VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

IMPLEMENTÁCIA JEDNODUCHÉHO ROZPOZNÁVAČA ŘEČI VO WEBOVOM PREHLIADAČI

IMPLEMENTATION OF SIMPLE SPEECH RECOGNIZER IN A WEB BROWSER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JAKUB CRKOŇ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. IGOR SZŐKE, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Crkoň Jakub**
Program: Informační technologie
Název: **Implementace jednoduchého rozpoznávače řeči ve webovém prohlížeči**
Implementation of Simple Speech Recognizer in a Web Browser
Kategorie: Zpracování řeči a přirozeného jazyka
Zadání:

1. Nastudujte základy implementace aplikací bežících na webovém prohlížeči (JavaScript, TypeScript) a základy rozpoznávání řeči.
2. Implementuje jednoduchý rozpoznávač řeči (extrakce příznaků, forward pass přes neuronovou síť, dekodér). Pokud to bude možné, využijte dostupných knihoven.
3. Tam kde to půjde, využijte akceleraci (např. maticové násobení na GPU).
4. Vyhodnoťte úspěšnost a hlavně časové a paměťové nároky. Navrhněte směry dalšího vývoje.
5. Vyrobte A2 plakátek a cca 30 vteřinové video prezentující výsledky vaší práce.

Literatura:

- Dle pokynů vedoucího
- http://www.fit.vutbr.cz/study/DP/BP.php.cs?id=19075&file=t

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

Vedoucí práce: **Szőke Igor, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 18. května 2020

Abstrakt

Cieľom tejto práce je vytvorenie jednoduchého rozpoznávača reči pre webový prehliadač. Práca popisuje základné komponenty nutné pre implementáciu rozponávača reči a venuje sa technikám, ktoré využíva pre optimalizáciu procesu rozpoznávania reči vo webovom prehliadači. Práca sa najskôr zameriava na teoretické priblíženie problematiky rozpoznávača. Opisuje jeho jednotlivé časti a princípy, ktoré využívajú. Ďalej práca popisuje dizajn, implementáciu a spôsob akcelerácie rozpoznávača za použitia obmedzených výpočetných prostriedkov webového prehliadača. Implementácia je rozdelená do modulov tvoriacich knižnicu pre použitie vo webových prehliadačoch. Knižnica je ľahko rozšíriteľná a integrovateľná v ľubovolnej webovej aplikácií. V závere rozoberá potencionálne smery vývoja a použiteľnosti tohto projektu.

Abstract

The goal of this project is to implement simple speech recognizer for web browser. This paper describes fundamental components required for implementing speech recognizer and techniques which are used for optimization process of speech recognition in web browser. At first, the paper focuses on introduction of speech recognition theory. It describes individual parts and principles of speech recognizer. In next section, thesis describes design, implementation and principles of acceleration of speech recognizer with limited computing resources of web browser. The implementation is divided into modules making up the library for usage in web browser. The library is easily extendable and usable in various web applications. Finally, it discusses potential directions of development and usability of this project.

Kľúčové slová

Umelá inteligencia, rozpoznávanie reči, digitálne spracovanie signálu, strojové učenie, typescript, javascript, webový prehliadač, dynamický dekodér

Keywords

Artificial intelligence, speech recognition, digital signal processing, machine learning, typescript, javascript, web browser, dynamic decoder

Citácia

CRKOŇ, Jakub. *Implementácia jednoduchého rozpoznávača řeči vo webovom prehliadači.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Igor Szőke, Ph.D.

Implementácia jednoduchého rozpoznávača řeči vo webovom prehliadači

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Igora Szőkeho Ph.D. Uviedom som všetky zdroje a publikácie, z ktorých som čerpal.

.....
Jakub Crkoň
28. mája 2020

Poďakovanie

Ďakujem vedúcemu práce pánovi Ing. Igorovi Szőkemu za pomoc pri vypracovaní práce, veľkú trpezlivosť a ochotu.

Obsah

Kapitola 1

Úvod

Oblasť spracovania reči je už po dlhú dobu nespochybniteľne jedna z najviac rozoberaných tém v informatike. Je využiteľná v mnohých oblastiach od prevodu reči na text až po využitie v inteligentných osobných asistentoch. Vďaka novým technológiám sa možnosti spracovania reči neustále rozširujú aj na zariadenia, v ktorých to doposiaľ nebolo možné. Avšak, väčšina rozpoznávania reči je stále vykonávaná na cloudových serveroch, ktoré majú naviac často špecializované výpočtové vybavenie. Možnosti upraviteľného rozpoznávania, v reálnom čase bez internetového pripojenia a spoplatneného API, sú teda značne obmedzené. Táto práca sa zameriava práve na toto obmedzenie. Jej cieľom je vytvorenie jednoduchej, prenosnej knižnice, ktorá zaručí komplexný prístup k rozpoznávaču vo webovom prehliadači.

Aj keď sa v posledných rokoch technológie a aj výpočtové možnosti webových prehliadačov značne rozšírili, stále nie sú porovnateľné s technológiami, ktoré sú priamo uspôsobené na výpočetne náročné úlohy, medzi ktoré patrí aj rozpoznávanie reči. Pri tvorbe výpočtovo náročných aplikácií je teda nutné sa sústrediť na optimalizáciu a paralelizáciu využitých algoritmov. Webové prehliadače to umožňujú vďaka paralelizácií za pomoci multithreadingu a akcelerácie výpočtov za použitia grafických shaderov.

Druhá kapitola tejto práce sa zameriava na objasnenie metód a techník využitých pri implementácií rozpoznávača reči. Vysvetľuje spôsob uloženia zvukového signálu na diskrétnych zariadeniach. Ďalej popisuje, ako sú zvukové dáta spracované, extrakciu príznakov, vyhodnotenie príznakov za pomoci neurónovej siete a základné princípy rozpoznávania reči za pomoci dekodéru.

V tretej kapitole práca rozoberá implementáciu. Obsahuje informácie o použitých technológiách, algoritmoch, využitých knižniciach a obsahuje popis štruktúry vytvorenej knižnice.

Posledná kapitola sa zaoberá vyhodnotením dosiahnutých výsledkov a možnostiam ďalšieho vývoja projektu.

1.1 Motivácia

Motiváciou tohto projektu bolo vytvorenie jednoduchého offline rozpoznávača reči spustiteľného vo webovom prehliadači, bez nutnosti online komunikácie, so vzdialeným serverom. Vďaka tomu by mal byť dostupný na všetkých zariadeniach podporujúcich webový prehliadač. Tento spôsob implementácie taktiež odstraňuje niektoré bezpečnostné riziká, ktoré prináša internetová komunikácia a to napríklad odchyt odoslaných paketov, ktoré obsahujú citlivé dáta.

V čase implementácie tohto projektu existovala iba jedna alternatíva ponúkajúca offline rozpoznávanie reči vo webovom prehliadači. Projekt Pocketsphinx [1] je open-sourcová knižnica, ktorá, ako jediná, ponúka totožnú funkcionalitu. Avšak, doposiaľ nie je optimalizovaná pre využitie so slovníkom obsahujúcim väčší počet slov. Knižnica tiež nepracuje s dostupnou grafickou akceleráciou maticových operácií, ktorá značne zlepšuje rýchlosť rozpoznávania.

