FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Diplomová práce

Automatické generování interpunkce v systémech rozpoznávání řeči

Studijní program:	N0613A140028 Informační technologie
Studijní obor:	Inteligentní systémy
Autor práce:	**Bc. Martin Poláček**
Vedoucí práce:	doc. Ing. Petr Červa, Ph.D. Ústav informačních technologií a elektroniky

Liberec 2023

FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Zadání diplomové práce

Automatické generování interpunkce v systémech rozpoznávání řeči

Jméno a příjmení:	**Bc. Martin Poláček**
Osobní číslo:	M21000168
Studijní program:	N0613A140028 Informační technologie
Specializace:	Inteligentní systémy
Zadávající katedra:	Ústav informačních technologií a elektroniky
Akademický rok:	2022/2023

Zásady pro vypracování:

1. Seznamte se s problematikou automatického generování interpunkce v systémech rozpoznávání řeči a proveďte rešerši existujících metod.
2. Na základě provedené rešerše zvolte state-of-the art metodu, která umožní automaticky doplňovat interpunkci v on-line systému pro přepis českého jazyka. Zaměřte se zejména na metody, které využívají neuronové sítě a pracují pouze s textovým výstupem z rozpoznávacího systému.
3. Zvolenou metodu optimalizujte na vhodné vývojové datové sadě vzhledem k nejdůležitějším výkonovým parametrům a hyperparametrům zvolené architektury.
4. Výsledný model s nejlepším nastavením parametrů ověřte na vhodné testovací množině a porovnejte dosažené výsledky s výsledky dostupnými v literatuře a případně také s podobnými existujícími systémy.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	40-50
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] Cristopher Bishop, Pattern Recognition and Machine Learning. Springer, 2006.

[2] Vasile Păiş and Dan Tufiş, Capitalization and punctuation restoration: a survey. Artif. Intell. Rev. 55, 3 (Mar 2022), 1681–1722. https://doi.org/10.1007/s10462-021-10051-x

Vedoucí práce:	doc. Ing. Petr Červa, Ph.D. Ústav informačních technologií a elektroniky
Datum zadání práce:	24. října 2022
Předpokládaný termín odevzdání:	22. května 2023

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 24. října 2022

Prohlášení

Prohlašuji, že svou diplomovou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Jsem si vědom toho, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má diplomová práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

21. května 2023 Bc. Martin Poláček

Automatické generování interpunkce v systémech rozpoznávání řeči

Abstrakt

Tato diplomová práce se zabývá úlohou automatického generování interpunkce (automatic punctuation restoration - APR) v systémech pro automatický přepis řeči, které zpracovávají v reálném čase streamovaná data, například titulkují televizní vysílání. Konkrétně bylo cílem práce navrhnout APR modul, který bude do výstupu rozpoznávacího systému doplňovat tečky, čárky a otazníky. Zároveň bude dostatečně rychlý pro režim online zpracování a bude pracovat s co nejmenším zpožděním a to bez využití prosodických příznaků počítaných z řečového signálu. Výsledný navržený APR modul využívá předtrénovaný jazykový model ELECTRA-Small, který je založený na architektuře typu transformer. Experimentální část práce obsahuje porovnání výsledků dosažených použitím několika dalších architektur a vyšetřuje vliv různých hyperparametrů na proces trénování. V poslední části práce je navržený APR modul porovnán s jiným, již existujícím modulem, který používá kombinaci textových a prosodických příznaků. Z výsledků porovnání vyplývá, že APR modul navržený v této diplomové práci zmíněný modul překonává, splňuje všechny požadavky zadání a dosahuje velmi dobrých výsledků, které jsou plně použitelné v praxi. Novost a vlastní přínos této diplomové práce podtrhuje skutečnost, že navržená metoda a dosažené výsledky byly přijaty k publikaci na prestižní mezinárodní konferenci Interspeech 2023.

Klíčová slova: automatické generování interpunkce, automatické rozpoznávání řeči, ELECTRA model, transformery

Abstract

This thesis deals with the task of automatic punctuation restoration (APR) in automatic speech recognition systems that process real-time streaming data, such as subtitling television broadcasts. Specifically, the goal of this work was to design an APR module that will add periods, commas and question marks to the output of the recognition system. At the same time, it will be fast enough for the online processing mode and work with the least possible delay, without using prosodic features computed from the speech signal. The resulting proposed APR module uses the pre-trained ELECTRA-Small language model, which is based on a transformer-type architecture. The experimental part of the thesis compares the results obtained using several other architectures and investigates the effect of different hyperparameters of the training process. In the last part of the work, the proposed APR module is compared with another existing module that uses a combination of textual and prosodic features. The comparison results show that the APR module proposed in this thesis outperforms the mentioned module, fulfills all the requirements of the assignment and achieves very good results that are fully applicable in practice. The novelty and original contribution of this thesis are underlined by the fact that the proposed method and yielded results were accepted for publication at the prestigious international conference Interspeech 2023.

Keywords: automatic punctuation restoration, automatic speech recognition, ELECTRA model, transformers

Poděkování

Rád bych poděkoval panu docentu Petru Červovi za vedení mé diplomové práce, poskytnutí konzultací při řešení jednotlivých kroků návrhu neuronové sítě a poskytnutí dat pro trénování.

Obsah

Seznam tabulek

Seznam obrázků

Seznam zkratek

TUL	Technická univerzita v Liberci
FM	Fakulta mechatroniky, informatiky a mezioborových studií Technické univerzity v Liberci
ASR	Automatické rozpoznávání řeči
NLP	Zpracování přirozeného jazyka
RNN	Rekurentní neuronové sítě
LSTM	Long-Short Term Memory
GRU	Gated Reccurent Unit
BERT	Bidirectional Encoder Representations from Transformers
ELECTRA	Efficiently Learning an Encoder that Classifies Token Replacements Accurately
SELU	Scaled Exponential Linear Unit
MLM	Masked Language Model
NSP	Next Sentence Prediction
RTD	Replaced Token Detection
TPU	Tensor Processing Units
CRF	Podmíněná náhodná pole
SEQ2SEQ	Sekvence na sekvenci
HMM	Skryté Markovovy modely

1 Úvod

V dnešním rychle se rozvíjejícím světě informačních technologií se stále více zaměřujeme na zpracování textu, což je klíčový prvek mnoha aplikací, jako jsou vyhledávače, analýzy sociálních médií, strojový překlad a další. Jednou z hlavních součástí takového zpracování je správné a efektivní generování interpunkce.

Interpunkce je klíčovým prvkem psaného jazyka, který zajišťuje jeho správné pochopení. Především rozděluje text na jednotlivé části a tím usnadňuje čtení a zpracování informací čtenářem. Navíc pomáhá vyjadřovat emoce a tón, které jsou v mluveném projevu signalizovány změnami hlasu, intonací a pauzami. Interpunkce zároveň zlepšuje srozumitelnost a zřetelnost sdělení tím, že objasňuje vztahy mezi slovy a myšlenkami. Bez správně použité interpunkce dochází k nejasnostem, nedorozuměním a zmatkům.

Automatické generování interpunkce může být prováděno v offline nebo online režimu. Offline režim se zaměřuje na zpracování již existujícího textu a interpunkce se doplňuje v průběhu následné editace a korektury. Online režim zpracovává text v reálném čase, například během konverzace nebo zasedání, což vyžaduje rychlejší a efektivnější metody zpracování.

V souvislosti s automatickým generováním interpunkce existuje několik způsobů, jak lze tuto úlohu realizovat. Jednou z možností je analýza textu, která může zahrnovat analýzu morfologie, slovosledu nebo gramatických funkcí slov. Jiný přístup zahrnuje využití prosodických příznaků, jako jsou intonace, délka slabik nebo pauzy mezi slovy, což může pomoci při rozpoznávání hranic vět a správného umístění interpunkce.

V obou případech může být automatické generování interpunkce založeno na tradičních statistických metodách, jako jsou skryté Markovovy modely (HMM) nebo gramatiky založené na pravidlech. Dnes jsou však nejvíce rozšířené metody strojového učení, konkrétně neuronové sítě.

Tato diplomová práce nejdříve představuje související práce v oblasti generování interpunkce, současné architektury jazykových modelů a metody pro jejich trénování a doladění. Poté popisuje navržený přístup ke generování interpunkce, včetně přípravy trénovacích dat, tokenizace, trénování modelu a jeho doladění. Součástí práce je také podrobný popis způsobů zpracování streamovaných dat a vyvážení tříd, které mohou ovlivnit výslednou výkonnost navrženého přístupu.

V experimentální části práce je provedeno porovnání několika různých architektur pomocí testovacích dat, přičemž jsou použity různé metriky pro měření výkonnosti, jako jsou precision, recall a F1 skóre. Toto porovnání umožňuje identifikovat nejvýkonnější architekturu pro danou úlohu. Dále práce zkoumá vliv různých hyper-

parametrů na výkonnost modelu a určuje jejich optimální kombinaci pro dosažení co nejlepších výsledků.

V posledním experimentu je porovnán navržený přístup s již existujícím systémem pro generování interpunkce na datech, které jsou vygenerované systémem pro rozpoznávání řeči (ASR). Tyto data jsou navíc zatížena i chybami v rozpoznaných slovech, která záměrně nejsou opravena, aby bylo možné porovnat výsledky na reálných datech.

2 Související práce

Automatické generování interpunkce (APR) představuje důležitou úlohu v oblasti zpracování přirozeného jazyka, která se často řeší jako sekvenční značkování s cílem obnovy několika z nejpoužívanějších interpunkčních znamének, jako jsou tečky (”.”), čárky (”,”) a otazníky (”?”). Výzkumníci se původně zaměřili na vývoj různých statistických metod pro řešení této úlohy, které zahrnovaly modely maximální entropie [4, 5] a podmíněná náhodná pole (CRF) [6].

Modely maximální entropie jsou pravděpodobnostní modely, které využívají princip maximální entropie k odhadu pravděpodobnostního rozložení mezi různými možnými interpunkčními znaménky. Tyto modely se staly oblíbenými v oblasti zpracování přirozeného jazyka, zejména pro úlohy jako je APR, díky své schopnosti zachytit nelineární vztahy a interakce mezi jednotlivými slovy vstupního textu.

Podmíněná náhodná pole představují alternativní statistický přístup, který modeluje pravděpodobnostní rozložení přiřazení interpunkčních značek na základě vlastností pozorovaných dat. Zároveň umožňují zohlednit závislosti mezi sousedními značkami a vstupními daty.

V průběhu času dochází k výraznému rozvoji v oblasti neuronových sítí, který se projevil velkým počtem výzkumných prací a aplikací, které používají neuronové sítě. První z použitých neuronových sítí jsou konvoluční neuronové sítě (CNN), které přinesly zásadní pokrok v oblasti analýzy textu a zpracování textu. V [7] byly CNN spolu s předtrénovanými slovními vektory úspěšně použity pro predikci interpunkčních znamének v nesegmentovaných přepisech[1]. Tento přístup otevřel cestu k dalším pokročilým technikám založeným na neuronových sítích.

Rekurentní neuronové sítě (RNN) jsou dalším významným typem neuronových sítí, který umožňuje modelovat sekvenci o téměř libovolné délce na sekvenci o jiné délce (seq2seq). RNN byly úspěšně použity v různých odborných pracích, jako například v [8] a [4], kde autoři pro generování interpunkce využívali textové příznaky a informaci o pauzách ze zvukové nahrávky.

V některých odborných článcích byly použity také obousměrné RNN (Bi-RNN) s attention mechanismem. Tyto modely umožňují procházet sekvence v obou směrech a přinášejí tak lepší schopnost zpracovat vstupní text a pochopit kontext jednotlivých slov. Příkladem takové práce je [9].

V další odborné práci [10] bylo zapojeno více RNN vrstev sekvenčně. Attention mechanismus byl následně aplikován na výstup z každé vrstvy. Díky tomu, se podařilo zachytit více vztahů a informací ze vstupních dat, což zvýšilo kvalitu

[1] Nesegmentované přepisy jsou bloky textu, které neobsahují žádné formátování a rozdělení.

vygenerované interpunkce.

Dalším pokrokem v oblasti neuronových sítí bylo vytvoření modelů založených na architektuře transformerů. Transformery oproti předchozím RNN nezpracovávají data sekvenčně, ale paralelně. Zároveň implementují self-attention mechanismus, který dokáže nalézt souvislosti mezi jednotlivými slovy ve vstupní sekvenci. Transformery byly úspěšně použity například v práci [11]. Významný posun nastal při použití již předtrénovaných trasformerů, jako je BERT [12]. Předtrénované transformery se staly základem pro řadu dalších prací, které využily jeho schopnosti ke zvýšení úspěšnosti při generování interpunkčních znamének. Příkladem takových prací jsou [13] a [14].

V rámci dalšího zlepšování výsledků byl využit transformer ELECTRA [15], který byl vyvinut jako inovativní vylepšení nad již existujícím transformerem BERT. Transformer ELECTRA představuje nový přístup v oblasti jazykových modelů, kdy se skládá ze dvou hlavních komponent - generátoru a diskriminátoru a oproti transformeru BERT se liší hlavně efektivnějším způsobem trénování.

V práci [16] byl proces trénování modelu ELECTRA rozšířen o diskriminátor řešící více úkolů. Tento přístup se zaměřuje na zvýšení robustnosti modelu, což vede ke zlepšení výsledků detekce nejednoznačností. To znamená, že model je schopen lépe identifikovat a řešit nesouvislosti a chyby ve vstupních textech.

2.1 Online režim zpracování

Tato práce se zaměřuje na nejnáročnější podúlohu automatického doplňování interpunkce, kterou je zpracování dat v reálném čase s nízkou odezvou. Konkrétním příkladem cílové aplikace je například titulkování živého televizního vysílání (TV/R) v českém jazyce. Počet odborných prací zaměřující se na APR úlohu s ohledem na nízkou latenci je přitom velice omezený.

Jedním z přístupů, který je popsán v odborné literatuře, je použití rekurentních neuronových sítí (RNN) s latencí třech slov. Tento přístup byl představen v odborném článku [1], kde autoři kombinovali prosodické a textové příznaky na vstupu modelu. Prosodické příznaky, jako jsou ticha mezi slovy, frekvence vstupních slov, výskyt nádechů a další, jsou generovány z audio nahrávky daného přepisu. Tento proces výrazně komplikuje trénování modelu, protože zpracování zvukových nahrávek vyžaduje dodatečné výpočetní zdroje a složitější předzpracování dat.

Druhým přístupem, který se zabývá problematikou latence v APR, je kontrolovaně časově zpožděný transformer (CT-transformer), představený v [17]. Tento model se snaží generovat jak interpunkční znaménka do textu, tak také rozpoznat nesouvisející části textu a ty odstranit. CT-transformer pracuje s latencí 10 slov.

2.2 Cíle práce

Na základě výše uvedené rešerše souvisejících prací a zadání této práce, byly stanoveny následující konkrétní cíle:

1. Navrhnout metodu pro doplňování interpunkce do streamovaných výstupů z ASR systému v češtině.
2. Optimalizovat navrženou metodu vzhledem ke klíčovým hyperparametrům použité architektury neuronových sítí.
3. Porovnat dosažené výsledky s dalšími existujícími architekturami nebo systémy.
4. Vyhodnotit navrženou metodu i na reálných datech z výstupu skutečného rozpoznávacího systému.

