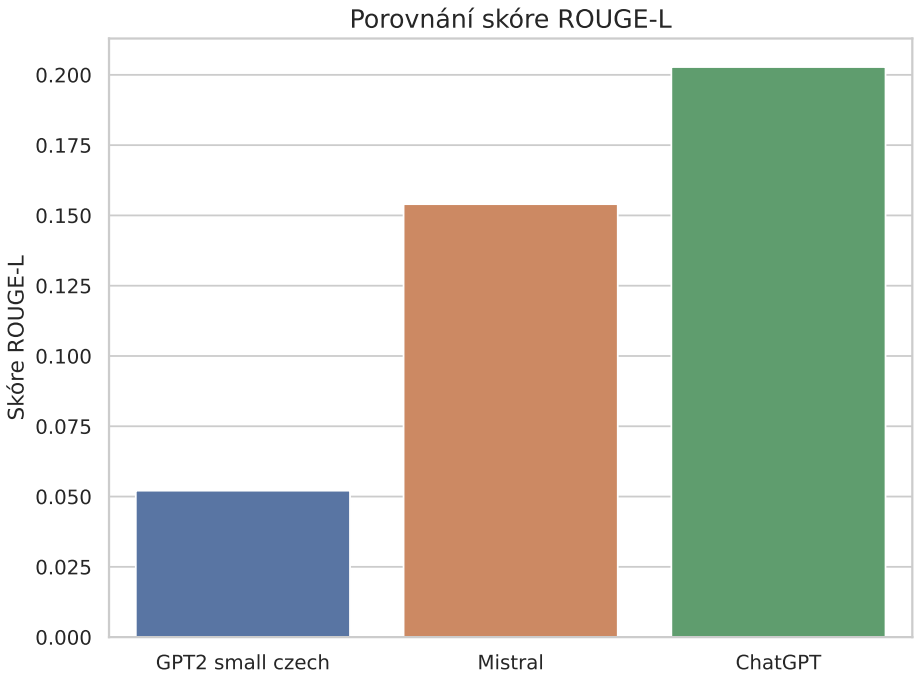
Nejlépe hodnocený byl ChatGPT, jehož využití má své úskalí. Jelikož se jedná o uzavřený model, tak není jasné, jak přesně funguje. Také je nutné posílat všechna data třetí straně, což může být v některých případech problematické kvůli ochraně firemních dat. Modely, které lze rozběhnout na dostupném hardware, na druhou stranu produkují v mnoha případech kompletně nepoužitelný text.



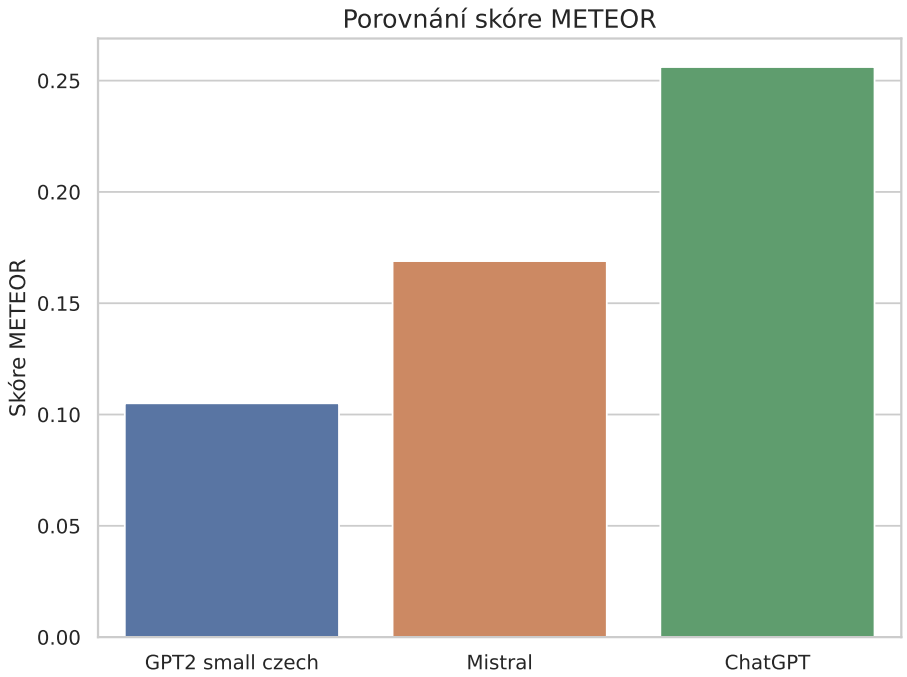
Obrázek 4.5 Porovnání BLEU skóre pro 3 různé přístupy ke generování popisků