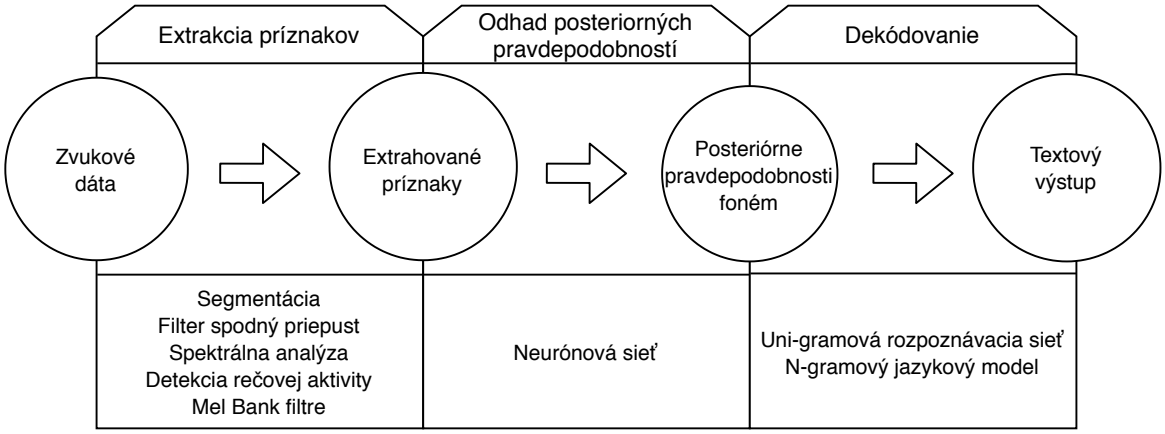
Vytvorením totožnej funkcionality pre platformu android sa zaoberá práca [10], ktorá pracuje s totožnými nástrojmi na rozpoznávanie príznakov v reči. Po podrobnom naštudovaní problematiky a implementácie práce pre platformu Android bolo nutné vykonať rozsiahly prieskum webových technológií pre webové prehliadače a ich možné využitie v úlohe spracovania reči. Nadobudnuté znalosti boli využité pri implementácií tejto práce.

[1] https://syl22-00.github.io/pocketsphinx.js/

Kapitola 2

Rozpoznávanie reči

Táto kapitola má za cieľ oboznámiť čitateľa s metódami rozpoznávania reči, ktoré boli využité v tejto práci. Všeobecne povedané, úlohou rozpoznávača je získanie prepisu postupnosti slov zo zaznamenaného audio signálu. Aj keď je úloha na prvý pohľad zrejmá, jej dosiahnutie nie je také priamočiare. Zväčša je nahraný zvukový signál, ktorý je ovplyvnený niekoľkými negatívnymi faktormi vrátane dialektu a tónu rečníka, ale aj rušivými zvukmi okolia. Vďaka týmto negatívnym vplyvom sa pre rozpoznávanie reči osvojil štatistický prístup. Rozpoznávanie je väčšinou založené na pravdepodobnostiach výskytov určitých foném v malých segmentoch reči, ktoré sú nazývané rámce. Táto časť rozpoznávača je nazývaná akustický model. O rozpoznanie slov, tvorených postupnosťou pravdepodobností foném, sa zasa zaoberá jazykový model. Jazykový model musí taktiež riešiť problémy s nejasnými hranicami slov a slovami, ktoré znejú rovnako, no majú rozličný význam.



Obr. 2.1: Zjednodušená schéma automatického rozpoznávania reči založeného na dynamickom jazykovom modeli a posteriórnych pravdepodobnostiach foném odhadovaných neurónovou sieťou.

2.1 Spracovanie audio signálu

Zvukový signál reprezentuje vibrácie vzduchu alebo iného prenosového média, ktoré boli vyprodukované napríklad ľudskými hlasivkami. Táto sekcia opisuje jednotlivé kroky spracovania tohto signálu, ktoré pripravujú dáta na ďalšie spracovanie. Cieľom spracovania

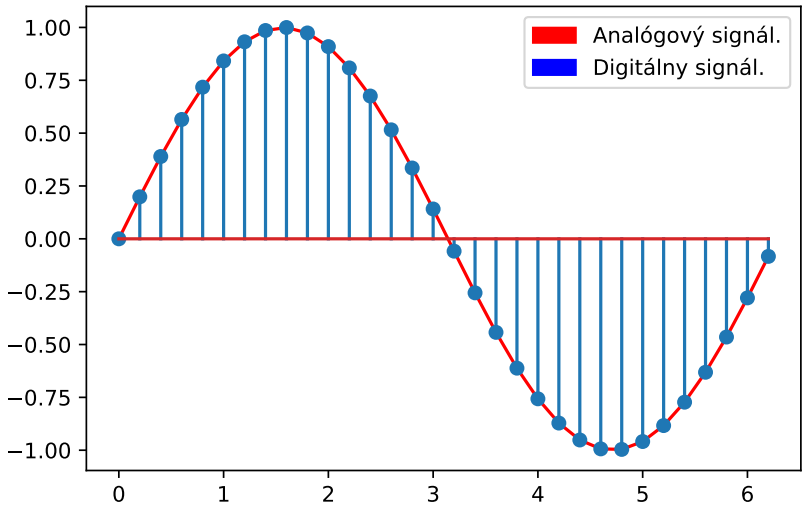
signálu je získať aproximáciu frekvencií vyskytujúcich sa na zvukovej nahrávke v určitom čase.

2.1.1 Predspracovanie signálu

Prvým krokom rozpoznávania reči je získanie audio signálu pre spracovanie. Následne je signál prevzorkovaný do vzorkovacej frekvencie, ktorá je vhodná pre spracovanie reči, a rozdelený do menších rámcov. Na jednotlivé rámce je aplikovaná okenná funkcia, ktorá má za úlohu zlepšenie výsledkov spektrálnej analýzy redukciou okrajových častí rámcu.

Digitálna reprezentácia signálu

Bežné zariadenia nahrávajú zvuk ako analógový signál, no pre jeho uloženie do pamäte a následnú manipuláciu je nutné ho previesť do diskrétnej podoby. O tento prevod sa stará analógovo-digitálny prevodník, ktorý prevzorkuje spojitý signál na diskrétny.



Obr. 2.2: Analógový signál a jeho diskrétna reprezentácia

V pamäti je vzorka signálu uložená v PCM (*pulse code modulation*) formáte. Hodnota každej vzorky je reprezentovaná diskrétnou hodnotou - zvyčajne vyjadrenou 16-bitovým celým číslom. Vzorka predstavuje amplitúdu signálu v určitom časovom momente. Ďalšou vlastnosťou diskrétneho signálu je vzorkovacia frekvencia. Definuje počet diskrétnych vzoriek zaznamenaných v jednej sekunde audio nahrávky. Signál je možné nahrávať vo viacerých formátoch. Stereo nahrávanie umožňuje uloženie signálu vo formáte, kedy sa pri prehrávaní zdá, že zvuk prichádza z určitého smeru. Stereo nahrávanie však vyžaduje dva mikrofóny a ukladá dvojnásobné množstvo dát. V rozpoznávaní reči je preto uprednostňované mono nahrávanie.