3 Současné architektury jazykových modelů

Současné architektury jazykových modelů zaujímají zásadní roli v řadě aplikací zpracování přirozeného jazyka. Tyto modely se staly nezbytným nástrojem v oblastech umělé inteligence a počítačové lingvistiky, přičemž jejich důležitost a využití neustále narůstá. Klíčovým aspektem těchto modelů je jejich schopnost efektivně zpracovávat a generovat texty v přirozeném jazyce, což umožňuje širokou škálu praktických využití - od strojového překladu a automatického shrnutí textů až po generování textů a analýzu sentimentu.

Moderní jazykové modely využívají pokročilé techniky a architektury neuronových sítí, které se učí na základě rozsáhlého množství textových dat. Tyto modely také vytvářejí vektorovou reprezentaci jednotlivých slov, což umožňuje zachytit komplexní vztahy mezi jednotlivými slovy, jejich významem a kontextem, ve kterém jsou použity.

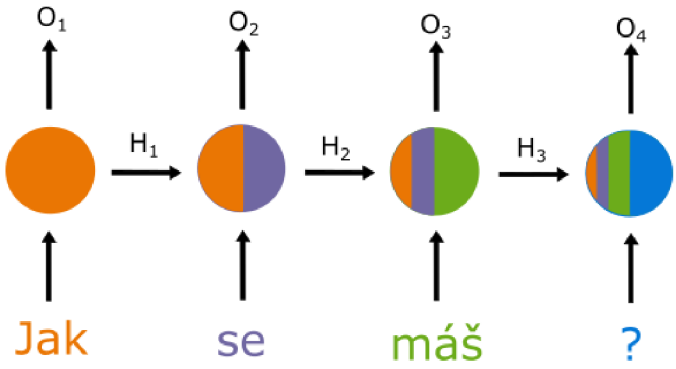
3.1 Rekurentní neuronové sítě

Rekurentní neuronové sítě (RNN) představují specifický druh neuronových sítí, který byl navržen pro efektivní zpracování sekvenčních dat a časových řad. Tyto sítě mají schopnost uchovávat informace z předchozích stavů a využívat je při zpracování následujících stavů.

Základním prvkem těchto sítí je umělý neuron (RNN neuronová buňka), který na vstupu bere v úvahu nejen aktuální data, ale i informace o předchozích datech v sekvenci. Vstup do každé RNN buňky je tedy tvořen aktuálním vzorkem ze vstupní sekvence a předchozí hodnotou skrytého stavu, na které se aplikuje nelineární aktivační funkce (např. tanh). Díky tomu mohou RNN efektivně zpracovávat sekvenční vstup.

Pro zpracování přirozeného jazyka je třeba vložit na začátek neuronové sítě embedding vrstvu, která pro každé slovo vytváří jeho vektorovou reprezentaci. Pro tyto účely můžeme použít již existující vektory (např. GloVe) nebo můžeme vektorovou reprezentaci zahrnout do procesu trénování. Často se slova rozkládají na menší části, což snižuje počet vektorových reprezentací, které je potřeba uchovávat v paměti. V takovém případě se pracuje místo celých slov s tokeny.

Na obrázku 3.1 postupně vstupují jednotlivá slova vstupní sekvence do sítě. U prvního slova do sítě vstupuje pouze samotný embedding vektor slova, protože neexistuje žádný předchozí skrytý stav sítě. Následně je pro každé slovo generován nový skrytý stav a výstup. Výstup může být pro každé slovo použit například pro



Obrázek 3.1: Průchod jednou RNN buňkou

klasifikaci, nebo může být ignorován, pokud nás zajímá pouze výstupní stav po průchodu celé sekvence.

U RNN sítí existuje několik problémů. Prvním z nich je tzv. mizející gradient (anglicky vanishing gradient). Tento jev nastává pro velmi dlouhé vstupní sekvence. Během zpětné propagace (backpropagation) se gradient stává s přibývajícími daty postupně extrémně malým, což významně ztěžuje proces učení. Aktivační funkce tanh také může zhoršit tento problém, jelikož její derivace má na výstupu vždy hodnoty v intervalu od 0 do 1. Druhým problémem je omezená krátkodobá paměť. Jak je zřejmé z obrázku 3.1, se zvyšujícím se počtem vstupů síť postupně ztrácí schopnost uchovávat informace z předchozích stavů, až si nakonec vůbec nemusí "pamatovat", jaká data byla na počátku sekvence.

Řešením obou problémů je využití pokročilejších RNN sítí, které efektivně pracují s ukládáním informací z předchozích stavů. Mezi tyto sítě patří sítě s jednotkami typu GRU (Gated Recurrent Unit) a LSTM (Long-Short Term Memory), které implementují sofistikovanější mechanismy pro ukládání a využití informací z předchozích stavů, což zlepšuje jejich schopnost zpracovávat dlouhé sekvence a řeší problém mizejícího gradientu [18].

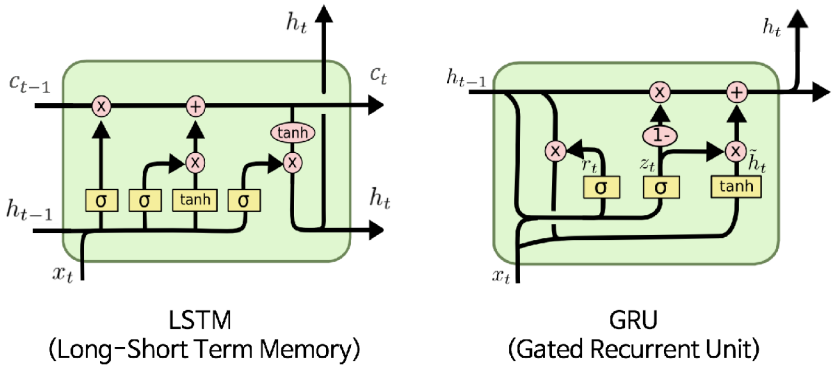
3.1.1 GRU a LSTM

Pro řešení problému mizejícího gradientu byla jako první v roce 1997 představena LSTM jednotka (Long Short-Term Memory). LSTM se skládá z paměťové buňky, která uchovává informace v průběhu času, a tří bran (vstupní, zapomínací a výstupní), jež regulují přenos dat mezi buňkami. Vstupní brána rozhoduje o tom, jaké informace budou přijaty do paměťové buňky, zapomínací brána určuje, jaké informace budou z paměťové buňky odstraněny, a výstupní brána řídí, jaké informace budou poskytnuty následujícím vrstvám sítě.

GRU byla představena v roce 2014 jako zjednodušená varianta LSTM. Tato varianta obsahuje pouze dvě brány (resetovací a aktualizační) a nevyužívá paměťovou buňku. Aktualizační brána je kombinací vstupní a zapomínací brány z LSTM a rozhoduje o přijímání a odstraňování informací. Resetovací brána určuje, jaké informace z předchozího stavu buňky budou použity pro výpočet nového stavu. Snížením počtu bran se GRU stává méně náročnou na výpočetní zdroje, což vede k rychlejšímu

učení a menším paměťovým nárokům.

V praxi se oba typy sítí používají v závislosti na konkrétních řešených problémech. Pokud datová sada obsahuje sekvence s dlouhými časovými závislostmi, může být účinnější použít LSTM, které umožňuje sítím efektivněji uchovávat informace o minulých stavech. Pokud je datová sada jednodušší a neobsahuje dlouhé časové závislosti, představuje rychlejší a efektivnější řešení naopak GRU [19].



Obrázek 3.2: Vnitřní struktura LSTM a GRU buněk [2]

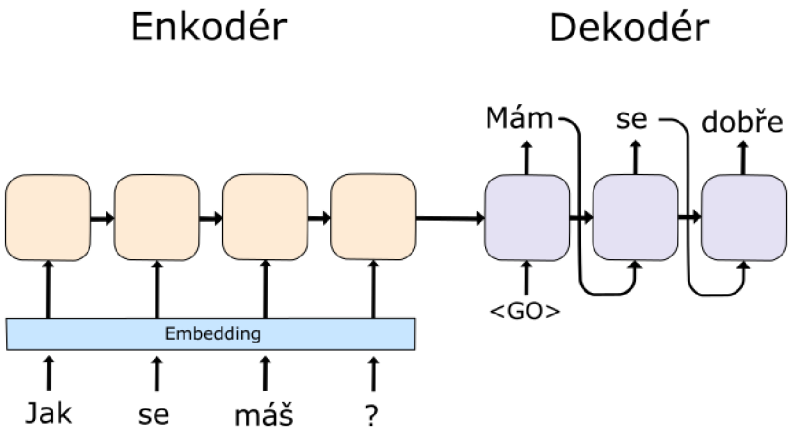
3.2 Enkodér - Dekodér

Významným zlepšením rekurentních neuronových sítí je architektura založená na principu enkodér-dekodér [20]. Tento přístup umožňuje modelům efektivně řešit úkoly, kdy vstupní a výstupní sekvence mají různou délku.

Enkodér, založený na RNN, má za úkol zpracovat vstupní data a vygenerovat z nich příznakový vektor. Postupně prochází všechna vstupní data a snaží se je shrnout do podoby jediného vektoru, který je označován jako kontextový vektor. Současně všechny výstupy z jednotlivých RNN vrstev nejsou dále využívány. Kontextový vektor tak reprezentuje všechna vstupní data (podobně jako skrytý stav u LSTM) a je předán dekodéru.

Dekodér, rovněž založený na RNN, přijímá kontextový vektor na jehož základě generuje výstupní sekvenci. Na začátku je modelu předán na vstupu embedding vektor startovacího symbolu a kontextový vektor. Následně model sekvenčně generuje jednotlivá slova (tokeny) na základě kontextového vektoru a předchozího skrytého stavu vygenerovaného slova. Generování končí v okamžiku, kdy model vygeneruje symbol pro zastavení.

I když enkodér-dekodér architektura přinesla významný pokrok ve zpracování přirozeného jazyka, stále existují určitá omezení. Jedním z nich je limitovaná dlouhodobá paměť. Vzhledem k tomu, že paměť reprezentuje jediný vektor z enkodéru, může docházet ke ztrátě informací, zejména pokud se jedná o delší vstupní sekvenci.



Obrázek 3.3: Architektura enkodér-dekodér

3.2.1 Attention mechanismus

Attention mechanismus (česky mechanismus pozornosti) [3, 20] představuje klíčové vylepšení enkodér-dekodér architektury tím, že řeší omezení kapacity informací uložených v kontextovém vektoru. Hlavním cílem tohoto mechanismu je poskytnout nejrelevantnější informace v kontextovém vektoru v každém kroku generování, což zlepšuje účinnost modelu při zpracování dlouhých vstupních sekvencí.

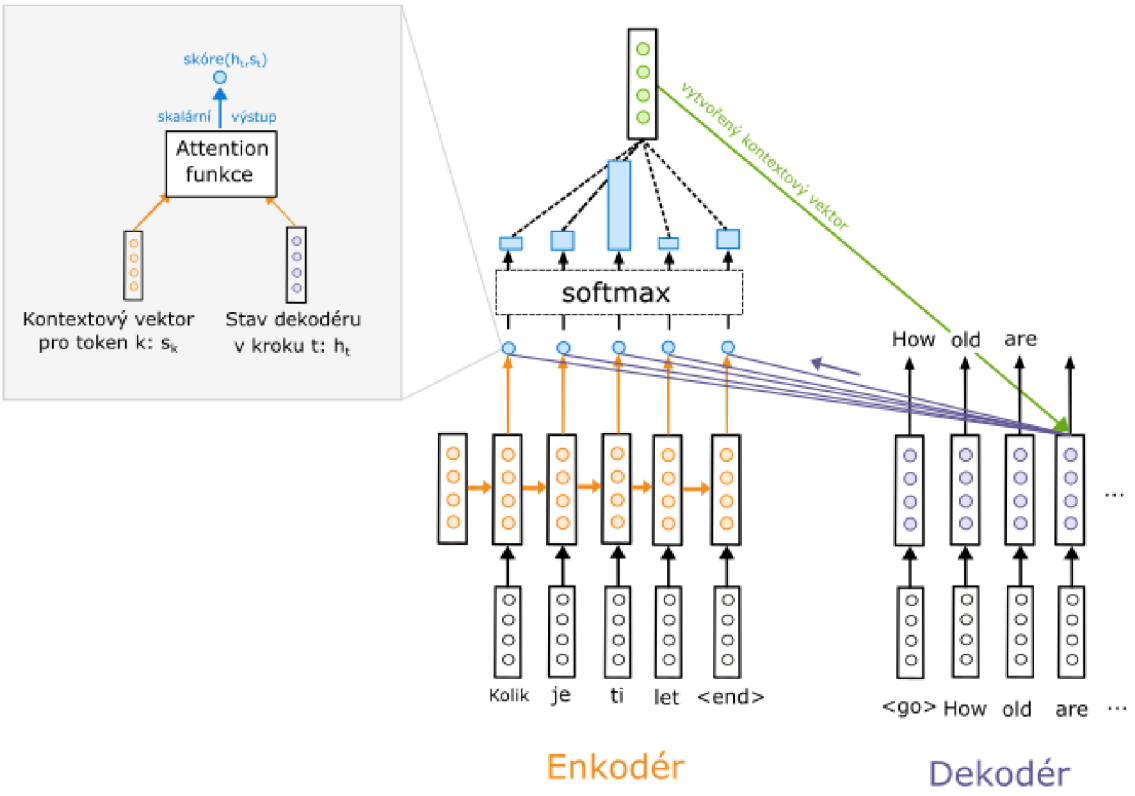
Obrázek 3.4 ukazuje, jak mechanismus pozornosti upravuje fungování enkodéru. Na rozdíl od tradičního přístupu, kde enkodér generuje pouze jeden kontextový vektor, attention mechanismus umožňuje ukládat kontextový vektor pro každý token ve vstupní sekvenci. Tyto kontextové vektory reprezentují částečné informace ze vstupních dat a slouží jako základ pro následné použití v dekodéru.

Během generování nového výstupního tokenu dekodérem je vypočítáno pravděpodobnostní skóre pro každý kontextový vektor vzhledem k aktuálnímu stavu dekodéru v daném kroku. Pravděpodobnostní skóre odráží míru relevance kontextového vektoru pro aktuální pozici výstupní sekvence. Skóre se často počítá pomocí skalárního součinu mezi kontextovým vektorem a stavem dekodéru, ale mohou se použít i jiné metody, jako například násobení váhovou maticí nebo průchod jednoduchou neuronovou sítí.

Pravděpodobnostní skóre vypočtené pro všechny kontextové vektory je normalizováno pomocí softmax[1] funkce, což zajišťuje, že součet všech pravděpodobnostních skóre je roven 1. Pravděpodobnostní skóre lze současně použít také jako váhy pro lineární kombinaci kontextových vektorů.

Nový kontextový vektor je vytvořen váženým součtem původních kontextových vektorů, kde váhy odpovídají pravděpodobnostnímu skóre. Tento nový kontextový vektor reprezentuje relevantní informace pro daný vstupní vektor a je použit při následujícím kroku generování výstupu dekodérem. Attention mechanismus tak umožňuje modelu se flexibilně zaměřit na různé části vstupních dat během generování výstupu.

[1] Softmax je matematická funkce, která transformuje vektor čísel na pravděpodobnostní rozdělení.