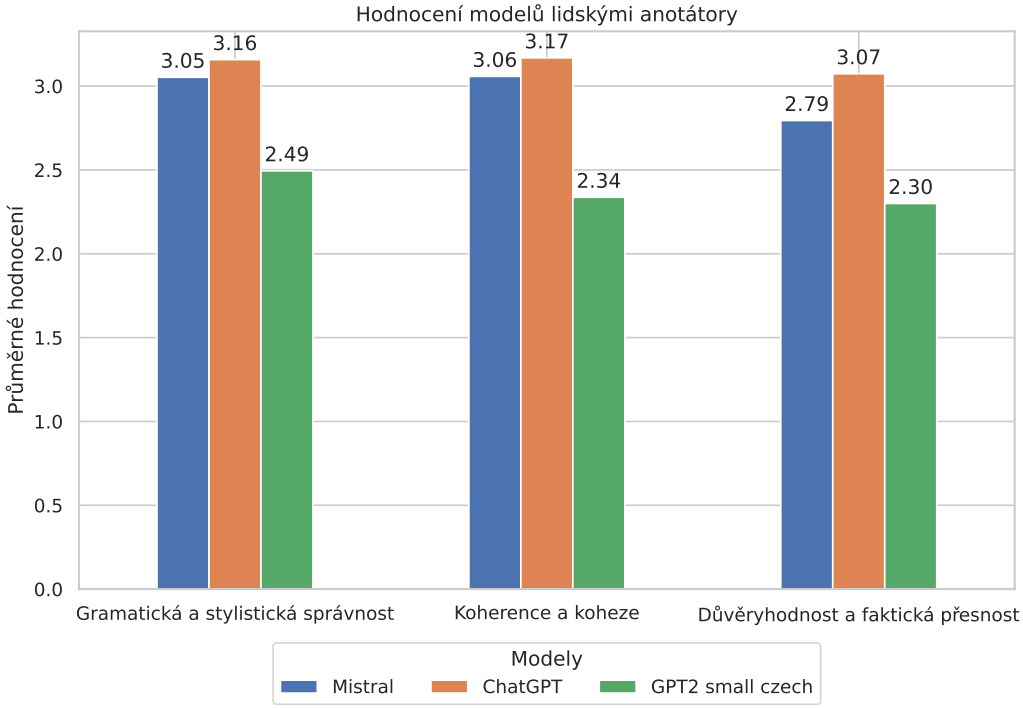
Obrázek 4.6 Porovnání skóre ROUGE-1 pro 3 různé přístupy ke generování popisků



Obrázek 4.7 Porovnání skóre ROUGE-L pro 3 různé přístupy ke generování popisků



Obrázek 4.8 Porovnání skóre METEOR pro 3 různé přístupy ke generování popisků



Obrázek 4.9 Výsledky hodnocení lidskými anotátory

Závěr

V této práci byly prozkoumány možnosti generování popisků produktů na základě dat o produktech v českém jazyce. K tomuto účelu byla použita data z reálného e-shopu. V průběhu práce byly porovnány tři přístupy generování dat. Prvním z nich bylo fine-tunování malého modelu GPT-2-small-cs na datech z e-shopu. Další způsoby používaly velké jazykové modely Mistral a ChatGPT, kde pro Mistral byla data přeložena do angličtiny a vygenerované výsledky zpět. ChatGPT byl použit přímo s českými daty. Pro fine-tuning byly použity různé hyperparametry a způsoby předzpracování dat, ale ve výsledku mezi nimi nebyly velké rozdíly.

Byla také vytvořena webová aplikace, která slouží k usnadnění organizace, zdrojových dat, dat připravených pro jazykové modely, spouštění výpočetních úloh na výpočetním clusteru pomocí SLURM a hodnocení vygenerovaných výsledků lidskými anotátory.

K hodnocení výsledků byla použita kombinace automatických metrik a hodnocení lidskými anotátory. Přesto, že výsledky ukázaly jasné pořadí použitých přístupů (ChatGPT, Mistral, GPT-2-small-cs), tak výsledné texty nejsou pro reálné použití příliš vhodné.

Seznam použité literatury

[1] Albert Gatt a Emiel Krahmer. „Survey of the state of the art in natural language generation: Core tasks, applications and evaluation“. In: *Journal of Artificial Intelligence Research* 61 (2018), s. 65–170.

[2] Ashish Vaswani et al. „Attention is all you need“. In: *Advances in neural information processing systems* 30 (2017).

[3] Michael A Nielsen. *Neural networks and deep learning*. Sv. 25. Determination press San Francisco, CA, USA, 2015.

[4] Yann LeCun, Yoshua Bengio a Geoffrey Hinton. „Deep learning“. In: *nature* 521.7553 (2015), s. 436–444.

[5] Alec Radford et al. „Improving language understanding by generative pre-training“. In: (2018).

[6] Jacob Devlin et al. „BERT: Pre-training of deep bidirectional transformers for language understanding“. In: *arXiv preprint arXiv:1810.04805* (2018).

[7] Warren S McCulloch a Walter Pitts. „A logical calculus of the ideas immanent in nervous activity“. In: *The bulletin of mathematical biophysics* 5 (1943), s. 115–133.

[8] Frank Rosenblatt. „The perceptron: A probabilistic model for information storage and organization in the brain.“ In: *Psychological review* 65.6 (1958), s. 386.