Prevzorkovanie signálu

Vzhľadom na to, že rozpoznávač sa snaží rozpoznať ľudskú reč, je možné výrazne obmedziť frekvencie signálu na tie, ktoré sú podstatné. Rozsah frekvencie zvuku vydávaného ľudskými

hlasivkami sa koncentruje medzi 300 Hz až 3800 Hz [2]. Vďaka tejto skutočnosti nie je nutné spracovávať frekvencie vyššie, ako je horná hranica tohto frekvenčného pásma.

Pre rozpoznávanie je v tomto prípade používaná vzorkovacia frekvencia 8000 Hz. Táto vzorkovacia frekvencia tvorí prijateľný kompromis medzi presnosťou a potrebnou výpočtovou náročnosťou. Taktiež je daná Shannonovým teorémom popísaným nižšie. Webové prehliadače však nezaručujú túto optimálnu vzorkovaciu frekvenciu. Prehliadač používa natívnu vzorkovaciu frekvenciu nahrávacieho zariadenia, ktorá je vo väčšine prípadov 44100 Hz alebo 48000 Hz. Pre podporu ľubovolného zariadenia je teda nutné implementovať vlastnú metódu na prevzorkovanie signálu.

FIR Filter - Spodná priepusť

Filter s konečnou impulznou odozvou je diskrétny lineárny filter, ktorý sa v konečnom čase ustáli na hodnote nula. V tejto implementácií je využitý ako dolná priepusť na odstránenie nepotrebných frekvencií za účelom vyhnutia sa aliasingu signálu pri podvzorkovaní. Spodná priepusť prepúšťa cez filter iba signál pod určitou frekvenčnou hranicou, ktorá je v našom prípade 4000 Hz. Môžeme ho vyjadriť rovnicou

$$x[n] = \sum_{k=0}^{N} h[k]x[n-k] \tag{2.1}$$

kde, x predstavuje vstupný signál, h je impulzná odozva, N je rád filtru a y je výstupný signál po aplikovaní filtru.

Shannonov teorém

Shannonov teorém udáva, že vzorkovacia frekvencia diskrétneho signálu musí byť najmenej dvakrát väčšia ako frekvencia, ktorou chceme signál prenášať. Preto je v našom prípade použitá frekvencia 8000 Hz. V prípade nedodržania tejto podmienky je kvalita informácií v prenášanom signále výrazne znehodnotená.

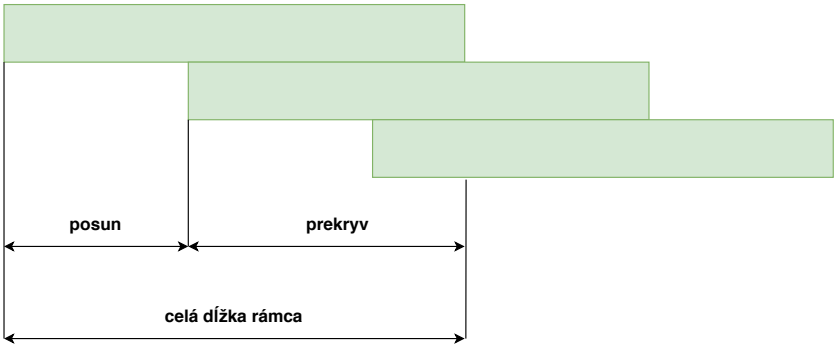
2.2 Extrakcia príznakov

Extrakcia príznakov je všeobecne v rozpoznávaní proces, ktorý je zameraný na zníženie množstva dát nutných k ich klasifikácií. Menšie množstvo dát urýchľuje klasifikáciu, a tak zlepšuje celkový čas rozpoznávania. Ďalej extrakcia umožňuje zamerať sa na dôležité dáta a tie nepotrebné zahodiť.

2.2.1 Segmentácia

Prvým krokom extrakcie príznakov je segmentácia. Vstupom tejto časti rozpoznávača je zvukový signál, ktorého predspracovanie sme popísali v kapitole vyššie. Tento signál je nutné rozdeliť na rámce, ktoré obsahujú malé časové úseky signálu, aby bolo možné určiť, aké frekvencie vydával človek, či zariadenie v danom časovom momente. Bežná dĺžka rámca pre rozpoznávanie reči je 20-25 milisekúnd. Jednotlivé rámce sa tiež prekrývajú v dĺžke 10-15 milisekúnd.

Pri vzorkovacej frekvencií 8000 Hz je dĺžka jedného vzorku $1000/8000 = 0.125$ ms. 25ms dlhý rámec teda obsahuje $25/0.125 = 200$ vzoriek. Z toho vyplýva, že dĺžka posunu je $10/0.125 = 80$ vzoriek.



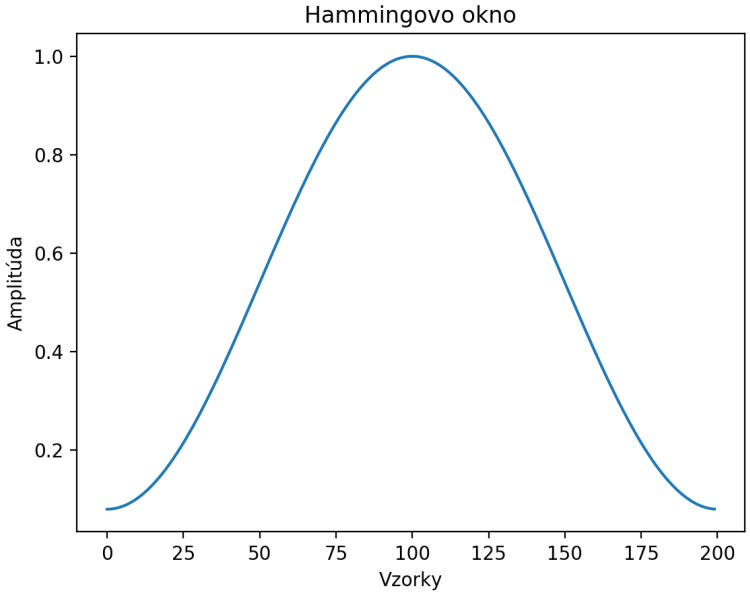
Obr. 2.3: Grafická reprezentácia prekrytia rámcov

2.2.2 Hammingovo okno

Rozkúskovanie signálu na rámce môže spôsobiť, že vlna reprezentujúca signál je odseknutá na mieste, kde dosahuje vrcholnú hodnotu. Tento fakt môže znehodnotiť výsledok spektrálnej analýzy. Tento jav sa nazýva spektrálny únik a je daný charakteristikou Fourierovej transformácie, ktorá periodizuje vstupný signál. Z týchto dôvodov je nutné použiť vhodnú okennú funkciu. V tejto implementácií je použitá okenná funkcia - Hammingovo okno. Táto funkcia oslabuje amplitúdu signálu na okrajoch rámca. Je možné ju vyjadriť nasledujúcou rovnicou a je graficky znázornená na obrázku 2.4.