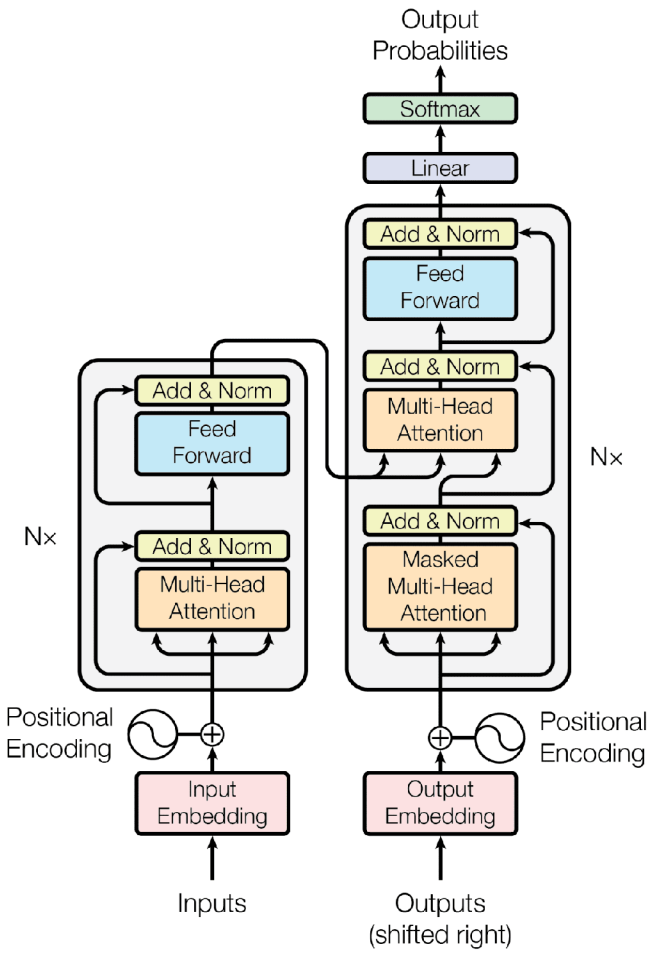
Obrázek 3.4: Attention mechanismus v architektuře seq2seq

3.3 Transformery

Transformery jsou typem neuronových sítí, které byl představen v roce 2017 ve výzkumném článku ”Attention is All You Need” [3]. Transformery představují revoluční změnu v oblasti zpracování sekvencí, protože se odchylují od tradičních RNN a enkodér-dekodér architektur. Hlavním přínosem transformerů je jejich schopnost efektivně a paralelně zpracovávat velké množství dat, díky čemuž dosahují vynikajících výsledků.

Transformery se liší od RNN tím, že nemají žádnou rekurentní strukturu, a od enkodér-dekodér architektur s attention mechanismem tím, že využívají self-attention mechanismus, který umožňuje modelu se soustředit na různé části vstupní sekvence během zpracování vstupních dat.

Jak je ukázáno na obrázku 3.5, transformer se skládá ze dvou hlavních částí: enkodéru a dekodéru. Enkodér se dále dělí na několik identických vrstev, které obsahují self-attention mechanismus a feed-forward vrstvy. Dekodér má také několik vrstev, které zahrnují masked self-attention mechanismus, attention mechanismus spojující enkodér a dekodér a feed-forward vrstvy. Tyto vrstvy spolupracují při zpracování sekvencí tak, že enkodér převádí vstupní sekvenci na vnitřní vektorovou reprezentaci, která je poté dekódována na výstupní sekvenci.



Obrázek 3.5: Schéma popisující jednotlivé vrstvy a vztahy v transformeru [3]

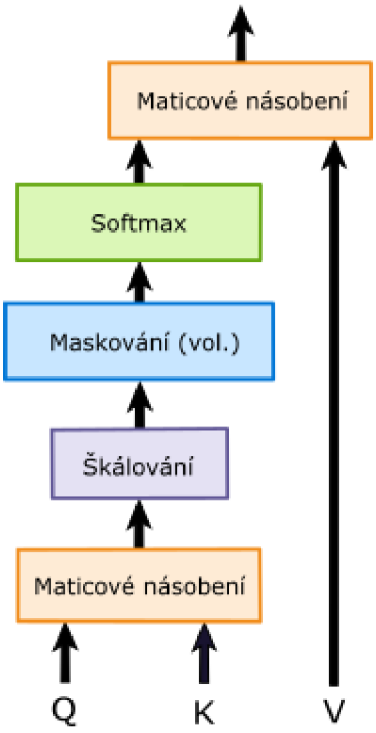
3.3.1 Self-attention mechanismus

Self-attention [3, 21] mechanismus funguje tak, že každý prvek v sekvenci je vážen s každým jiným prvkem na základě jejich vzájemného vztahu. To znamená, že model se soustředí na různé části vstupní sekvence během zpracování, což mu umožňuje zachytit souvislosti mezi jednotlivými tokeny (slovy).

Liší se od klasického attention mechanismu, kdy klasický attention mechanismus propojuje prvky různých sekvencí, jako je například propojení mezi vstupní a výstupní sekvencí, zatímco self-attention se soustředí na vztahy mezi prvky v rámci jedné vstupní sekvence.

Pro výpočet self-attention se každý prvek sekvence transformuje do tří vektorů: vektoru dotazů (query), vektoru klíčů (key) a vektoru hodnot (value). Tyto vektory se vytvářejí pomocí lineárních projekcí vstupních dat.

Následně se pro každý token v sekvenci vypočítá attention skóre pomocí maticového násobení mezi query vektory a key vektory. Attention skóre určuje, jak moc se se má model soustředit na daný token v sekvenci ve vztahu k ostatním prvkům.



Obrázek 3.6: Self-attention mechanismus

Skóre se normalizuje pomocí softmax funkce, což zajišťuje, že součet attention skóre všech tokenů se rovná jedné.

Poté se vektory hodnot každého prvku vynásobí normalizovaným attention skórem. Výsledné vážené vektory hodnot se sečtou, čímž vznikne výstupní vektor pro každý prvek v sekvenci. Tento výstupní vektor zahrnuje informace z celé sekvence a je závislý na vztazích mezi prvky.

Self-attention mechanismus je často prováděn v paralelním zpracování skrze více attention hlav (multi-head attention). Tento přístup umožňuje modelu se soustředit na více informací v sekvenci zároveň. Každá attention hlava je inicializována náhodně a pracuje nezávisle na ostatních, což umožňuje zachytit různé typy vztahů mezi prvky.

Rozšířením je self-attention mechanismus s maskováním. Tento mechanismus zabraňuje dekodéru v nahlédnutí do budoucích pozic v sekvenci během trénování, čímž respektuje časovou posloupnost vstupních dat. Maskovaný self-attention mechanismus vytváří masku, která nastavuje attention skóre pro budoucí prvky na nulu, a tím je vylučuje ze všech následujících výpočtů. Tímto způsobem dekodér může zohlednit pouze aktuální a předchozí prvky ve vstupní sekvenci, což zaručuje, že model je schopen generovat výstupní sekvenci krok za krokem, aniž by se spoléhal na informace z budoucnosti.

Transformery využívají self-attention mechanismus v rámci enkodérů a dekodérů. V enkodéru se používá self-attention pro analýzu vstupní sekvence, zatímco v dekodéru se používá self-attention i klasický attention mechanismus, který propojuje prvky vstupní sekvence s výstupní sekvencí.

3.3.2 Positional encoding

Narozdíl od tradičních rekurentních neuronových sítí, transformery nemají rekurentní strukturu, která zpracovává sekvence slov postupně. To znamená, že transformery musí použít jiný způsob, jak zachovat informace o pořadí slov v sekvenci.

Pro řešení problému zachování pořadí slov v sekvenci je navržen positional encoding [3]. Positional encoding přičítá k vektorové reprezentaci každého slova v sekvenci speciální vektor, který kóduje absolutní nebo relativní pozici slova v sekvenci. Tento přístup umožňuje transformerům rozlišit pořadí slov a jejich vzájemné vztahy.

V tomto postupu se pro každou vektorovou reprezentaci tokenu vypočítá hodnota sinové nebo kosinové funkce s určitými frekvencemi. Tímto způsobem jsou vytvořeny unikátní vektory pro každou pozici slova v sekvenci.

Alternativní způsoby pro výpočet positional encoding vektorů zahrnují například použití trénovatelných parametrů, kde se model učí z dat optimální positional encoding vektory během trénování. Tyto vektory mohou být také získány pomocí jiných matematických funkcí nebo předem určených pravidel, která splňují požadavky na rozlišení pozic slov v sekvenci.

3.3.3 Feed-forward vrstvy a layer normalization

Feed-forward vrstvy [3] jsou zodpovědné za zpracování dat, které jsou získány prostřednictvím attention mechanismu. Tyto vrstvy umožňují modelu zaměřit se na různé části vstupní sekvence, což vede k lepšímu pochopení a reprezentaci kontextu. Strukturálně se feed-forward vrstvy skládají z lineárních vrstev a aktivačních funkcí.

Layer normalization (LN) [22, 23], neboli normalizace vrstvy, je klíčovou součástí optimalizace výpočtů v rámci feed-forward vrstev v transformerech. Tato technika má za úkol urychlit trénování modelu a zlepšit jeho konvergenci.

Základním principem layer normalization je, že normalizace probíhá nezávisle na velikosti jednotlivých dávek. Tento přístup přináší výhodu při trénování na sekvencích s proměnlivou délkou, což je častý jev právě při zpracování přirozeného jazyka. Díky tomu je možné efektivně trénovat model i s menšími dávkami dat, což snižuje nároky na výpočetní zdroje.

Na rozdíl od batch normalization, která normalizuje data ve všech sekvencích v dávce současně, layer normalization zpracovává každou vrstvu zvlášť. Navíc tento způsob normalizace umožňuje zachovat informace o časové závislosti mezi jednotlivými prvky v sekvenci, což je klíčový požadavek.

Proces normalizace vrstvy spočívá ve výpočtu průměru a směrodatné odchylky pro všechny výstupy jedné vrstvy (pro jednu sekvenci). Následně se data upraví tak, aby průměr všech výstupů byl co nejblíže nule a směrodatná odchylka měla hodnotu jedna. Tímto způsobem se snižuje rozsah výstupních hodnot, což zlepšuje stabilitu trénování a urychluje konvergenci modelu. Layer normalization je v transformerech používána jak před vstupem do feed-forward vrstev, tak i na jejich výstupu.

3.3.4 Klasifikační hlava

U natrénovaného transformeru je každému vstupnímu tokenu přiřazena vektorová reprezentace založená na kontextu dané vstupní sekvence. Aby bylo možné jednotlivé vektorové reprezentace efektivně využít pro další úkoly, zejména pro úkoly spojené s klasifikací, musí být k transformeru připojena tzv. klasifikační hlava [24]. Tato klasifikační hlava je zodpovědná za převod výstupních vektorových reprezentací na pravděpodobnostní skóre pro různé třídy, například pro určení interpunkčního znaménka.

Velikost vektorových reprezentací je závislá na konfiguraci a velikosti použitého transformeru [15]. Pro menší modely, označované jako "Small", jsou generovány vektory o velikosti 256 příznaků. Střední modely, označované jako "base", vytvářejí vektory o velikosti 768 příznaků, zatímco velké modely, označované jako "large", generují vektory o velikosti 1536 příznaků.

Klasifikační hlava je obvykle navržena pomocí jedné nebo více lineárních vrstev.. V průběhu trénování se váhy těchto lineárních vrstev upravují tak, aby model dosahoval co nejlepších výsledků na trénovacích datech.

Výběr konkrétní velikosti transformeru a klasifikační hlavy závisí na potřebách daného úkolu a dostupných výpočetních zdrojích. Menší modely mohou být vhodné pro rychlejší výpočty a menší datasety, zatímco větší modely mohou poskytnout lepší výsledky za cenu vyššího výpočetního času a nároků na paměť.

3.3.5 BERT

BERT (Bidirectional Encoder Representations from Transformers) [12] je pokročilý model pro NLP, který byl v roce 2018 představen výzkumným týmem Google AI.

Existují dvě hlavní varianty modelu BERT: BERT-Base a BERT-Large. BERT-Base obsahuje 12 transformer vrstev, které tvoří přibližně 110 milionů parametrů. BERT-Large obsahuje 24 transformer vrstev, které obsahují přibližně 340 milionů parametrů. Více transformer vrstev poskytuje modelu schopnost zpracovávat složitější sekvence dat, zachytit vztahy mezi vzdálenými slovy a lépe se zaměřit na specifické detaily vstupní sekvence.

Jedním z klíčových aspektů modelu BERT je jeho obousměrný přístup. Na rozdíl od tradičních NLP modelů, které zpracovávají text buď zleva doprava nebo zprava doleva, BERT zpracovává text v obou směrech současně. Tato vlastnost umožňuje modelu lépe rozumět kontextu a získat více informací ze vstupní sekvence.

Trénování

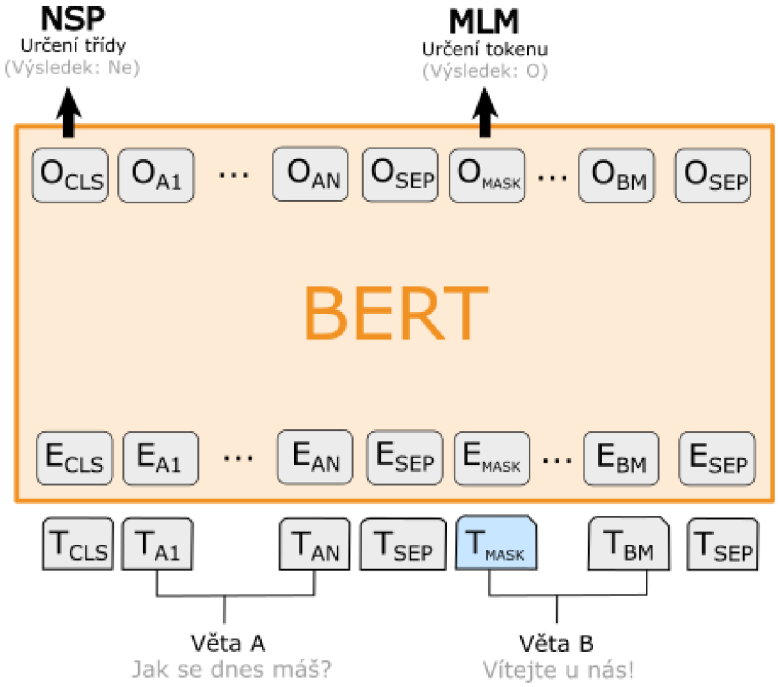
Při trénování modelu se používají dva základní techniky [25]: Masked Language Model (MLM) a Next Sentence Prediction (NSP). BERT je trénován na rozsáhlých textových korpusech, které zahrnují různé formy psaného jazyka, jako jsou knihy a odborné články.

Masked Language Model spočívá v náhodném zakrytí určitého procenta tokenů (ve vstupním textu) maskovacím tokenem "MASK". Cílem modelu BERT je pak

předpovědět původní slova, která byla zakryta, na základě kontextu, ve kterém se nacházejí.

Next Sentence Prediction spočívá v predikci následnosti vět. Modelu jsou předloženy páry vět a jeho úkolem je určit, zda druhá věta v páru skutečně následovala po první větě v původním textu. Tento úkol pomáhá modelu pochopit, jak jsou věty vzájemně spojeny a jak je jejich uspořádání důležité pro pochopení celkového významu textu.