[9] David E Rumelhart, Geoffrey E Hinton a Ronald J Williams. „Learning representations by back-propagating errors“. In: *nature* 323.6088 (1986), s. 533–536.

[10] Yann LeCun et al. „Gradient-based learning applied to document recognition“. In: *Proceedings of the IEEE* 86.11 (1998), s. 2278–2324.

[11] Ian Goodfellow, Yoshua Bengio a Aaron Courville. *Deep learning*. MIT press, 2016.

[12] Sepp Hochreiter a Jürgen Schmidhuber. „Long short-term memory“. In: *Neural computation* 9.8 (1997), s. 1735–1780.

[13] Kyunghyun Cho et al. „Learning phrase representations using RNN encoder-decoder for statistical machine translation“. In: *arXiv preprint arXiv:1406.1078* (2014).

[14] Tomas Mikolov et al. „Efficient estimation of word representations in vector space“. In: *arXiv preprint arXiv:1301.3781* (2013).

[15] Jeffrey Pennington, Richard Socher a Christopher D Manning. „Glove: Global vectors for word representation“. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, s. 1532–1543.

[16] Armand Joulin et al. „Bag of tricks for efficient text classification“. In: *arXiv preprint arXiv:1607.01759* (2016).

[17] Albert Q Jiang et al. „Mistral 7B“. In: *arXiv preprint arXiv:2310.06825* (2023).

[18] Jiri Spitalsky. *GPT-2 Small Czech (cs)*. `https://huggingface.co/spital/gpt2-small-czech-cs`. Accessed: 2024-01-08. 2022.

[19] Adam Hájek a Aleš Horák. „CzeGPT-2 – New Model for Czech Summarization Task“. 2023.

[20] Thomas Wolf et al. „Transformers: State-of-the-Art Natural Language Processing“. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, říj. 2020, s. 38–45. URL: `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

[21] Martin Popel et al. „Transforming machine translation: a deep learning system reaches news translation quality comparable to human professionals“. In: *Nature communications* 11.1 (2020), s. 4381.

Příloha A

Webová aplikace

Tato příloha obsahuje zdrojové kódy webové aplikace. Její součástí jsou i skripty, pomocí kterých byly fine-tunovány modely, spouštěna inference a hodnocení kvality vygenerovaných textů. Tyto skripty se nachází v adresáři `jobs`. Dají se spustit i samostatně, ale jelikož ukládají data do databáze, jsou primárně určeny k spouštění pomocí webové aplikace, která zároveň zajišťuje předání validních parametrů. Struktura aplikace je podrobně popsána v sekci 3.2. Příloha obsahuje soubor `README.md`, ve kterém je popsáno, jak aplikaci zprovoznit.

Po zprovoznění podle návodu je možné spouštět vše, co aplikace obsahuje. Výjimkou je stahování dat z e-shopu, ke kterému je potřeba přístupový klíč ke GraphQL API nějakého e-shopu postaveného na platformě WPJShop[1], který v rámci práce nemohu sdílet veřejně. Pro vyzkoušení tvorby datasetu a fine-tuningu modelu by bylo potřeba vytvořit JSON soubor, který strukturou odpovídá GraphQL dotazu `dataCreation/loaders/wpj_products_query.graphql`. Po vložení takového souboru do adresáře `data/rawData` aplikace soubor zpracuje a tím umožní použití všech funkcí.

Další úprava je nutná pro vyzkoušení inference pomocí ChatGPT. K tomu je potřeba, aby v prostředí, ve kterém je spuštěna webová aplikace, byla proměnná `OPENAI_API_KEY` s platným API klíčem pro ChatGPT[2].

[1] `https://www.wpj.cz/`
[2] `https://openai.com/product`

Příloha B

Ukázková data

Tato příloha obsahuje ukázková data a databázi pro webovou aplikaci. Tato data se nachází na adrese `https://owncloud.cesnet.cz/index.php/s/XJZypEFSux4zaTO`. Archiv na této adrese obsahuje dva adresáře, které je nutné vložit do stejného adresáře jako webovou aplikaci.

instance Obsahuje ukázkovou databázi.

data Obsahuje nejlepší fine-tunovaný model, který byl použit k finálnímu hodnocení lidskými anotátory, vygenerované výsledky všech 4 fine-tunovaných modelů, vygenerované výsledky velkých jazykových modelů Mistral a ChatGPT, výsledky automatické evaluace a hodnocení lidských anotátorů.

Všechna tato data je možné zobrazit pomocí webové aplikace. Výjimkou jsou výsledky automatické evaluace. Tyto výsledky se nacházejí v adresáři `data/evaluation/<hodnocení-konkrétních-výsledků>`. Ve webové aplikaci na stránce „Evaluation“ je možné najít, jaký adresář odpovídá výsledkům konkrétního modelu.

Hodnocení lidských anotátorů se nachází v adresáři `data/experiment`. Webová aplikace obsahuje pouze prostředí, ve kterém byly výsledky hodnoceny.