$$f_h[n] = f[n](0.54 - 0.46\cos(\frac{2\pi n}{N-1})) \tag{2.2}$$

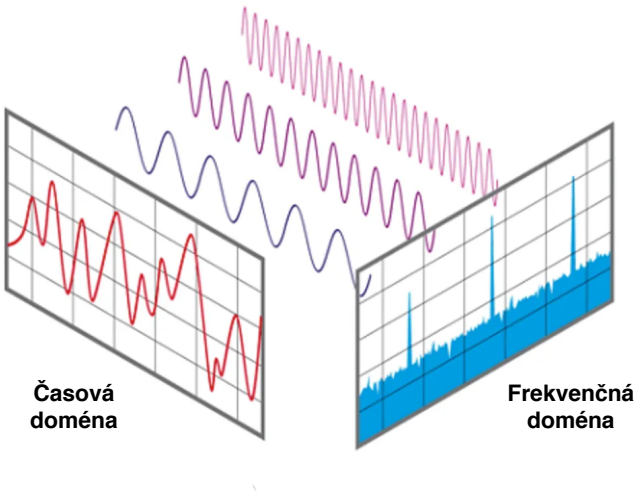
kde, N predstavuje dĺžku rámca a n predstavuje index vzorky.



Obr. 2.4: Grafická reprezentácia Hammingovho okna

2.2.3 Spektrálna analýza

Spektrálna analýza umožňuje rozdelenie vstupného signálu v rámci na jednotlivé periodické signály, z ktorých je zložený [1]. Vďaka tomu je možné zistiť, v ktorých frekvenciách má signál najviac energií. Nakoľko má každý tón, ktorý je vydávaný ľudským hlasovým ústrojenstvom, svoje špecifické frekvenčné zložky je ich rozloženie pre automatizované rozpoznávanie veľmi dôležité.



Obr. 2.5: Grafické znázornenie spektrálnej analýzy

Rýchla Fourierova transformácia

Všeobecne cieľom spektrálnej analýzy je prevod medzi časovou a frekvenčnou doménou signálu. Keďže signál, ktorý spracovávame nie je spojitý, ale diskrétny, táto transformácia je prevedená za pomoci diskrétnej Fourierovej transformácie (DFT).

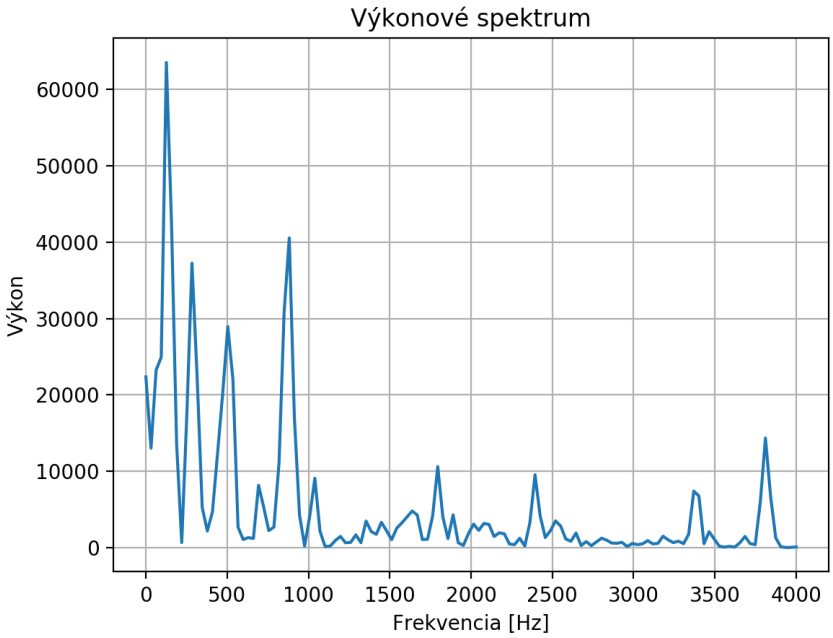
$$D(n) = \sum_{k=0}^{N-1} d(k) e^{-\frac{ink2\pi}{N}} \quad n \in [0, N-1] \tag{2.3}$$

Bežná DFT definovaná formulou 2.3 nie je bežne používaná kvôli jej zložitosti, ktorá je $O(N^2)$. Namiesto toho sa používa jej optimalizovaná verzia nazývaná rýchla Fourierova Transformácia (FFT), ktorej zložitosť je $O(N \log N)$. Tieto optimalizované varianty využívajú periodickosti a symetrie DFT algoritmu na zredukovanie komplexnosti výpočtov.

Výkonové spektrum (*Power spectrum*)

Jednotlivé frekvencie v rámci môžeme vyjadriť ich výkonovým spektrom. Výkonové spektrum rámca je spočítané ako:

$$P_n = \sqrt{x_n^2 + i_n^2}^2 = x_n^2 + i_n^2 \tag{2.4}$$



Obr. 2.6: Výkonové spektrum signálu

2.2.4 Mel bank filtre

Vstupom do mel bank filtrov je výkonové spektrum Fourierovej transformácie jednotlivých rámcov signálu. Cieľom použitia mel bank filtrov je simulovanie ľudského ucha, ktoré lepšie rozlišuje nižšie frekvencie [6]. Silové spektrum je rozdelené na 26 prekrývajúcich sa pásiem, na ktoré je aplikovaných 24 trojuholníkových filtrov, ktoré sa nazývajú banky.

Parametre okenných funkcií, použitých pre výpočet energií, sú závislé na počte použitých filtrov, z ktorých chceme počítať energie vzorkovacích frekvencií a veľkosti vstupných dát. Mel banky pre jednotlivé frekvencie silového spektra sú definované rovnicou 2.5.

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \frac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \leq k \leq f(m) \\ \frac{f(m+1) - k}{f(m+1) - f(m)} & f(m) \leq k \leq f(m+1) \\ 0 & k > f(m+1) \end{cases} \tag{2.5}$$

Kde:

k sú indexy jednotlivých vzoriek silového spektra , $k \in [0, N]$

N je dĺžka silového spektra

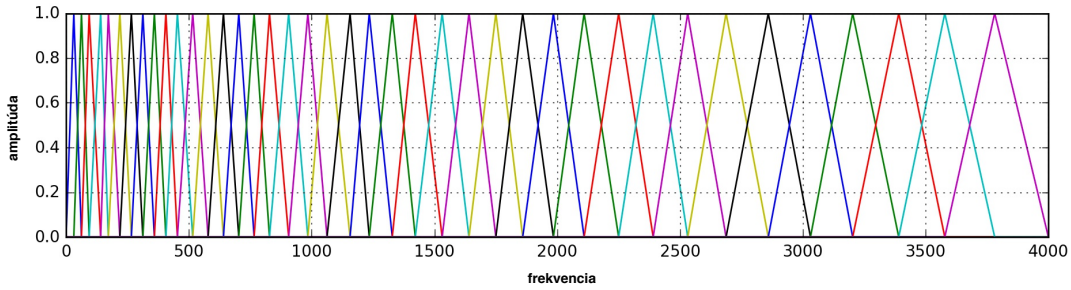
M je počet Mel baniek - v našom prípade 24 baniek