Konkrétně se NSP provádí tak, že se do modelu BERT vloží dvojice vět, přičemž jedna věta je označena jako "věta A" a druhá věta jako "věta B". Model poté vypočítá vektorové reprezentace obou vět a tyto reprezentace jsou následně použity k predikci následnosti vět. K tomuto účelu se používají speciální tokeny "CLS" a "SEP". Token "CLS" se přidá na začátek věty A, zatímco token "SEP" se přidá na konec věty A a na konec věty B. Takto upravená dvojice vět je poté vložena do modelu BERT jako vstup [12].



Obrázek 3.7: Trénování transformeru BERT

Využití v NLP

Díky svému obousměrnému přístupu a výborným výsledkům se BERT stal jedním z nejpoužívanějších modelů v oboru zpracování přirozeného jazyka. BERT je vhodný pro širokou škálu NLP úloh, jako jsou klasifikace textu, analýza sentimentu, extrakce informací, strojové překlady, otázky a odpovědi a mnoho dalších [26].

BERT je také často používán jako základ pro další výzkum a vývoj v oblasti NLP. Vědci a vývojáři vytvářejí nové modely založené na již natrénovaném modelu BERT nebo jeho architektuře, kde se snaží zlepšit jeho výkon, zmenšit velikost modelu, zlepšit rychlost a efektivitu nebo přizpůsobit model pro specifické úkoly či jazyky.

Výhody a omezení

BERT přinesl řadu výhod pro NLP, ale také má svá omezení [26]. Mezi hlavní výhody patří:

1. Flexibilita: BERT může být doladěn a upraven pro širokou škálu úloh NLP, což z něj činí univerzální řešení pro mnoho jazykových problémů.
2. Obousměrný přístup: Díky tomuto přístupu je BERT schopen lépe zachytit kontext a význam slov, což vede k lepším výsledkům v různých jazykových úlohách.
3. Předtrénování: BERT je předtrénován na velkých korpusech textu, což mu umožňuje získat bohatou jazykovou znalost a snížit množství potřebných anotovaných dat pro doladění na konkrétní úlohu.

Mezi hlavní omezení patří:

1. Rychlost: BERT může být pomalejší než některé jiné NLP modely, zejména při trénování a inferenci, což může být problematické pro aplikace, které vyžadují rychlé zpracování.
2. Velikost modelu: BERT, zejména BERT-large, má velmi vysoké výpočetní a paměťové nároky, což ztěžuje jeho nasazení na zařízeních s omezenou výpočetní kapacitou, jako jsou mobilní telefony nebo vestavěné systémy.

3.3.6 ELECTRA

ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately) [15] je stejně jako BERT založený na principu transformerů. Byl představen v roce 2020 týmem z Google AI. Hlavní myšlenkou modelu ELECTRA je rozpoznávání a klasifikace nahrazených tokenů ve vstupních textových sekvencích. Model se snaží rozlišit, zda byl konkrétní token ve vstupní sekvenci původní nebo nahrazený jiným slovem. Tento přístup se nazývá RTD (Replace Token Detection) a stojí v kontrastu s principem MLM, který využívá BERT model. ELECTRA má několik velikostních variant, které se liší podle velikosti modelu a počtu parametrů.

ELECTRA-Small je nejjednodušší a nejmenší verze modelu ELECTRA, která je optimalizována pro rychlé trénování a nasazení na zařízeních s omezenými výpočetnými prostředky. Tato varianta obsahuje 12 transformer vrstev a zhruba 14 milionů parametrů. Díky menšímu počtu vrstev a parametrů je ELECTRA-Small vhodný pro méně náročné úlohy a rychlejší nasazení v praxi.

ELECTRA-Base je střední velikostí modelu, který nabízí lepší výkon než ELECTRA-Small díky většímu počtu vrstev a parametrů. Tato varianta obsahuje 12 transformer vrstev a zhruba 110 milionů parametrů. ELECTRA-Base je vhodný pro náročnější NLP úlohy, kde je potřeba vyšší úroveň jazykového porozumění a detailnější reprezentace vstupních textových dat.

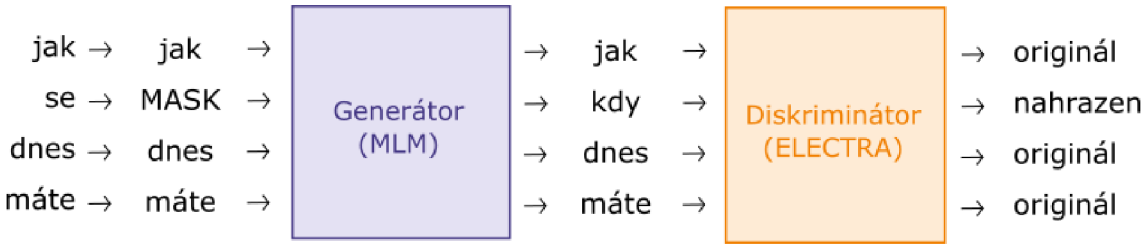
ELECTRA-Large je největší a nejsložitější verze modelu ELECTRA, která poskytuje nejlepší výsledky z dostupných variant. Tento model obsahuje 24 transformer vrstev a zhruba 335 milionů parametrů, díky čemuž je vhodný pro velmi náročné NLP úlohy, kde je potřeba detailní porozumění a generování komplexních výstupů. Kvůli své velikosti a náročnosti na výpočetní prostředky je ELECTRA-Large často trénován na specializovaném hardwaru, jako jsou GPU s velkou kapacitou paměti nebo TPU (Tensor Processing Units).

Trénování

ELECTRA se skládá ze dvou hlavních částí, které v průběhu trénování spolupracují. První z nich je generátor a druhá diskriminátor.

Začátek trénování spočívá ve vytvoření vstupní sekvence. Pro každou vstupní sekvenci je náhodný počet tokenů (např. 15 %) nahrazen tokenem "MASK". Tyto maskované tokeny slouží jako tokeny, které má za úkol generátor předpovědět. Generátor je navržen tak, aby byl co nejmenší a nejrychlejší, což minimalizuje potřebný výpočetní výkon spojený s trénováním a dopředným průchodem modelem.

Tokeny předpovězené generátorem jsou následně hodnoceny diskriminátorem. Diskriminátor je trénován na rozpoznávání správných a nesprávných tokenů, které generátor vytváří. Jeho cílem je rozlišit mezi původními tokeny ve vstupní sekvenci a tokeny nahrazenými generátorem. To znamená, že diskriminátor provádí binární klasifikaci, a na rozdíl od modelu BERT, algoritmus zpětné propagace se aplikuje na všechny vstupní tokeny [15].



Obrázek 3.8: Trénování transformeru ELECTRA

Trénování ELECTRA modelu je inspirován GAN (Generative Adversarial Network) metodou, kde generátor a diskriminátor soutěží a zlepšují se navzájem. Tento přístup umožňuje modelu ELECTRA dosahovat srovnatelné nebo lepší výsledky než modely BERT, avšak s menšími požadavky na výpočetní zdroje. Důvodem je, že se oba modely (generátor a diskriminátor) učí současně, což efektivněji využívá trénovací data.

Jakmile je trénování dokončeno, generátor se již nepoužívá a zůstává samotný diskriminátor. Diskriminátor je následně používán pro dotrénování na řešení specializovaných NLP úloh.

Porovnání s transformerem BERT

Zatímco oba modely mají mnoho společného, existují i rozdíly mezi nimi. Jelikož ale ELECTRA model vznikl jako vylepšení BERT transformeru, tak transformer ELECTRA téměř ve všem převažuje nad transformerem BERT [27].

1. Efektivní trénování: ELECTRA využívá techniku GAN pro trénování, která je efektivnější než kombinace MLM a NSP. Díky tomu je trénování ELECTRA modelů rychlejší a vyžaduje méně výpočetních zdrojů.

2. Lepší výkon: ELECTRA dosahuje srovnatelných nebo lepších výsledků než BERT na různých úlohách zpracování přirozeného jazyka, jako je klasifikace textu, zodpovídání otázek a další.

3. Menší modely: ELECTRA může dosáhnout lepších výsledků s menším počet parametrů než BERT, což vede k menší velikosti modelu a snazšímu nasazení na zařízeních s omezenými výpočetními prostředky.

Nicméně, s tím, že ELECTRA je poměrně nový transformer, přichází také určité problémy:

1. Složitější architektura: Implementace ELECTRA je složitější než BERT, protože zahrnuje současné trénování generátoru i diskriminátoru. To může ztížit odhalení chyb v průběhu trénování.

2. Méně dostupné: BERT je obecně více rozšířený v komunitě zabývající se zpracováním přirozeného jazyka, což znamená, že je snazší najít předtrénované modely. U ELECTRA transformeru se může stát, že pro určitý jazyk nemusí existovat již předtrénovaná verze nebo požadovaná velikost modelu. V takovém případě je často nutné si natrénovat vlastní verzi od počátku, což může být časově náročné, vyžadovat značné výpočetní zdroje a velké množství trénovacích dat.

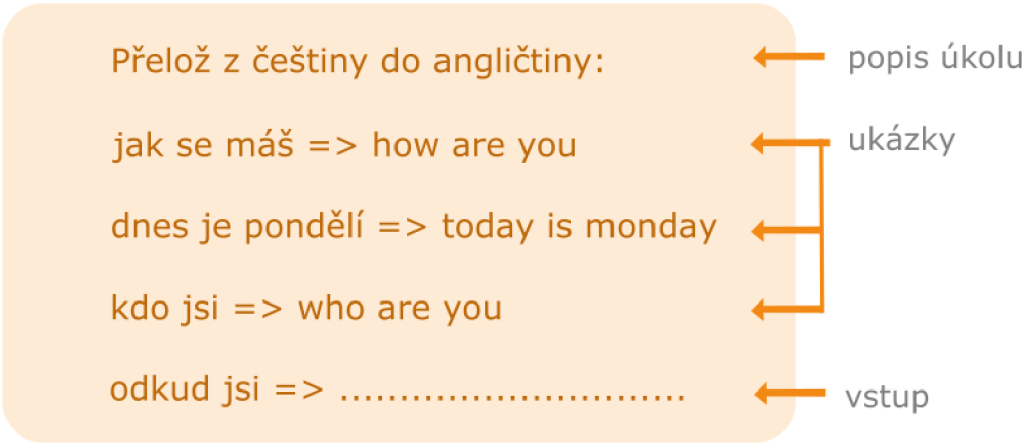
3.3.7 GPT-3

Základní informace

Generative Pre-trained Transformer 3 (GPT-3) [28] je jazykový model vyvinutý společností OpenAI. Byl představen v červnu 2020 a je založen na architektuře transformerů. GPT-3 je třetí generací modelů GPT a vyznačuje se obrovským počtem parametrů. S 175 miliardami parametrů je mnohem větší než jeho předchůdce GPT-2, který má 1.5 miliardy parametrů.

GPT-3 je schopen generovat text, který je často k nerozeznání od textu napsaného člověkem. Zároveň je schopen zvládat širokou škálu úkolů, jako je strojový překlad, doplňování textu, sumarizace, analýza sentimentu a mnoho dalších. GPT-3 také dokáže rozumět a generovat text ve více než 20 jazycích.

Jedním z hlavních přínosů GPT-3 je jeho schopnost few-shot learning [29], což znamená, že se dokáže rychle adaptovat na nové úkoly s minimálním množstvím trénovacích dat. V praxi tak již předtrénovanému modelu na začátku popíšeme úkol a ukážeme vzorové příklady. Model následně pro následující vstup (na základě předchotí zkušenosti ze vzorových ukázek) vygeneruje odpověď, která odpovídá úkolu, který byl ukázán ve vzorových příkladech. To umožňuje vývojářům vytvářet aplikace s GPT-3 bez nutnosti provádět rozsáhlé trénování modelu pro každý nový úkol.



Obrázek 3.9: Few-shot learning

Architektura

GPT-3 obsahuje velký počet vrstev a parametrů, což umožňuje modelu učit se extrémně složité jazykové vzory a závislosti. Proces dekódování je založen na autoregresivním principu, což znamená, že model generuje text slovo za slovem, přičemž se v každém kroku podmíněně generuje následující slovo na základě všech předchozích slov [30].

Jedním z hlavních omezení architektury GPT-3 je její paměťová náročnost, která se výrazně zvyšuje s počtem vrstev a parametrů. To může ztěžovat trénování a nasazení modelu na méně výkonných zařízeních. V praxi je nezbytné použít cluster s více grafickými kartami. Tím jsou jednotlivé výpočty distribuovány, což umožňuje časově efektivní provozování modelu.

Trénování

Trénování GPT-3 [28] je založeno na učení s učitelem (supervised learning), kde model dostává páry vstupních a cílových sekvencí tokenů. GPT-3 je trénován na obrovském korpusu textových dat, který zahrnuje webové stránky, knihy, články a další zdroje. Díky velkému množství trénovacích dat se model dokáže naučit téměř jakékoliv jazykové vzory a závislosti mezi slovy.

Pro trénování GPT-3 se používá metoda maximalizace pravděpodobnosti. Model je trénován tak, aby co nejlépe predikoval následující token ve vstupní sekvenci na

základě kontextu předchozích tokenů. V průběhu trénování se upravují váhy a parametry modelu tak, aby se minimalizovala chyba mezi predikovanými a skutečnými tokeny.

Vzhledem k velikosti a složitosti modelu je trénování GPT-3 náročné na výpočetní zdroje a čas. Vyžaduje velké množství GPU nebo TPU a trénování může trvat několik týdnů. Kromě toho je trénování modelu také energeticky velmi náročné.

4 Navržený přístup ke generování interpunkce

Modul pro generování interpunkce (APR) slouží jako jedna z částí automatického rozpoznávání řeči. Pouze na základě textu z rozpoznávače řeči má za úkol rozhodnout pro každé ze slov, zda za slovem má následovat interpunce a pokud ano, tak jaká. Zároveň modul musí být dostatečně robusní, aby dokázal identifikovat chyby v přepisu, které by měl ignorovat a dokázal dostatečně rychle zpracovávat streamovaná data s co nejmenším zpožděním.

4.1 Použitá architektura neuronové sítě

Výsledná architektura použitá v této práci odpovídá architektuře typu transformer, konkrétně architektuře ELECTRA (viz. 3.3.6). Výběr této architektury by proveden na základě experimentálního vyhodnocení různých dalších architektur na testovací sadě (viz. 5.4.7) Toto vyhodnocení prokázalo, že tento model dosahuje nejlepších výsledků, přestože má malý počet parametrů, díky čemuž jsou nízké i jeho výpočetní nároky.

Aby byl model ELECTRA použitelný v APR modulu, musí být (stejně jako všechny transformery) doplněn o klasifikační hlavu, která každý z tokenů přidělí do jedné z interpunkčních tříd. Celkem se rozlišují čtyři třídy. Třída None značí, že za tokenem nenásleduje žádné interpunkční znaménko. Zbylé tři třídy zastupují jedno z nejpoužívanějších interpunkčních znamének (tečka, čárka a otazník).

Klasifikační hlava použitá v této práci se skládá ze dvou lineárních vrstev. První lineární vrstva, s aktivační funkcí SELU, má za úkol rozšířit příznakový vektor (pro každý token) o velikosti 256 na velikost 512. Následuje druhá lineární vrstva, která ze vstupního příznakového vektoru generuje vektor o velikosti 4. Tento vektor následně prochází funkcí softmax, která určuje pravděpodobnostní skóre pro každou interpunkční třídu. Na základě skóre se následně zvolí nejpravděpodobnější třída a doplní se interpunkce do vstupního textu.