f je pole o dĺžke M + 2

m je index mel bank filtra

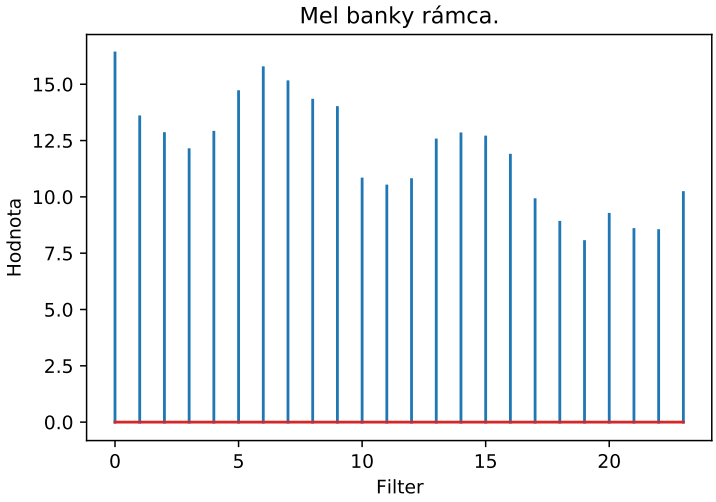
Po výpočte sú jednotlivé hodnoty bánk rámca sčítané a logaritmicky transformované pre použitie v ďalšej časti rozpoznávača.

$$B[m] = \log \sum_{k} = 0^{N} H_m(k) \tag{2.6}$$



Obr. 2.7: Grafické znázornenie mel filtrov - 24 filtrov, vzorkovacia frekvencia 8000 Hz

Výstupom je vektor logaritmov energií v signáli rámca. Týmto je dosiahnuté podstatné zníženie dimenzionality oproti pôvodným dátam.



Obr. 2.8: Hodnoty bank pre rámec

Mel mierka

Mel mierka sa snaží aproximovať akým spôsobom človek v skutočnosti vníma zvuk. Prevod medzi frekvenčnou doménou a mel doménou je prevádzaný za pomoci rovníc 2.7 a 2.8.

$$m = 2595 \log_1 0(1 + \frac{f}{700}) = 1127 \ln(1 + \frac{f}{700}) \tag{2.7}$$

$$f = 700(10^{\frac{m}{2595}} - 1) = 700(e^{\frac{m}{1127}-1)} \tag{2.8}$$

2.2.5 Uniformná normalizácia

Aby sme zistili, či sú špecifické Mel banky jednotlivých rámcov dôležité, ich hodnoty musia byť normalizované. Pri spracovaní nahrávky z audio súboru je táto úloha pomerne jednoduchá. Je ju možné zrealizovať výpočtom priemerov pre jednotlivé banky všetkých rámcov a ich následnou substrakciou. Toto samozrejme nie je možné pre zvuk, ktorý je nahrávaný v reálnom čase. Priemery jednotlivých bánk musia byť počítané počas nahrávania a to môže spôsobiť znehodnotenie prvej časti vzoriek.

Najjednoduchšia adaptácia normalizácie, pri spracovaní v reálnom čase, je obmedzenie počtu použitých vzoriek. Problém s touto metódou je samozrejme voľba správneho N, ktoré predstavuje počet vzoriek braných do úvahy pri tomto výpočte. Presnosť sa zvyšuje súčasne s rastúcim N, no príliš veľké N spôsobí väčší počet znehodnotených vzoriek pri zmene hluku v pozadí. Uniformná normalizácia môže byť spočítaná ako:

$$y_n = x_n - \sum_{i=0}^{n-1} h_i \tag{2.9}$$

Kde, x_n je vstup, y_n je výstup a h sú predošle vstupy.

2.2.6 Exponenciálna normalizácia

Exponenciálna normalizácia adresuje problém znehodnotenia aktuálneho rámca staršími vzorkami. Exponenciálna funkcia $f(x) = a^x$ kde, $a \in (0, 1)$ umožňuje znižovanie váhy akou staršie vzorky ovplyvňujú normalizáciu. Podobne ako uniformná normalizácia vyžaduje správnu voľbu parametru a, ktorý môže byť určený rovnicou:

$$a = \sqrt[t+\frac{f}{s}]{w} \tag{2.10}$$

kde, t vyjadruje rozdiel času od aktuálnej vzorky v sekundách, f vyjadruje vzorkovaciu frekvenciu, s vyjadruje počet vzoriek v jednom kroku a w váhu s akou má vzorka ovplyvňovať aktuálny rámec.

2.3 Odhad fonémových posteriórov

Proces odhadu posteriórov zaisťuje prevod extrahovaných audio príznakov získaných zo signálu na posteriórnej pravdepodonosti foném. V tejto práci je na získavanie týchto pravdepodobností využitý model tvorený doprednou neurónovou sieťou, ktorá bola natrénovaná špecificky na túto úlohu. Táto sekcia opisuje získanie posteriórnych pravdepodobností a predstaví modely, ktoré sú k tomu využité.

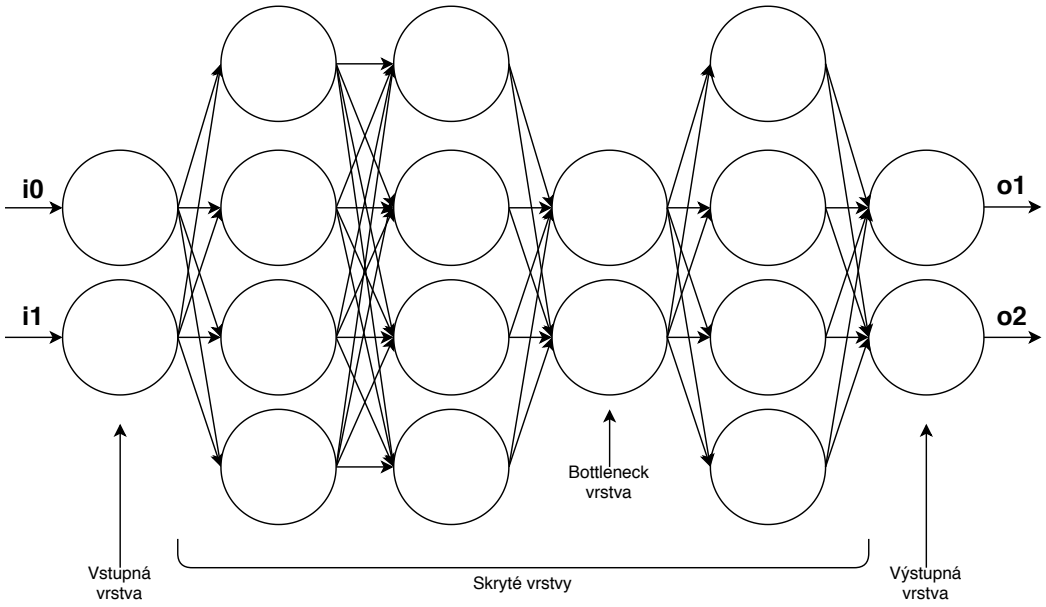
2.3.1 Neurónová sieť

Neurónové siete sú modely skladajúce sa z menších výpočtových jednotiek, ktoré sa nazývajú perceptrony. Perceptrony sú matematické modely predstavujúce neuróny. Perceptron je jednoduchý model, ktorý spracováva sériu vstupov a transformuje ich na výstupnú hodnotu. Množstvo vstupov perceptronov sa môže líšiť na základe typu a veľkosti siete.