Počet klasifikačních vrstev jsem určil na základě experimentálních výsledků, které jsou detailně porovnány v podkapitole 5.6.1.

4.2 Příprava trénovacích dat a tokenizace

Všechna data (představená v 5.1), která jsem použil v rámci trénování i doladění všech modelů, musela být vhodně upravena a normalizována. Tento proces se skládá ze čtyř kroků:

1. Zachování pouze standardních znaků abecedy, číslic, interpunkčních znamének a odstranění všech ostatních znaků.
2. Převod všech velkých písmen na písmena malá.
3. Odstranění nadpisů článků.
4. Rozdělení dat na jednotlivé věty.

Následně je třeba provést tokenizaci dat. V době psaní této práce byla většina modelů pro zpracování přirozeného jazyka dostupná prostřednictvím portálu HuggingFace, na kterém uživatelé sdílí již předtrénované modely. Zároveň existuje i stejnojmenná knihovna, která výrazně ulehčuje práci s modely, jejich daty pro trénování a tokenizéry. V rámci knihovny je také implementován WordPiece tokenizér, který však není možné použít pro již existující univerzitní ASR moduly. Ty naopak využívají SentencePiece tokenizér [31]. Pro volbu vhodného tokenizéru jsem použil následující kritéria:

- Rychlost převodu textu na tokeny a získání ID tokenů.
- Implementační náročnost do již existujících ASR systémů používaných na univerzitě.

Z implementačního hlediska bylo jednodušší zvolit SentencePiece tokenizér, který již byl implementován v ASR systémech na univerzitě. Pro oba typy tokenizerů jsem také změřil časy potřebné k tokenizaci a výsledky ukázaly, že SentencePiece tokenizér byl až pětkrát rychlejší. Na základě těchto měření jsem zvolil SentencePiece tokenizér, pro který bylo třeba nastavit několik parametrů. Velikost slovníku, která určuje maximální počet slov, který může slovník obsahovat. Rozdělení číslovek, které rozděluje čísla složená z více číslic na jednotlivé číslice, aby se zabránilo vytváření slov obsahujících více číselných znaků. Předem definované symboly určují symboly, které slovník musí obsahovat jako samostatný token a informace o velikosti korpusu, kdy pro velké korpusy dochází k lepší správě paměti v průběhu vytváření modelu.

Pro použití tokenizéru v této práci jsem parametry nastavil následovně:

- **Velikost slovníku:** 30522, dle doporučení pro trénování ELECTRA modelu [1].
- **Rozdělení číslovek:** Ano, aby se zabránilo vytváření slov obsahujících více číselných znaků.

- **Předem definované symboly:** Standardní interpunkční znaky, jako jsou tečka, čárka, otazník atd.
- **Trénování na velkém korpusu:** Ano

S nastavenými parametry jsem zahájil proces tvorby unigramového modelu (na kompletním datasetu), který bude následně využíván tokenizérem. Vzhledem k tomu, že knihovna SentencePiece není schopna paralelního zpracování ve více vláknech, došlo ke značnému prodloužení doby potřebné pro vytvoření modelu. Celková doba potřebná pro vytvoření modelu činila šest hodin, přičemž během některých fází tvorby bylo zapotřebí až 450 GB operační paměti RAM.

4.3 Trénování modelu

Pro trénování ELECTRA transformeru je k dispozici připravená knihovna Electra[1], kterou vyvinul výzkumný tým společnosti Google. Tato knihovna slouží k trénování různých variant ELECTRA transformeru s poměrně srozumitelnou implementací.

Proces trénování přesto přináší několik výzev:

- Knihovna využívá zastaralou verzi TensorFlow 1.5.
- Verze 1.5 nepodporuje trénování na nejmodernějších grafických kartách.
- Knihovna nenabízí podporu pro tokenizér SentencePiece.
- Výsledné modely je nutné konvertovat do formátu kompatibilního s frameworkem Pytorch, ve kterém probíhá doladění.

Každou z těchto výzev bylo nezbytné vyřešit.

Mnoho stávajících projektů, které byly vytvořeny v minulosti, využívá framework TensorFlow verze 1.5. S aktualizací TensorFlow na verzi 2.0 byl ukončen vývoj starších verzí a nebylo možné využívat modernější grafické karty. Problém jsem musel vyřešit pomocí speciální verzí TensorFlow od společnost NVIDIA, která podporuje trénování na nejmodernějších grafických kartách.

Instalace probíhá tradičním způsobem za použití správce balíčků Pip následujícím příkazem:

```
$ pip install nvidia-tensorflow[horovod]
```

Další výzvou byl problém s využitím SentencePiece tokenizéru. Knihovna podporovala pouze WordPiece tokenizér, který je také založený na unigramovém modelu. Bohužel, neexistoval jasně definovaný způsob převodu unigramového modelu pro tokenizér SentencePiece na model pro WordPiece tokenizér tak, aby tokenizace probíhala shodně. Z tohoto důvodu jsem musel upravit část knihovny tak, aby knihovna

[1] https://github.com/google-research/electra

přestala používat WordPiece tokenizér a místo toho začala využívat Sentencepiece tokenizér.

Poslední výzvou byla konverze modelu do formátu kompatibilního s frameworkem Pytorch. Řešení přinesl projekt electra_pytorch[2], který byl navržen přímo pro transformer ELECTRA.

Jako vstup přijímá TensorFlow checkpoint model vytvořený v průběhu trénování a soubor s popisem nastavení všech hyperparametrů transformeru tak, jak byly použity během trénování. Výstupem je model kompatibilní s frameworkem Pytorch a současně i knihovnou HuggingFace, která implementuje funkce pro jeho snadné načtení.

4.3.1 Příprava dávek a spuštění trénování

Pro přípravu dávek pro trénování slouží skript, který umožňuje načítání a zpracování korpusu textových dat. Skript přijímá následující parametry:

- **corpus-dir**: cesta k adresáři, obsahujícímu předzpracované textové soubory pro trénování
- **vocab-file**: cesta k souboru se slovníkem, který bude využit tokenizérem
- **output-dir**: cesta k adresáři, kam budou uloženy připravené soubory pro trénování
- **num-processes**: počet vláken, která budou použita pro zpracování dat

Pro trénování modelu jsem použil veškerá dostupná data. Jejich preprocessing (viz. 4.2) trval při využití 10 jader (20 vláken) více než 6 hodin.

Druhý skript v projektu již slouží k samotnému trénování modelu. Na vstupu je nutné pouze určit adresář s předzpracovanými daty, název modelu a hyperparametry modelu. Pro každý model jsou dvě možnosti nastavení hyperparametrů:

1. Nastavení velikosti modelu a výběr z předdefinovaných konfigurací.
2. Individuální úprava každého z přednastavených hyperparametrů.

Během trénování se ukládají tzv. checkpointy po průchodu každých 1000 dávek, které slouží pro obnovení trénování v případě jeho nečekaného přerušení. Při opětovném spuštění trénování (s totožnými parametry) se načtou uložené checkpointy a trénování pokračuje od místa, kde bylo přerušeno. Zároveň checkpointy také slouží k následnému převodu na Pytorch model.

[2] https://github.com/lonePatient/electra_pytorch

4.4 Doladění modelu

Pro správnou funkčnost klasifikační hlavy se musí model doladit. Ladění probíhá na samotném modelu i na klasifikačních vrstvách. Příprava dat pro doladění probíhá následovně.

Vygenerují se čísla v rozmezí 1 až 10 určující počet vět ke spojení do delších textových řetězců.

Následně se řetězce seřadí podle počtu slov a seskupí do jednotlivých dávek tak, aby se počty slov v každé dávce lišily co nejméně. Ve 25% případů se navíc odstraní koncové slovo řetězce, aby se model nenaučil předpoklad, že všechny řetězce jsou zakončené tečkou.

Generování dávky probíhá následovně:

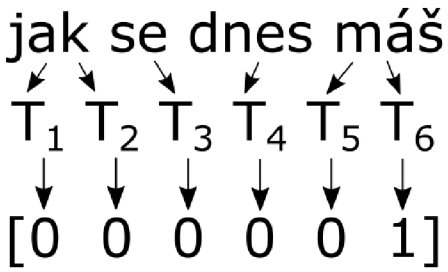
1. Jednotlivé řetězce v dávce se normalizují a tokenizují tak, jak je uvedeno v předchozí kapitole.

2. Pro každý token se vytvoří prázdný vektor interpunkce, který bude obsahovat informaci o tom, zda za daným tokenem následuje interpunkční znaménko a pokud ano, tak jaké.

3. Tokeny v řetězci se postupně prochází. Tokeny značící interpunkci se odstraní a informace o jejich existenci se zaznamená do vektoru interpunkce, který náleží předchozímu tokenu.

4. Všechny řetězce tokenů se doplní nulovými tokeny, aby délka řetězců byla stejná.

5. Pro každý z interpunkčních vektorů se nastaví příznak, který určuje, zda daný vektor odpovídá plnohodnotnému tokenu nebo tokenu nulovému.



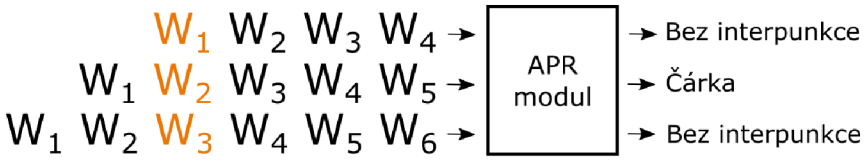
Obrázek 4.1: Grafické znázornění generování dávky

Výsledkem jsou tři vektory. První reprezentuje jednotlivé tokeny, druhý obsahuje informaci o tom, jaké interpunkční znaménko má následovat, a třetí určuje, které tokeny jsou relevantní pro trénování.

4.5 Způsob zpracování streamovaných dat

V režimu offline zpracování je možné vstupní text rozdělit do bloků a následně vygenerovat interpunkci pro celý blok najednou. Rozdělování do bloků může probíhat s překryvem, což zajišťuje dostatečný kontext pro každé slovo v textu.

Avšak blokové zpracování není ideální volbou pro aplikace pracující se streamovanými daty. V případě streamovaných dat se interpunkce doplňuje ke každému slovu zvlášť, a to s co nejmenším možným zpožděním. To znamená, že budoucí kontext je omezen na nejnutnější minimum.



Obrázek 4.2: Zpracování streamovaných dat po jednotlivých slovech

Kontext jednotlivých slov je tak obvykle tvořen velkým počtem předchozích slov, zatímco budoucí kontext zahrnuje pouze několik následujících slov. Maximální délka kontextu je určena maximálním počtem vstupních tokenů, které může zpracovat model ELECTRA.

Detailní srovnání obou režimů, včetně výhod a nevýhod, je představeno v kapitole 5.5.

4.6 Vyvážení tříd

Nevyváženost tříd v datasetu je běžným jevem ve strojovém učení a může vést ke zkresleným výsledkům při trénování a testování modelů. Při zpracování takovýchto nevyvážených datasetů je důležité zaměřit se na správné nastavení vah pro jednotlivé interpunkční třídy, aby bylo dosaženo optimálních výsledků.

V mém datasetu byl počet tokenů v jednotlivých třídách značně nevyvážený, neboť frekvence výskytu třídy "none" (bez interpunkce) byla mnohem vyšší než u teček, čárek a zejména otazníků. Na základě validačních dat jsem podle četností jednotlivých interpunkčních znamének určil váhy tak, jak jsou uvedené v následující tabulce.

třída	váha
bez interpunkce	1.0
otazník	5.0
tečka	2.0
čárka	1.5

Tabulka 4.1: Váhy jednotlivých tříd

5 Experimentální vyhodnocení

V rámci této práce byl proces trénování i doladění časově a výpočetně velice náročný. Z toho důvodu jsem využil možnost využít výpočetní prostředky Metacentra[1] Cesnet.

Pro trénování jsem používal jednu grafickou kartu a vždy jsem alokoval výpočetní prostředky s následujícími parametry:

- Počet jader: 4 (8 vláken)
- Grafická karta: NVIDIA Tesla A100 40 GB
- Pamět RAM: 512 GB

Pro distribuci jazyka Python (verze 3.9) jsem zvolil software Anaconda, který mi navíc umožnil nainstalovat několik knihoven, které byly klíčové pro trénování jednotlivých architektur a jejich vyhodnocování. Nejdůležitější z těchto knihoven byly:

- NVIDIA Tensorflow (trénování modelu)
- Pytorch (doladění modelu)
- HuggingFace (implementace architektur)
- Scikit-Learn (vyhodnocovací metriky)

5.1 Trénovací data

Pro správné natrénování jednotlivých modelů bylo klíčové použít co nejvíce dat (z různých zdrojů), aby se výsledné modely dokázaly přizpůsobit libovolným vstupním datům. Celková velikost dat byla 25 GB a skládaly se z následujících kategorií:

- Novinové a internetové články.
- Manuálně opravené přepisy českých televizních a rádiových vysílání.
- Diplomové práce.

[1] Distribuovaná výpočetní infrastruktura

- Zákony, rozsudky a smlouvy.

Na vytvořeném datasetu jsem provedl analýzu všech interpunkčních znamének. Z této analýzy vyplynulo, že téměř 95% veškeré interpunkce tvořily tečky, čárky a otazníky. Tato tři interpunkční znaménka byla navíc pro čitelnost a pochopení textu klíčová.

Kromě teček, čárek a otazníků byla všechna ostatní interpunkční znaménka z textu odstraněna. Jedinou vyjímkou byly vykřičníky, které jsem nahradil tečkou.

Vzhledem k tomu, že některé modely se nejdříve trénovaly a až následně doladily, bylo nezbytné rozdělit celý dataset na několik částí:

- Trénovací sada (**90%**)
- Trénovací sada pro doladění (**9%**)
- Validační sada pro doladění (**1%**)

Pro testování jsem použil sadu dat, která byla zvlášť vyčleněna a obsahovala přepisy zpráv z rozhlasu. Tato data dobře reprezentují cílovou aplikaci, kterou je doplňování teček do výstupu ASR systému používaného pro monitoring televizního a rozhlasového zpravodajství. Jde ovšem o ručně opravené přepisy. Vyhodnocení na neopravených přepisech zatížených chybou rozpoznávacího systému je provedeno zvlášť v kapitole 6.

Tento testovací dataset měl celkově velikost 25 MB a obsahoval 256608 tokenů. Na třídu None připadalo 225443 tokenů, na třídu otazníky 17103 tokenů, na třídu tečky 17103 tokenů a třídě čárky odpovídalo 15467 tokenů.

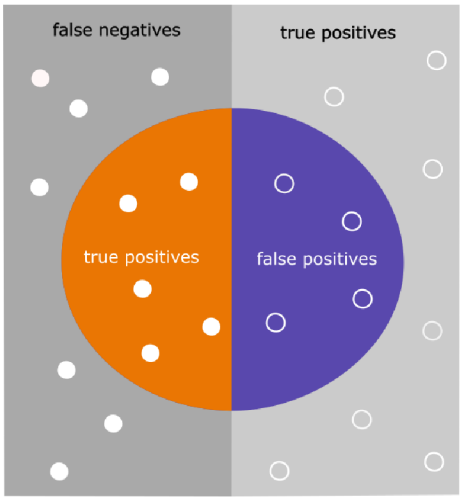
5.2 Metriky pro vyhodnocování

Pro správné vyhodnocení vygenerované interpunkce je nezbytné zvolit metriku, která bude měřit úspěšnost jednotlivých modelů. Při doplnění každého interpunkčního znaménka mohou nastat z hlediska správnosti čtyři situace:

- Interpunkce byla správně určena (true positive - TP)
- Interpunkce nebyla správně určena (true negative - TN)
- Interpunkce nebyla určena na místě, kde měla být (false negative - FN)
- Interpunkce byla určena na místě, kde neměla být (false positive - FP)

Během vyhodnocení se pro každou ze tříd spočítají 4 výše definované hodnoty. Zároveň se také provede vyhodnocení přes všechny třídy současně. To nebere ohled na jednotlivá interpunkční znaménka, neboť ta jsou v některých případech nejednoznačná (například aplikace tečky nebo čárky v rámci dlouhého souvětí je velmi subjektivní).

Pro výpočet jednotlivých metrik přes všechny třídy se postupuje tak, že nejdříve jsou jednotlivé metriky vypočítány pro každou z interpunkčních tříd zvlášť. Následně se vypočítá vážený průměr, který zohledňuje počty tokenů v jednotlivých třídách.



Obrázek 5.1: Grafické znázornění významu hodnot TP, TN, FN a FP

5.2.1 Precision, recall a F1 skóre

První z vyhodnocovacích metrik je přesnost (precision - P). Ten udává poměr správně identifikovaných znamének ke všem identifikovaným znaménkům:

$$P = \frac{TP}{TP + FP} \tag{5.1}$$

Další z vyhodnocovacích metrik je záchyt (recall - R), který udává poměr správně identifikovaných znamének ke všem referenčním znaménkům:

$$R = \frac{TP}{TP + FN} \tag{5.2}$$

V určitých případech je vhodné popisovat úspěšnost modelu pouze jedním číslem. K tomu slouží F1 skóre, které se počítá na základě dvou předešlých metrik. Pro výpočet lze využít rovnici 5.3 nebo v případě již spočítaných metrik (přesnost a záchyt) lze vypočítat F1 skóre jako jejich harmonický průměr.

$$F1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{5.3}$$

Pro efektivní výpočet u rozsáhlých datových sad lze využít funkci "classification_report", která je součástí knihovny scikit-learn. Tato funkce přijímá na vstupu vektor X, jenž slouží jako referenční hodnoty, a vektor Y, který představuje výstupy z APR modulu. Volitelným parametrem je seznam tříd, pro něž mají být metriky vypočítány. Výstupem funkce jsou pro každou třídu vypočtené hodnoty precision, recall a F1 skóre. Současně také poskytuje informaci o zastoupení každé třídy v referenčních datech [32].

5.3 Hyperparametry použité při trénování

Základní nastavení hyperparametrů bylo převzato z experimentů provedených autory [15]. V tabulce 5.1 jsou uvedeny hyperparametry, které se u jednotlivých velikostí architektury liší. Nejvýznamnější změny nastávají u počtu attention hlav, velikosti skryté vrstvy a velikosti embedding vektoru. Model ELECTRA-Base vyžaduje téměř devětkrát více paměti oproti modelu ELECTRA-Small, což si vyžádalo úpravu velikosti dávky a počtu trénovacích kroků, aby nedošlo k překročení paměťových nároků (viz. 5.4.5).

Parametr	ELECTRA-Small	ELECTRA-Base
Počet vrstev	12	12
Velikost skryté vrstvy	256	768
Počet attention hlav	4	12
Velikost embedding vektoru	128	768
Learning rate	$5e^{-4}$	$2e^{-4}$
Velikost dávky	128	96 (256)
Počet trénovacích kroků	1 000 000	2 000 000 (766 000)

Tabulka 5.1: Parametry trénování ELECTRA modelů

Abych dosáhl úspěšného dotrénování modelů, bylo nutné zvolit vhodný optimizér, počet epoch a přizpůsobit rychlost učení.

Jako optimizér jsem použil AdamW, který se osvědčil při dotrénování transformerů [33]. Oproti běžně používanému optimizéru Adam lépe reguluje úpravu vah v průběhu trénování, což je klíčové u modelů s velkým počtem parametrů a trénovaných na velkém množství dat.

Rychlost učení jsem nastavil na hodnotu 10^{-5} pro samotný model a 4×10^{-4} pro přidanou klasifikační hlavu v první epoše. Každých 10 000 dávek jsem rychlost učení snižoval na 95 % předchozí hodnoty. Tento postup umožňuje modelu na začátku se rychle adaptovat na vstupní data a následně s malou vahou měnit jednotlivé parametry, aby bylo dosaźeno nejlepších výsledků.

Počet epoch jsem stanovil na hodnotu 3. Během dotrénování jsem model ukládal každých 20 000 dávek. Po dokončení trénování jsem následně jednotlivé modely porovnával na validační sadě a vybral ten, který dosahoval nejlepších výsledků. Tento postup zároveň zajistil, že v případě nečekaného přerušení bylo možné pokračovat v dotrénování od posledního uloženého modelu.

5.4 Porovnání různých architektur

První experiment porovnává několik architektur neuronových sítí. Porovnává se nejen úspěšnost modelu, ale také jeho velikost a čas zpracování. Pro jeho výpočet je použit procesor Intel i7 9700K v režimu zpracování v jednom vlákně. Výsledný čas

reprezentuje jeden dopředný průchod modelem. Celé vyhodnocení probíhá v režimu blokového zpracování, přičemž jednotlivé bloky se navzájem nepřekrývají.

5.4.1 Rekurentní neuronová síť

První architektura odpovídá rekurentní neuronové síti. Jedná se o kombinace obousměrných LSTM a GRU vrstev zakončených klasifikační hlavou. První embedding vrstva má velikost 30522 přesně dle počtu tokenů, se kterými pracuje tokenizér. Každý vstupní token je reprezentován příznakovým vektorem o velikosti 256. Další vrstvy jsou uspořádány následovně:

Vrstva	Počet vstupních parametrů	Počet výstupních parametrů
Embedding	-	256
LSTM	256	100
BiLSTM	100	40
GRU	80	20
BiLSTM	20	20
GRU	40	20
LSTM	20	10
Lineární	10	4

Tabulka 5.2: Uspořádání vrstev v rekurentní síti

Výsledný model dosáhl na testovací sadě výsledků uvedených v tabulce 5.4.1. Z výsledků vyplývá, že otazníky jsou velmi obtížně rozpoznatelné - model téměř žádné nenalezl. Pokud ano, tak pouze 19 % bylo určeno správně. Lepších výsledků model dosáhl u teček a čárek, kde hlavní vliv mělo několikanásobně větší zastoupení v trénovacích datech. Stále se však od modelu očekává, že precision a recall budou dosahovat podobných hodnot. V tomto případě se podobných hodnot nedosáhlo, kdy interpunkčních znaménka byla nadužívána nebo naopak nadměrné vynechávána.

	P [%]	R [%]	F1 [%]
otazník	19.4	3.4	5.8
tečka	35.4	54.7	43.0
čárka	70.6	13.2	22.2
jedna třída	57.5	46.2	51.2

Tabulka 5.3: Výsledky dosažené pomocí RNN

V případě vyhodnocení nezávisle na třídě interpunkčního znaménka byla dosažena precision přes 57 % a recall přes 46 %. I z toho výsledku vyplývá, že model poměrně úspěšně nalezne slova po kterých následuje interpunkce, avšak často dochází k zaměňování jednotlivých interpunkčních znamének.

5.4.2 BERT (Czert)

Další vyhodnocovanou architekturou jsou transformery. V tomto konkrétním případě se jedná o transformer typu BERT od Googlu (viz. 3.3.5). V rámci této práce nebyl prostor trénovat tuto architekturu od počátku. Místo toho jsem použil již existující předtrénované modely. Konkrétně existují dva modely (trénované pouze na českých datech) pojmenované jako Czert[2]. První z nich pracuje s textem obsahujícím i velká písmena, zatímco druhý dokáže pracovat pouze s textem zapsaným pomocí malých písmen.

Vzhledem k tomu, že výstup cílového ASR systému obsahuje pouze malá písmena, byl výběr modelu jednoznačný a zvolil jsem ten, který pracuje pouze s malými písmeny. Jednalo se o model ve velikosti "Base", který obsahuje celkem 110 milionů parametrů. Pro jeho trénování byl jeho autory použit dataset o celkové velikosti 36 GB složený z článků z Wikipedie a českých zpráv.

K tomuto již existujícímu modelu jsem připojil klasifikační hlavu a provedl jeho doladění. V tomto případě byl vstup do klasifikační hlavy tvořen vektorem o velikosti 768.

	P [%]	R [%]	F1 [%]
otazník	42.2	39.1	40.6
tečka	75.9	68.2	71.8
čárka	76.2	83.7	79.8
jedna třída	87.3	90.5	88.9

Tabulka 5.4: Výsledky dosažené pomocí transformeru Czert

Výsledky v tabulce 5.4 ukazují, že model po doladění vykazuje výrazně lepší výsledky než model založený na architektuře RNN. Zlepšení bylo dosaženo ve všech třídách a metrikách. Při porovnání výsledků F1 metriky u jednotlivých tříd došlo u otazníku ke zlepšení o 34,8 %, u teček o 28,8 % a u čárek o 57,6 %. Při porovnání interpunkce (jako jedné třídy) došlo také k výraznému zlepšení. Přes 90% všech interpunkčních znamének bylo nalezeno a to s přesností 87,3 %. Oproti předchozímu modelu se tak splnil požadavek, že hodnoty u metrik precision a recall by měly být vyrovnané. Horší výsledky model vykazoval u otazníků, kde výsledky stále ovlivňuje jejich menší počet v trénovacích datech.

5.4.3 ELECTRA-Small (Small-E-Czech)

Následující typ transformeru odpovídá novější architektuře ELECTRA. Výběr již existujících modelů je však v tomto případě ještě více omezený než u transformeru BERT. V době psaní této práce existoval pouze jeden model natrénovaný na českých datech - a to model od společnosti Seznam s názvem Small-E-Czech[3]. Slovo "small"

[2]https://huggingface.co/UWB-AIR/Czert-A-base-uncased
[3]https://huggingface.co/Seznam/small-e-czech

v názvu modelu značí, že z několika možných variant (v počtu parametrů) se jedná o tu nejmenší z nich, která obsahuje celkem 13,6 milionů parametrů.

Tokenizér použitý v rámci modelu byl unigramový s maximálním počtem tokenů omezeným na 30522. K samotnému trénování byl použit dataset o velikosti 253 GB, který byl složený z dokumentů nalezených pomocí webového vyhledávače společnosti Seznam. Celý dataset byl navíc převeden na malá písmena, což je pro dosažení dobrých výsledků klíčové.

Parametry a klasifikační hlava pro doladění byly zvoleny stejně jako u modelu BERT. Vstup do klasifikační hlavy byl tvořen vektorem o velikosti 256.

	P [%]	R [%]	F1 [%]
otazník	35.1	36.2	35.6
tečka	67.5	79.7	73.1
čárka	81.5	79.2	80.3
jedna třída	89.5	91.9	90.7

Tabulka 5.5: Výsledky dosažené pomocí transformeru Electra-Small (od počátku)

Výsledky (uvedené v tabulce 5.5) získané po doladění byly velmi podobné těm dosaženým u transformeru BERT. Došlo k mírnému zhoršení u otazníků, ale naopak ke zlepšení u teček a čárek. Zároveň bylo dosaženo i zlepšení v případě vyhodnocení bez ohledu na interpunkční třídu - o téměř 2 %.

Celkově lze konstatovat, že model ELECTRA-Small vykazuje srovnatelné výsledky s modelem BERT, ačkoliv má výrazně menší počet parametrů. To naznačuje, že tento model může být vhodnou alternativou pro úlohy, kde je potřeba šetřit výpočetními zdroji nebo pamětí, aniž by došlo k poklesu ve kvalitě doplňování interpunkce.

5.4.4 ELECTRA-Small (od počátku)

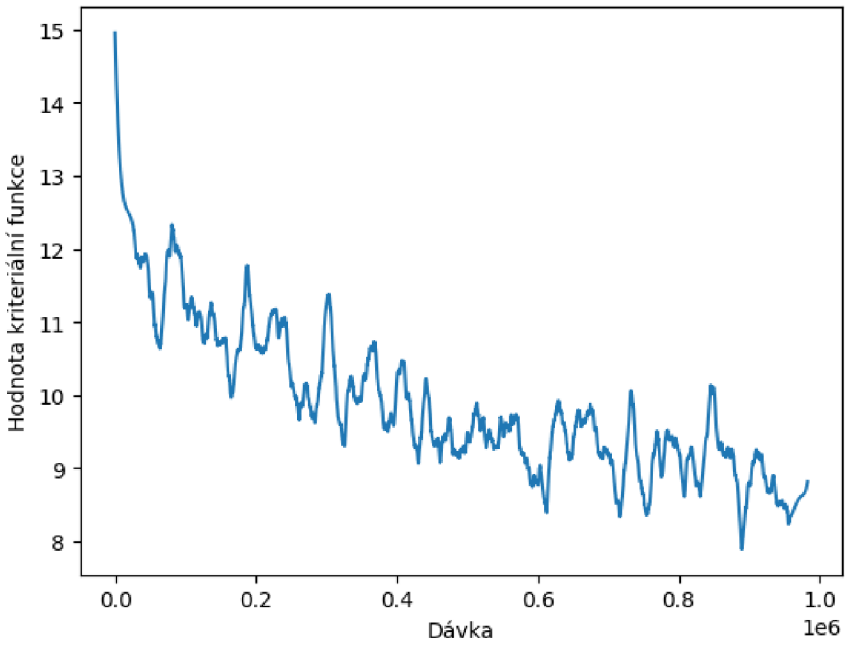
Vzhledem k dobrým výsledkům předtrénovaného modelu ELECTRA jsem rozhodl, že tento model bude tvořit jádro výsledného APR modulu. Do budoucna se přitom předpokládá, že stejný APR modul bude vytvořen i pro další jazyky, pro které ovšem nemusí předtrénovaný ELECTRA model existovat. Zároveň předtrénovaný model využíval jiný tokenizér, než využívá používaný ASR systém univerzity. Z obou těchto důvodů bylo rozhodnuto, že bude provedeno natrénování modelu ELETCRA také vlastními silami a z vlastních dat. To zároveň umožňilo natrénovat i větší variantu ELECTRA modelu upravením několika hyperparametrů.

Pro trénování varianty small jsem alokoval stejné výpočetní prostředky jako u předchozích modelů, jelikož použitá knihovna pro trénování nepodporuje paralelní výpočty na více grafických kartách.

Z hlediska hyperparametrů byla klíčovým faktorem velikost dávky, která byla stanovena na hodnotu 128. Běžné grafické karty by mohly mít problém s paměťovou kapacitou, avšak v rámci této práce jsem použil grafickou kartu NVIDIA A100,

která s 40 GB paměti poskytuje dostatečnou rezervu. Pro natrénování jsem použil 1 milion dávek, a celková doba trénování činila 84 hodin.