Vrstvy neurónových sietí sa delia na tri typy. Prvý typ je vstupná vrstva, ktorá na vstupe prijíma dáta určené ku klasifikácií a propaguje ich do ďalších vrstiev. Druhý typ

je takzvaná skrytá vrstva. V sieti môže byť virtuálne neobmedzené množstvo skrytých vrstiev. Práve skryté vrstvy prevádzajú matematické operácie neurónovej siete. Posledný typ je vrstva výstupná. Výstupná vrstva udáva výsledky klasifikácie a počet výstupných uzlov, tiež určuje početnosť tried, ktoré je sieť schopná rozoznávať. V našom prípade sú vstupy siete hodnoty mel bank filtrov jednotlivých rámcov a výstupom sú pravdepodobnosti výskytu jednotlivých foném v tomto rámci.

Typ siete, ktorá je použitá v tejto práci, sa nazýva dopredná neurónová sieť. Je to najjednoduchší a zároveň najpoužívanejší typ neurónových sietí. Informácie sa v tomto druhu siete pohybujú iba jedným smerom a jej vrstvy sú plne oddelené. Typicky obsahujú najmenej jednu skrytú vrstvu a všetky uzly susedných vrstiev sú prepojené. Grafickú reprezentáciu jednoduchej doprednej neurónovej siete predstavuje obrázok 2.9.



Obr. 2.9: Grafická reprezentácia jednoduchej doprednej neurónovej siete s bottleneck vrstvou.

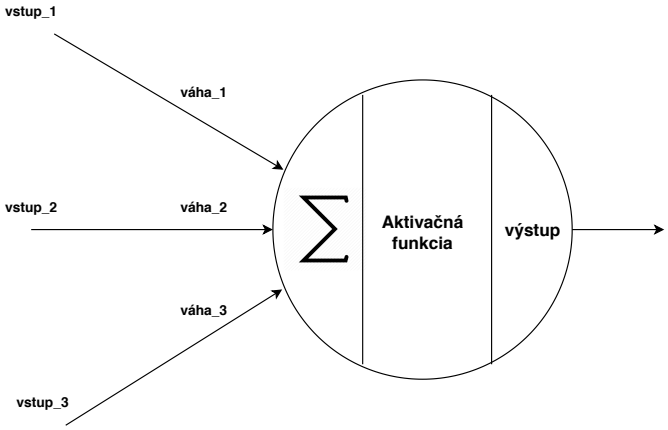
2.3.2 Perceptron- neurón

Perceptron je základná výpočtová jednotka siete. Vo vstupnej vrstve každý perceptron odpovedá jednej hodnote zo vstupných dát. V skrytej vrstve prijíma na vstup výstupnú hodnotu všetkých perceptronov, ktoré sú naň napojené z predošlej vrstvy. Tieto prijímané hodnoty sú vynásobené váhou spojenia, sčítané a v niektorých implementáciách sa naviac pričíta takzvaný bias. Táto hodnota je následne poslaná do aktivačnej funkcie perceptronu, ktorej výsledok reprezentuje výstupnú hodnotu tejto jednotky. Perceptron môžeme vyjadriť rovnicou 2.11 a je graficky znázornený na obrázku 2.10.

$$y_i = F(\sum_{i=1}^{N}(x_i w_i) + B_i) \tag{2.11}$$

Kde, y je výstupná hodnota , x je vstupná hodnota, w je váha spojenia, b je bias hodnota a F je aktivačná funkcia. V tejto implementácií je použitá sigmoidná aktivačná funkcia vyjadrená rovnicou 2.12.

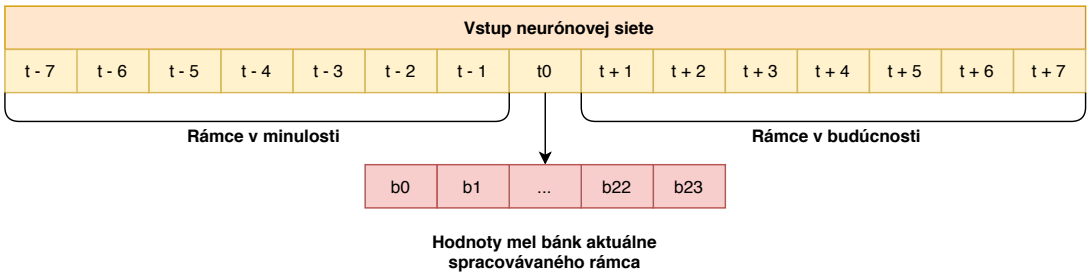
$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.12}$$



Obr. 2.10: Grafická reprezentácia perceptronu

2.3.3 Skladanie vstupu neurónovej siete

Vstupy neurónovej siete využitej v tejto implementácií, sú normalizované hodnoty mel bank filtrov jednotlivých audio rámcov. Vstup však neobsahuje iba hodnoty aktuálne spracovávaného rámca, ale aj hodnoty N rámcov pred a N rámcov po aktuálne spracovanom rámci. Dĺžka vstupného vektora môže byť teda vyjadrená ako $(2N+1)*M$ kde, N je rádius rámcov okolo aktuálne spracovaného rámca a M je počet mel bank filtrov.



Obr. 2.11: Grafická reprezentácia vstupu neurónovej siete. Vstup tvoria hodnoty mel bánk jednotlivých rámcov, ktoré sú získané pri spracovaní zvukového signálu. Vstup obsahuje aktuálne spracovávaný rámec a jeho okolie v podobe predchádzajúcich a nasledujúcich rámcov.

2.3.4 Interpretácia výstupu neurónovej siete

Výsledným výstupom neurónovej siete je séria reálnych čísiel. Tieto čísla sú transformované na pravdepodobnosti výskytu jednotlivých príznakov pomocou normalizovanej exponenciálnej funkcie tiež nazývanej softmax. Táto funkcia normalizuje hodnoty výstupu neurónovej

siete na rozloženie pravdepodobnosti. Výsledkom je séria reálnych čísiel v intervale $(0, 1)$, ktorých súčet je rovný 1, a tak môžu byť interpretované ako pravdepodobnosti. Softmax funkciu môžeme vyjadriť vzorcom 2.13

$$s_{x_i} = \frac{e^{x_i}}{\sum_{j=1}^{L} e^{x_j}} \tag{2.13}$$

Pričom $x \in N$ a $j \in [1, L]$ kde, L je veľkosť výstupnej vrstvy a x_i je hodnota výstupnej vrstvy na indexe i.