Během trénování bylo možné pozorovat oscilaci kriteriální funkce. U běžné neuronové sítě by tento jev naznačoval přetrénování, kdy model ztrácí schopnost generalizace, což ve většině případů způsobuje nedostatečná velikost datasetu. Nicméně, v případě generátoru a diskriminátoru u transformeru ELECTRA, tento jev souvisí s faktem, že jakmile se zlepšuje diskriminátor, zlepšuje se i generátor, což vyvolává oscilaci kriteriální funkce. Bez předchozí zkušenosti nelze určit, zda se diskriminátor učí tak, jak je požadováno. Proto k ověření výsledků modelu lze přistoupit až po jeho doladění.



Obrázek 5.2: Graf oscilace kriteriální funkce v průběhu trénování

Po konverzi modelu na kompatibilní verzi s frameworkem Pytorch jsem provedl doladění stejně jako u předchozího ELECTRA modelu, s výsledky uvedenými v tabulce 5.6.

	P [%]	R [%]	F1 [%]
otazník	46.2	38.6	42.1
tečka	76.9	70.9	73.8
čárka	74.6	84.7	79.3
jedna třída	89.2	92.2	90.7

Tabulka 5.6: Výsledky dosažené pomocí transformeru Electra-Small

Porovnání výsledků v tabulce 5.6 s modelem předtrénovaným společností Seznam ukazuje mírné zlepšení v generování teček a otazníků, avšak naopak zhoršení u čárek.

Z porovnání výsledků napříč všemi interpunkčními třídami je zřejmé, že oba modely vykazují téměř identické výsledky. Podstatným zjištěním je, že navzdory použití o 80 % menšího datasetu (oproti datasetu spol. Seznam) bylo dosaženo téměř shodných výsledků.

5.4.5 ELECTRA-Base (od počátku)

Implementace většího modelu vyvolala komplikace spojené s nedostatečnou kapacitou paměti grafického karty. Hlavním důvodem byla skutečnost, že model typu base obsahuje 109 milionů parametrů, což představuje více než osminásobek počtu parametrů oproti modelu small.

Jediným možným řešením byla redukce velikosti dávky z 256 na 96 sekvencí. V souladu s doporučeními mělo být pro trénování využito 744 tisíc dávek, avšak za účelem dodržení celkového počtu všech trénovacích sekvencí, jsem zvýšil počet dávek na 2 miliony. Výsledná doba trénování činila 288 hodin.

Model ELECTRA-Base generuje pro každý token na výstupu vektor příznaků o velikosti 768 stejně jako Czert. Z toho důvodu byla pro klasifikaci využita identická klasifikační hlava jako pro tento model.

	P [%]	R [%]	F1 [%]
otazník	48.1	33.4	37.1
tečka	75.6	71.2	73.3
čárka	76.0	81.8	78.7
jedna třída	89.5	91.9	90.7

Tabulka 5.7: Výsledky dosažené pomocí transformeru Electra-Base

Z výsledků v tabulce 5.7 vyplývá, že model typu base dokáže detekovat interpunkci bez ohledu na třídy srovnatelně s variantou typu small. Mírné zhoršení nastalo u výsledků přes jednotlivé třídy. To může být způsobeno menší počtem použitých trénovacích dat, než model ve velikosti base požaduje. V době psaní této práce ovšem neexistoval model typu base natrénovaný na opravdu velkém množství českých dat (řádově 100 GB) a proto nebylo možné provést srovnání s touto variantou. Předpokládám ovšem, že na více datech by mohlo být dosaženo stejných nebo lepších výsledků oproti variantě typu small.

5.4.6 Webová služba s GPT-3 modelem

Některé velké a obecné předtrénované modely jsou dnes k dispozici také ve formě webové služby. Příkladem je například GPT-3 model vyvinutý společností OpenAI, který v posledním období zaznamenává značný zájem ze strany veřejnosti.

Pro účely porovnání jsem proto pro úlohu tečkování zkusil využít výše zmíněnou službu dostupnou na adrese https://platform.openai.com/playground. Proces doplňování interpunkce jsem realizoval prostřednictvím API, kam jsem odesílal textové

bloky o maximální velikosti 512 tokenů a následně obdržel text s doplněnou interpunkcí. Model GPT-3 je v rámci tohoto API možné využít v tzv. editačním režimu, který zabraňuje častému doplňování nebo odstraňování slov ze vstupní sekvence. Zadání bylo vždy formulováno následovně: "Doplň do textu tečky, čárky a otazníky. Vykřičníky nahraď tečkami."

	P [%]	R [%]	F1 [%]
otazník	37.2	45.3	40.9
tečka	77.3	63.5	69.7
čárka	85.7	45.1	59.1
jedna třída	90.3	62.2	73.7

Tabulka 5.8: Výsledky dosažené pomocí modelu GPT-3 v editačním režimu

Výsledky uvedené v tabulce 5.8 ukazují, že model GPT-3, navzdory své robustnosti, dosahuje výrazně horších výsledků než dosud vyhodnocené specializované modely. Přestože metrika precision vykazuje mírně lepší výsledky, recall je v porovnání s ostatními transformery horší o téměř 30 %.

Z praktického hlediska však není možné tento model použít. Čas pro zpracování požadavku se liší dle aktuálního vytížení webové služby a není zaručeno, že požadavek bude zpracován. Navíc se celý systém stává závislým na kvalitním internetovém připojení.

5.4.7 Porovnání všech výsledků

Následující tabulka 5.9 porovnává jednotlivé architektury na základě vážených průměrů přes jednotlivé interpunkční třídy. Zároveň obsahuje také počty parametrů jednotlivých architektur a čas průchodu pro jeden bloku textu o velikosti 512 tokenů.

architektura	P [%]	R [%]	F1 [%]	param.	inf. čas [ms]
LSTM + BiLSTM + linear layers	51.5	34.5	41.3	8.5M	3
ELECTRA-Small trénovaný od počátku	75.3	76.8	**76.0**	13.6M	11
ELECTRA-Base trénovaný od počátku	75.3	75.5	75.4	109M	60
Natrénovaný ELECTRA-Small z [34]	73.4	**78.7**	**76.0**	13.6M	11
Natrénovaný BERT z [35]	75.4	74.9	75.2	110M	61
GPT-3-v editačním módu	**80.5**	54.6	65.1	-	-

Tabulka 5.9: Porovnání jednotlivých architektur v režimu blokového zpracování

Nejjednodušší zkoumanou architekturou je RNN, která dosahuje F1 skóre 41,3 % a má 8,5 milionu parametrů. Tato síť je také nejrychlejší s dobou inference 3 ms.

Modely ELECTRA-Small a ELECTRA-Base, trénované od počátku, dosahují podstatně lepších výsledků než RNN, přičemž F1 skóre se pohybuje mezi 75,4 %

a 76,0 %. Zároveň ELECTRA-Small obsahuje pouze 13,6 milionu parametrů, zatímco ELECTRA-Base má 109 milionů parametrů. Doba inference se zvyšuje na 11 ms pro ELECTRA-Small a 60 ms pro ELECTRA-Base.

Další zkoumané architektury zahrnují dva předtrénované modely typu ELECTRA-Small a BERT. Tyto modely dosahují F1 skóre 76,0 %, respektive 75,2 %, s 13,6 miliony parametrů pro ELECTRA-Small a 110 miliony parametrů pro BERT.

Webová služba využívající GPT-3 model v editačním módu dosahuje nejvyšší přesnosti (80,5 %), ale její recall (54,6 %) je výrazně nižší než u ostatních zkoumaných modelů. Přesné parametry tohoto GPT-3 modelu přitom nejsou veřejně dostupné.

Z výsledků výsledků vyplývá, že nejlepších hodnot dosahují ELECTRA-Small modely. Zároveň mají výrazně menší počet parametrů než varianta base, což má za následek i nižší inferenční čas. Vzhledem k implementačním výhodám je přitom vhodné použít ELECTRA-Small trénovaný od počátku.

5.5 Úspěšnost ve streamovacím režimu

Streamovací a blokový režim zpracování jsou velice odlišné. Zatímco v blokovém režimu lze doplnit interpunkci pro až 512 tokenů během jediného dopředného průchodu, tak v režimu streamování musí být proveden dopředný průchod pro každý token respektivě slovo zvlášť. To má za následek výrazné zvýšení výpočetního času. Pokud by byl proveden dopředný průchod pro každý z tokenů zvlášť, došlo by k jeho 512 násobnému prodloužení.

Průměrný počet slov u většiny evropských jazyků se přitom pohybuje v rozmezí 100 až 200 slov za minutu v závislosti na stylu mluvení a délce slova. Z toho vyplývá, že maximální čas pro zpracování každého slova se pohybuje mezi 300 ms až 600 ms. ELECTRA model tak se značnou rezervou tento předpoklad splňuje, protože jeho inferenční čas pro blok o velikosti 512 tokenů je 11ms.

max. levý kont.	max. pravý kont.	P	R	F1
100	1	73.3	67.4	70.2
100	2	75.0	72.5	73.7
100	3	75.3	74.2	74.7
100	4	75.2	75.1	75.1
100	5	75.5	75.5	75.5
100	6	75.8	75.8	75.8
100	10	**76.0**	76.0	**76.0**
100	100	75.6	73.7	74.6
blokové zpracování bez překryvu		75.3	**76.8**	**76.0**

Tabulka 5.10: Porovnání velikosti kontextu ve streamovacím režimu

Z analýzy výsledků, které jsou prezentovány v tabulce 5.10, vyplývá, že použití pouze jednoho slova jako pravého kontextu je nedostatečné. V tomto případě model

dosahuje F1 skóre pouhých 70,2 %, což představuje výrazný rozdíl oproti výsledkům, které byly dosaženy při použití 10 slov, kde model dosahuje F1 skóre 76 %, což se vyrovnává zpracování v blokovém režimu.

Naopak, použití více než 10 slov jako pravého kontextu již nevede ke zlepšení výsledků modelu. Při příliš velkém kontextu dokonce dochází ke zhoršení výsledků. Na základě těchto zjištění lze konstatovat, že optimální délka pravého kontextu se nachází v rozmezí tří až čtyř slov, kde model dosahuje dobrých výsledků a zároveň nedochází k významnému zpoždění při generování přepisu.

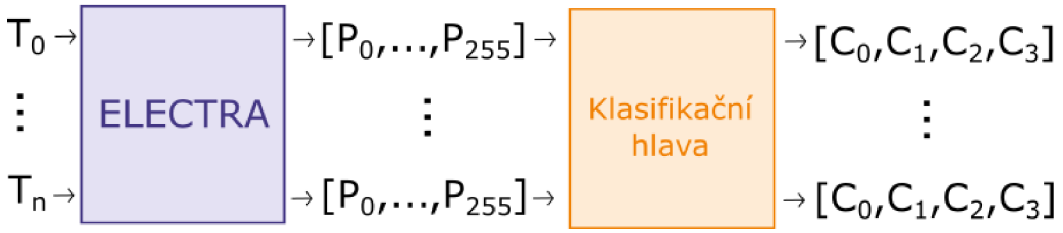
5.6 Optimalizace modelu

Z výsledků v předchozích experimentech vyšel jako nejlepší model ELECTRA-Small. Vzhledem k tomu byly všechny následující optimalizační experimenty provedeny pouze na tomto jediném modelu.

5.6.1 Vliv velikost klasifikační hlavy

V rámci porovnání modelů jsem pro všechny modely využíval klasifikační hlavu složenou ze dvou sekvenčně zapojených lineárních vrstev. V následujícím experimentu jsem vyhodnotil 5 jiných klasifikačních hlav.

První čtyři varianty využívají stejné sekvenční propojení, avšak s odlišným počtem lineárních vrstev. Poslední varianta klasifikačního hlavy obsahuje pouze jednu LSTM vrstvu.



Obrázek 5.3: Klasifikace tokenů pomocí ELECTRA modelu a připojené klasifikační hlavy

Ve všech případech je vstupem do klasifikační hlavy příznakový vektor (pro každý token) o velikosti 256 a na výstupu se provádí klasifikace do jedné ze čtyř interpunkčních tříd. U hlav s vyšším počtem lineárních vrstev se v první vrstvě rozšiřuje příznakový vektor na velikost 512 a následně se postupně redukuje až na výsledný vektor reprezentující interpunkční třídy. Všechny lineární vrstvy využívají aktivační funkci SELU.

Výsledky ukazují, že počet vrstev v klasifikační hlavě nemá významný vliv na úspěšnost modelu. S narůstajícím počtem vrstev dochází dokonce k mírnému zhoršení. Předpoklad použití dvouvrstvé klasifikační hlavy byl proto správný.

Klasifikační hlava	P [%]	R [%]	F1 [%]
1 lineární vrstva	75.8	74.6	76.1
2 lineární vrstvy	75.3	76.8	76.0
3 lineární vrstvy	75.6	75.8	75.7
4 lineární vrstvy	75.5	76.0	75.8
LSTM vrstva	75.5	76.0	75.8

Tabulka 5.11: Porovnání klasifikačních hlav

5.6.2 Kvantizace parametrů modelu

Kvantizace parametrů je proces, při němž dochází k převodu konkrétních parametrů v neuronové síti z původních datových typů s vyšším počtem bitů (např. 16bitová čísla s plovoucí řádovou čárkou) na datové typy, které potřebují nižší počet bitů pro uložení (například 8bitová celá čísla). Kvantizace modelu tedy umožňuje značné snížení velikosti modelu, což je užitečné při použití na zařízeních, které mají omezenou paměť. Zároveň tím dochází ke zvýšení rychlosti modelu, kdy nižší bitová reprezentace snižuje nároky na výpočetní prostředky

Naopak hlavní nevýhodou kvantizace je snížení přesnosti modelu, dané snížením přesnosti reprezentace jeho jednotlivých parametrů. V každém konkrétním případě je tak tedy nutné zvážit, zda snížení velikosti a tím i zvýšení rychlosti modelu může kompenzovat snížení jeho přesnosti.

Pro kvantizaci modelu jsem využil připravenou funkci ”quantize_dynamic”, která je součástí knihovny PyTorch. Kvantizaci jsem následně provedl voláním funkce:

```
quantize_dynamic(model, {torch.nn.Linear}, dtype=torch.qint8)
```

Ve volání funkce jsem nejdříve zvolil, pro který model má být kvantizace provedena. Následně jsem specifikoval, pro které vrstvy má být kvantizace provedena, a jaký je cílový datový typ. V mém případě jsem provedl kvantizaci všech lineárních vrstev modelu ELECTRA-Small a výsledným datovým typem byl qint8[4].

Výsledkem kvantizace byl model, jehož velikost se snížila na polovinu (z 53 MB na 27 MB) a inferenční čas zůstal stejný jako u modelu bez kvantizace.

Na testovací sadě model dosáhl výsledků uvedených v následující tabulce 5.12:

	P [%]	R [%]	F1 [%]
otazník	46.0	36.8	40.9
tečka	76.6	71.0	73.7
čárka	73.8	84.8	78.9
jedna třída	88.4	92.0	90.2

Tabulka 5.12: Výsledky dosažené pomocí kvantizace modelu

[4]Datový typ v knihovně PyTorch, který reprezentuje desetinné číslo po kompresi.

Z výsledků vyplývá, že model se mírně zhoršil u všech tříd. U modelu ELECTRA-Small je jedinou skutečnou výhodou kvantizace snížení paměťové náročnosti modelu, která ovšem nepřevažuje nad tím, že se zhorší výsledky. Ve finální verzi APR modulu proto není kvantizace prováděna.