2.3.5 Zlepšenie rozpoznania jednotlivých foném

Hodnoty získané z neurónovej siete reprezentujú pravdepodobnosti výskytu foném v jednotlivých rámcoch. Pre zlepšenie odhadu pravdepodobností niektorých foném sa spolu s nimi zohľadňujú aj takzvané apriórne pravdepodobnosti. Tieto pravdepodobnosti sú získané z testovacej sady siete na základe výskytu jednotlivých foném. Výsledné pravdepodobnosti foném sú teda získané vzťahom:

$$P(z|p_i) = P(p_i)(P(z|p_i) \tag{2.14}$$

Kde, $(P(z|p_i)$ sú posteriórne pravdepodobnosti získané z výstupu neurónovej siete a $P(p_i)$ sú apriórne pravdepodobnosti. Dekodér pracuje s logaritmami fonémových pravdepodobností. Vzhľadom na apriórne pravdepodobnosti a to, že posteriórne pravdepodobnosti sú počítané funkciou softmax, výsledné logaritmy fonémových pravdepodobností môžu byť získané za pomoci rovnice:

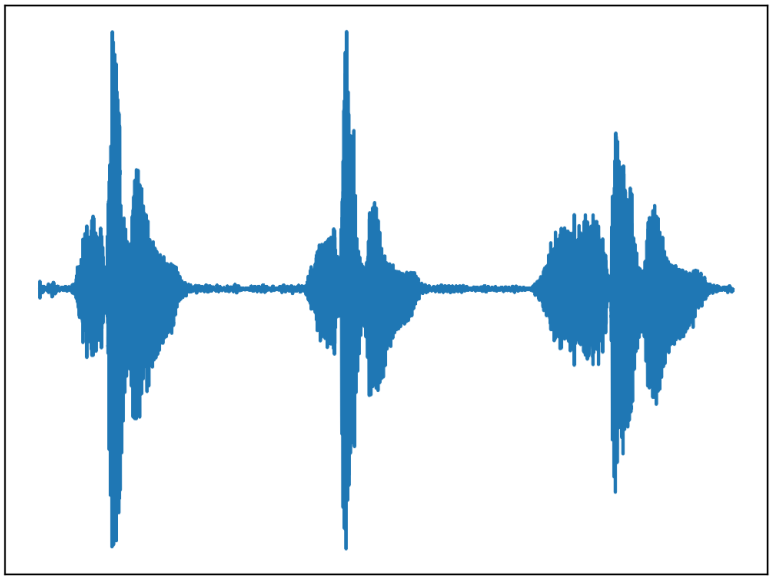
$$\ln P(z|p_i) = \ln P(p_i) + \ln \frac{e^{x_i}}{\sum_{j=1}^{L} e^{x_j}} = \ln P(p_i) + x_i - \ln \sum_{j=1}^{L} e^{x_j} \tag{2.15}$$

Výsledné fonémové pravdepodobnosti použité pre dekódovanie sú teda získané súčtom logaritmov apriórnych pravdepodobností a logaritmov výstupu neurónovej siete.

2.4 Dekodér

Úloha dekodéru pri problematike spracovania reči je nájdenie najpravdepodobnejšej reprezentácie slova zo získanej sekvencie príznakov. V tejto kapitole sa práca zaoberá problematikou a technikami dekodéru pracujúceho s posteriórnymi pravdepodobnosťami. Ďalej opisuje komponenty potrebné pre tvorbu jednoduchého dekodéru. Princípy popísane v tejto kapitole boli čerpané z [3] a [4].

Proces a techniky dekódovania sú nevyhnutnou časťou spracovania reči vzhľadom na obrovskú variáciu výslovnosti slov. Faktory ovplyvňujúce výslovnosť sú napríklad prízvuk alebo rýchlosť s akou je slovo vyslovené. Tento problém je graficky znázornený na obrázku 2.12.



Obr. 2.12: Slovo “seven” vyslovené trikrát po sebe rovnakým rečníkom.

2.4.1 Dekódovanie príznakov

Proces dekódovania môžeme považovať za hľadanie najpravdepodobnejšej sekvencie slov na základe získaných príznakov zvukovej stopy. V tomto prípade sú príznaky reprezentované posteriórnymi pravdepodobnosťami získanými zo zvukového signálu. Cieľom je teda nájdenie postupnosti slov $\overline{W}$, ktorá najviac odpovedá pozorovaným príznakom O. Aplikáciou Bayesovho pravidla získame rovnicu 2.16. Táto úprava umožňuje tento problém rozdeliť na niekoľko častí, ktoré sa dajú samostatne modelovať.

$$\overline{W} = \underset{W}{\operatorname{argmax}} \frac{P(O|W)P(W)}{P(O)} \tag{2.16}$$

Prvá časť $P(O|W)$ je nazývaná akustický model. Reprezentuje rozdelenie jednotlivých slov na fonémy. $P(W)$ sa nazýva jazykový model, ktorý popisuje pravdepodobnosť výskytu slova. Pravdepodobnosť $P(O)$ nemá žiadny vzťah s hľadaným slovom W ,takže nie je pri výpočtoch používaná. Zanedbaním $P(O)$ môžeme rovnicu hľadania postupnosti $\overline{W}$ zjednodušiť na nasledujúci tvar:

$$\overline{W} = \underset{W}{\operatorname{argmax}}\, P(W,O) = \underset{W}{\operatorname{argmax}}\, P(O|W)P(W) \tag{2.17}$$

Jazykový model je možné rozšíriť o takzvané n-gramové pravdepodobnosti, ktoré reprezentujú pravdepodobnosť výskytu sekvencie slov. Hodnoty používané na modelovanie spomínaných pravdepodobnostných modelov je nutné získať pred samotným procesom rozpoznávania. Na tieto účely sa využívajú rečové dáta v podobe zvukových nahrávok a textové dáta tvorené ich prepismi.

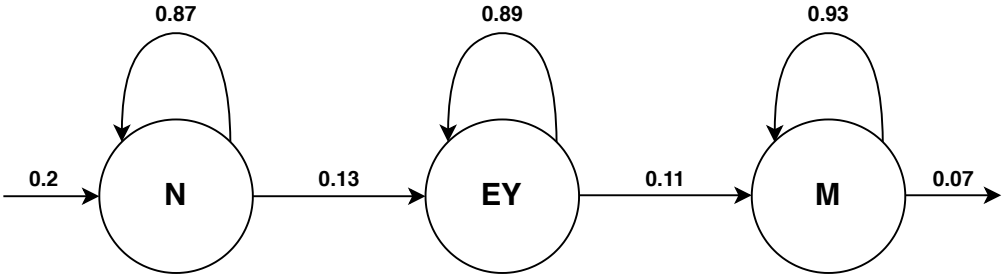
2.4.2 Akustický model

Akustický model mapuje sekvenciu akustických príznakov na rečové fonémy. Reprezentuje teda stavy jednotlivých rečových foném a pravdepodobnosti prechodov medzi týmito stavmi. Pre tento účel je predpokladané, že pri rozprave je ľudské rečové ústrojenstvo, pri rozdelení na dostatočne malé časové úseky, vždy v jednom z konečného počtu stavov, ktoré sú v tomto prípade tvorené fonémami. Akustický model je najčastejšie modelovaný za pomoci skrytých Markovových modelov (HMM).