6 Výsledky pro streamované ASR přepisy

Ve všech předchozích experimentech byla použita testovací data odpovídající textům bez chyb se správně anotovanou interpunkcí. Šlo o ručně (člověkem) vytvořené přepisy rozhlasových pořadů.

Avšak v reálném provozu bude navržený APR modul doplňovat interpunkci do výstupu ASR systému, který může obsahovat chyby. Ty mohou negativně ovlivňovat chování APR modulu a snižovat jeho přesnost.

Pro tyto účely jsem vytvořil novou testovací sadu sestavenou z audio nahrávek a odpovídajících automatických přepisů s ručně doplněnou interpunkcí. Tato sadda se skládá z celkem 10 komentářů z rozhlasu o aktuálních světových událostech (předpřipravená řeč) a 5 televizních politických debat (spontánní řeč).

V prvním případě jde o pořad Názory a argumenty a ve druhém případě o pořad Partie vysílaného televizí Prima. U spontánní řeči se očekává vyšší počet chyb v přepisu, protože často dochází k překryvu řeči více osob, což může vést k nesprávnému přepisu řeči.

Rozhlasové komentáře obsahují 31932 slov, 53 otazníků, 2113 teček a 1954 čárek. Televizní debaty obsahují 39807 slov, 289 otazníků, 2724 teček a 4768 čárek.

Automatické přepisy těchto dat byly vytvořeny pomocí ASR systému pro titulkování televizních a rádiových přenosů. Tento systém využívá RNN-T [36] architekturu a byl natrénován na více než 15 000 hodinách českých audio nahrávek.

Latence systému se pohybuje kolem jedné sekundy a dosahuje chybovosti (míra Word Error Rate [37]) 3,9 % pro komentáře a 5,4 % pro televizní debaty.

V rámci provedeného experimentu navíc porovnávám navržený APR modul (s nejlepším modelem typu ELECTRA-Small trénovaným od počátku) s APR modulem publikovaným v [1]. Ten je také navržen pro češtinu a zpracování datových streamů, využívá ovšem RNN s LSTM vrstvami. Klíčovou odlišnost představuje skutečnost, že vstup do toho systému je tvořen nejen slovy (textem), ale i prosodickými příznaky. Konkrétně se jedná o informaci o umístění ticha mezi slovy, frekvenci vstupních slov a výskyt nádechů. Oba moduly přitom pracují se stejným budoucím kontextem zahrnujícím tři slova.

6.1 Porovnání

Výsledky pro Názory a argumenty jsou uvedeny v tabulce 6.1 a výsledky pro debaty v tabulce 6.2. Celkově z nich vyplývá, že navržený modul dosahuje oproti referenčnímu systému výrazně lepších výsledků v metrice F1.

Konkrétně z výsledků pro komentáře je pak patrné, že navržený modul s kontextem 100 slov vlevo a 3 slova vpravo dosahuje výrazného zlepšení u metriky recall, a to až o 50 % u otazníků. F1 metrika je výrazně vyšší i v případě určení interpunkce bez ohledu na typ znaménka, kde navržený modul dosahuje hodnoty 84,3 % a referenční modul pouze hodnoty 73,4 %. Výsledky obou modelů pak shodně ukazují, že je jednodušší určit slovo, po kterém má následovat interpunkce, než správně určit přesné interpunkční znaménko.

	pořad Názory a argumenty					
	navržený APR modul			APR modul z [1]		
	P [%]	R [%]	F1 [%]	P [%]	R [%]	F1 [%]
otazník	31.7	59.4	**41.4**	40.0	7.5	12.7
tečka	82.6	66.7	**73.8**	69.6	54.0	60.8
čárka	60.2	82.3	**69.5**	61.7	66.8	64.1
vážený průměr	68.7	73.8	**71.2**	65.2	59.4	62.1
jedna třída	80.7	88.3	**84.3**	77.5	69.8	73.4

Tabulka 6.1: Detailní porovnání navrženého APR modulu s APR modulem z [1] na pořadu Názory a argumenty.

U televizních debat došlo k mírnému poklesu u F1 metriky. Navržený APR model dosáhl hodnoty 69,4 %, zatímco referenční systém dosáhl 71,6 %. Hlavním důvodem těchto celkově horších výsledků jsou výrazně horší dílčí výsledky pro tečky, které souvisí s tím, že televizní debaty často obsahují dlouhé monology. Tyto monology se skládají ze složitých a umělé navazovaných sekvencí vět, u kterých je obtížné (bez akustických informací) určit, zda by měly být rozděleny na více kratších souvětí nebo naopak na několik dlouhých. Tuto hypotézu podporují i výsledky dosažené pro jednu globální interpunkční třídu, které jsou u navrženého APR modulu naopak lepší o více než 15 %.

	spontánní TV debaty					
	navržený APR modul			APR modul z [1]		
	P [%]	R [%]	F1 [%]	P [%]	R [%]	F1 [%]
otazník	48.5	39.9	**43.8**	75.8	22.8	35.0
tečka	67.5	35.7	**46.7**	68.8	58.4	63.2
čárka	75.8	84.1	**79.7**	83.3	72.7	77.6
vážený průměr	73.2	65.9	69.4	78.2	66.0	**71.6**
jedna třída	93.1	85.3	**89.0**	92.2	75.9	73.3

Tabulka 6.2: Detailní porovnání navrženého APR modulu s APR modulem z [1] na televizních debatách.

7 Závěr

V rámci této práci byl navržen vlastní modul umožňující automatické generování interpunkce v textovém výstupu (přepisu řeči) z ASR systému, které pracují v reálném čase a zpracovávají streamovaná data.

Modul využívá architekturu typu ELECTRA ve verzi Small a umožňuje doplňovat do rozpoznaného textu, čárky, tečky a otazníky. Jako tokenizér byl zvolen modul SentencePiece, využívaný již existujícími ASR systémy vyvíjenými na TUL, který má zároveň výraznou převahu v rychlosti oproti standardně používaným tokenizérům. S ohledem na výběr tokenizéru musela být dostupná knihovna pro trénování přepsána tak, aby zvolený tokenizér podporovala.

Pro trénování navrženého APR modulu byl připraven a použit dataset o velikost 25 GB, který obsahoval přepisy různých typů pořadů. Trénování parametrů použité architektury na těchto datech trvalo 4 dny a to na velmi výkonném hardwaru poskytnutém Metacentrem Cesnet. Poté následovalo doladění parametrů modelu, kdy byla k ELECTRA-Small modelu připojena klasifikační hlava sestávající ze dvou lineárních vrstev. Během tohoto procesu se s velkým learning ratem upravovaly parametry klasifikační hlavy a s malým všechny ostatní předtrénované parametry výchozího modelu.

V experimentu zaměřeném na zpracování streamovaných dat pak bylo zjištěno, že pro optimální výsledky a malou odezvu modulu je možné pracovat s pravým kontextem o velikosti pouze třech slov. Jako levý kontext bylo využíváno maximálně 100 slov.

Navržená metoda byla také experimentálně porovnána s existujícím modulem popsaným v [1], který v současné době využívá firma Newton Technologies ve svých produktech. Tento modul se liší od navrženého tím, že pracuje nejen s textovými příznaky, ale také s prosodickými příznaky získanými ze zvukové nahrávky. Kromě toho využívá rekurentní neuronovou síť. Pro porovnání obou modulů byl vytvořen speciální dataset, který obsahoval televizní a rozhlasové přepisy (neopravené člověkem a tedy obsahující chyby v přepisu). Modul navržený v této práci dosáhl na těchto datech celkově výrazně lepších výsledků. To osvědčuje jeho praktickou použitelnost.

Na základě výše uvedených a provedených výzkumných prací a dosažených výsledků byl vytvořen také odborný článek [38], který byl již přijat na prestižní mezinárodní konferenci Interspeech 2023. Ta představuje největší mezinárodní konferenci v oboru počítačového zpracování řeči. Přijetí článku na tuto konferenci v rámci náročného recenzního řízení potvrzuje kvalitu, novost a přínos této diplomové práce i pro mezinárodní vědeckou komunitu.

Literatura

[1] P. Hlubik, M. Spanel, M. Bohac, and L. Weingartova, "Inserting punctuation to ASR output in a real-time production environment," in *TSD 2020, Brno, Czechia, September 8-11, 2020*, ser. Lecture Notes in Computer Science, vol. 12284. Springer, 2020, pp. 418--425.

[2] Rezzy Eko Caraka, "Network structure of rnn, lstm, and gru," 2021. [Online]. Available: https://www.researchgate.net/figure/Network-Structure-of-RNN-LSTM-and-GRU_fig2_346410173

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[4] M. A. Tundik, B. Tarjan, and G. Szaszak, "Low latency maxent-and rnn-based word sequence models for punctuation restoration of closed caption data," in *SLSP 2017, Le Mans, France, October 23-25, 2017*, ser. Lecture Notes in Computer Science, vol. 10583. Springer, 2017, pp. 155--166.

[5] J. Huang and G. Zweig, "Maximum entropy model for punctuation annotation from speech," in *ICSLP2002 – Interspeech 2002, Denver, Colorado, USA, September 16-20, 2002*. ISCA, 2002.

[6] W. Lu and H. T. Ng, "Better punctuation prediction with dynamic conditional random fields," in *EMNLP 2010, Massachusetts, USA, October 9-11, 2010*. ACL, 2010, pp. 177--186.

[7] X. Che, C. Wang, H. Yang, and C. Meinel, "Punctuation prediction for unsegmented transcript based on word vector," in *LREC 2016, Portoroz, Slovenia, May 23-28, 2016*. ELRA, 2016.

[8] O. Tilk and T. Alumae, "LSTM for punctuation restoration in speech transcripts," in *Interspeech 2015, Dresden, Germany, September 6-10, 2015*. ISCA, 2015, pp. 683--687.

[9] ——, "Bidirectional recurrent neural network with attention mechanism for punctuation restoration," in *Interspeech 2016, San Francisco, CA, USA, September 8-12, 2016*. ISCA, 2016, pp. 3047--3051.

[10] S. Kim, "Deep recurrent neural networks with layer-wise multi-head attentions for punctuation restoration," in *ICASSP 2019, Brighton, United Kingdom, May 12-17, 2019*. IEEE, 2019, pp. 7280--7284.

[11] B. P. Nguyen, V. B. H. Nguyen, H. Nguyen, P. N. Phuong, T. Nguyen, Q. T. Do, and L. C. Mai, "Fast and accurate capitalization and punctuation for automatic speech recognition using transformer and chunk merging," in *O-COCOSDA 2019, Cebu, Philippines, October 25-27, 2019*. IEEE, 2019, pp. 1--5.

[12] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019*. ACL, 2019, pp. 4171--4186.

[13] Y. Cai and D. Wang, "Question mark prediction by bert," in *APSIPA ASC 2019, Lanzhou, China, November 18-21, 2019*. IEEE, 2019, pp. 363--367.

[14] K. Makhija, T. Ho, and E. S. Chng, "Transfer learning for punctuation prediction," in *APSIPA ASC 2019, Lanzhou, China, November 18-21, 2019*. IEEE, 2019, pp. 268--273.

[15] K. Clark, M. Luong, Q. V. Le, and C. D. Manning, "ELECTRA: pre-training text encoders as discriminators rather than generators," in *ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[16] W. Wang, Y. Liu, W. Jiang, and Y. Ren, "Making punctuation restoration robust with disfluency detection," in *CSCWD 2022, Hangzhou, China, May 4-6, 2022*. IEEE, 2022, pp. 395--399.

[17] Q. Chen, M. Chen, B. Li, and W. Wang, "Controllable time-delay transformer for real-time punctuation prediction and disfluency detection," in *ICASSP 2020, Barcelona, Spain, May 4-8, 2020*. IEEE, 2020, pp. 8069--8073.

[18] S. Kostadinov, "How recurrent neural networks work," *Towards Data Science*, 12 2017. [Online]. Available: https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7

[19] M. Nguyen, "Illustrated guide to lstms and gru's: A step by step explanation," *Towards Data Science*, 11 2018. [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

[20] H. Sharma, "Understanding encoders-decoders with attention based mechanism," *Medium*, 2 2021. [Online]. Available: https://medium.com/data-science-community-srm/understanding-encoders-decoders-with-attention-based-mechanism-c1eb7164c581

[21] R. Karim, "Illustrated: Self-attention," *Towards Data Science*, 11 2019. [Online]. Available: https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

[22] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv:1607.06450*, 2016.

[23] Pinecone, "Batch normalization in deep learning," *Pinecone Blog*, 5 2023. [Online]. Available: https://www.pinecone.io/learn/batch-layer-normalization/

[24] A. Vadapalli, "An investigation of speaker independent phrase break models in end-to-end tts systems," *arXiv preprint arXiv:2304.04157*, 2023.

[25] S. Yadav, "Understanding the bert model," *Medium*, 8 2020. [Online]. Available: https://medium.com/analytics-vidhya/understanding-the-bert-model-a04e1c7933a9

[26] B. Muller. (2022) Bert 101 - state of the art nlp model explained. [Online]. Available: https://huggingface.co/blog/bert-101

[27] B. Ratheesh. (2023) Electra vs bert– a comparative study. [Online]. Available: https://bijular.medium.com/electra-vs-bert-a-comparative-study-36f07ba88e61

[28] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *arXiv:2005.14165*, 2020.

[29] P. Schmid. (2021) Few-shot learning in practice: Gpt-neo and the accelerated inference api. [Online]. Available: https://huggingface.co/blog/few-shot-learning-gpt-neo-and-inference-api

[30] Y. Gupta. (2023) How chat gpt works - technical architecture and intuition with examples. [Online]. Available: https://medium.com/nerd-for-tech/gpt3-and-chat-gpt-detailed-architecture-study-deep-nlp-horse-db3af9de8a5d

[31] T. Kudo and J. Richardson, "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," in *EMNLP 2018, Brussels, Belgium, October 31 - November 4, 2018*. ACL, 2018, pp. 66--71.

[32] A. Vidhya. (2023) Confusion matrix, accuracy, precision, recall, f1 score. [Online]. Available: https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd

[33] G. Marcus, "Deep learning: A critical appraisal," *arXiv preprint arXiv:1711.05101*, 2018. [Online]. Available: https://arxiv.org/pdf/1711.05101.pdf

[34] M. Kocian, J. Naplava, D. Stancl, and V. Kadlec, "Siamese BERT-based model for web search relevance ranking evaluated on a new czech dataset," in *AAAI 2022, IAAI 2022, EAAI 2022, Virtual Event, February 22 - March 1, 2022*. AAAI Press, 2022, pp. 12 369--12 377.

[35] J. Sido, O. Prazak, P. Priban, J. Pasek, M. Sejak, and M. Konopik, "Czert - Czech BERT-like model for language representation," in *RANLP 2021, Virtual Event, September 1-3, 2021*. INCOMA Ltd., 2021, pp. 1326--1338.

[36] D. Albesano, J. Andrés-Ferrer, N. Ferri, and P. Zhan, "On the prediction network architecture in rnn-t for asr," in *Interspeech 2022 Incheon, Koream 18-22 September 2022*, 2022.

[37] T. von Neumann, C. Boeddeker, K. Kinoshita, M. Delcroix, and R. Haeb-Umbach, "On word error rate definitions and their efficient computation for multi-speaker speech recognition systems," 2022.

[38] M. Poláček, P. Červa, J. Žďánský, and L. Weingartová, "Online punctuation restoration using electra model for streaming asr systems," in *Proceedings of the InterSpeech 2023*, 2023.