Skryté Markovove modely

Skryté Markovove modely sú pravdepodobnostné modely popisujúce postupnosť stavov, ktoré sú tvorené uzlami a ich prechodmi. Jedná sa o model stochastického procesu, ktorý v diskrétnom čase generuje pozorovanie O. Prechody medzi jednotlivými uzlami vyjadrujú pravdepodobnosť s akou sa môže model dostať z určitého uzlu do iných uzlov. Každý uzol môže nadobúdať jeden z vopred definovaných stavov.

V tomto prípade stavy tvoria jednotlivé fonémy reči spolu s hodnotami pravdepodobností pre prechod do nasledujúceho stavu alebo cyklenie v aktuálnom stave. Akustický model je teda tvorený slovníkom, ktorý definuje slová známe dekodéru a ich rozloženie na fonémy.



Obr. 2.13: Akustická reprezentácia slova “name” za pomoci HMM. Foném v uzle určuje prijímanú pravdepodobnosť z aktuálneho pozorovania. Hodnoty nad prechodmi určujú pravdepodobnosť opustenia uzla.

2.4.3 Jazykový model

Jazykový model poskytuje dekodéru pravdepodobnosti výskytu slov a slovných sekvencií v reči. Jednotlivé záznamy sa nazývajú n-gramy, kde n určuje dĺžku sekvencie slov. Väčšina statických dekodérov využíva bi-gramové alebo tri-gramové modely, pretože väčšie modely výrazne zvyšujú veľkosť rozpoznávacej siete. V tejto práci bol použitý dynamický dekodér, ktorý je založený na jednoduchej uni-gramovej sieti a pravdepodobnosti n-gramov sú dynamicky vyhľadávané z ich úložiska počas rozpoznávania za pomoci histórie už rozpoznaných slov.

Uni-gramy definujú pravdepodobnosť výskytu slov v reči a dlhšie n-gramy definujú pravdepodobnosť výskytu slovných sekvencií. Ich úlohou je teda zlepšiť odhad najpravdepodobnejšej sekvencie slov $P(W)$. Túto pravdepodobnosť pre uni-gramový jazykový model je možné vyjadriť rovnicou:

$$P(W) = \Pi_{i=0}^{m} P(w_i) \tag{2.18}$$

Kde, $P(w_i)$ je uni-gramová pravdepodobnosť slova w_i a m je počet doposiaľ rozoznaných slov. Rovnicu možno rozšíriť o n-gramové pravdepodobnosti slovných sekvencií. Získanie najpravdepodobnejšej sekvencie slov za pomoci n-gramového jazykového modelu môže byť vyjadrené rovnicou:

$$P(W) = \Pi_{i=0}^{m} P(w_i | w_{i-(n-1)}, ..., w_{i-1}) \tag{2.19}$$

Keďže definovanie všetkých možných kombinácií n-gramov má za výsledok exponenciálny rast jazykového modelu, niektoré sekvencie používajú takzvanú backoff pravdepodobnosť. Týmto spôsobom je dosiahnuté toho, aby sa pre nedefinovanú sekvenciu nepripočítala nulová pravdepodobnosť výskytu. Backoff pravdepodobnosť taktiež adresuje problém, kedy neexistuje dostatočné množstvo dát pre tréning modelu, a tak môžu byť niektoré n-gramové sekvencie nedostupné. Na výpočet backoff pravdepodobností je použitá rovnica:

$$P(w_i | w_{i-n}) = P(w_(i) | w_{i-(n-2)}, ..., w_{i-1}) B(w_{i-(n-2)}) \tag{2.20}$$

Tabuľka 2.1: Príklad tri-gramového jazykového modelu pre sekvenciu slov. Pri rozoznaní prvého slova sekvencie, v tomto prípade 1, je k aktuálnej predikcií pripočítané skóre uni-gramového záznamu. Ak, je pri rozpoznávaní získaná sekvencia slov [1, 2], tak je v n-gramovom úložisku vyhľadaný jej bi-gramový záznam a pripočítané jemu priliehajúce skóre. V prípade, že by záznam pre sekvenciu [1, 2] neexistoval, je použité backoff skóre uni-gramového záznamu. Po nasledovnom rozpoznaní slova 3 je najskôr vyhľadané skóre tri-gramového záznamu sekvencie [1, 2, 3]. Ak, by záznam neexistoval, bolo by použité backoff skóre sekvencie [1, 2].

Typ záznamu				
uni-gram	slovo 1	skóre	backoff	
bi-gram	slovo 1	slovo 2	skóre	backoff
tri-gram	slovo 1	slovo 2	slovo 3	skóre

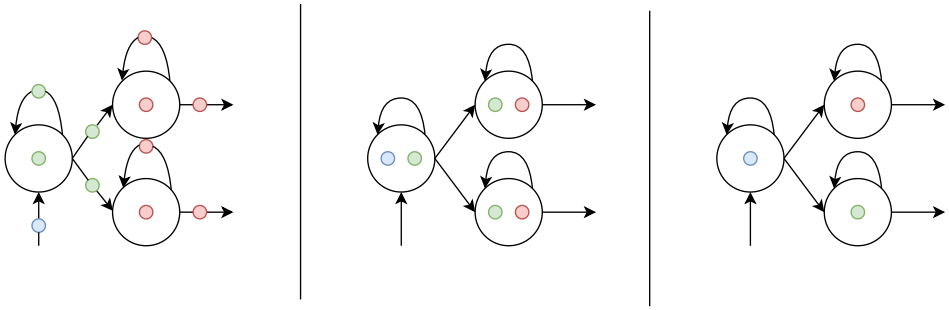
2.4.4 Token passing

Jednotlivé uzly akustického modelu vytvárajú grafovú štruktúru nazývanú rozpoznávacia sieť. Úlohou dekodéru je nájsť najpravdepodobnejšiu cestu v tejto sieti. Keďže nie je jasné, ktorý foném bude rozoznaný ako ďalší, je nutné rátať s viacerými cestami, ktorých skóre môže byť na konci rozpoznávania najlepšie. Algoritmus použitý v tejto práci, na adresovanie tohto problému, sa nazýva token passing.

Token je reprezentovaný ako objekt, ktorý si ukladá informáciu o svojej doterajšej ceste. Na začiatku rozpoznávania je do počiatočného stavu umiestnený prázdny token. Pri každom ďalšom kroku je token kopírovaný do nasledujúcich prepojených stavov. Pri kopírovaní tokenu sú zohľadnené skóre prechodov medzi uzlami, ktoré v našom prípade reprezentujú jednotlivé fonémy.

Viterbiho kritérium

Pri kopírovaní tokenov v každom kroku rozpoznávania sa ich počet v jednotlivých uzloch exponenciálne zvyšuje. Viterbiho kritérium [8] hovorí, že pokiaľ sa v jednom uzle stretne viac tokenov, ponechaný je len token s najlepším skóre. Vďaka tomuto kritériu je počet tokenov v rozpoznávacej sieti minimalizovaný na počet jej uzlov.



Obr. 2.14: Grafické znázornenie Viterbiho kritéria pri token passingu

Optimalizačné metódy

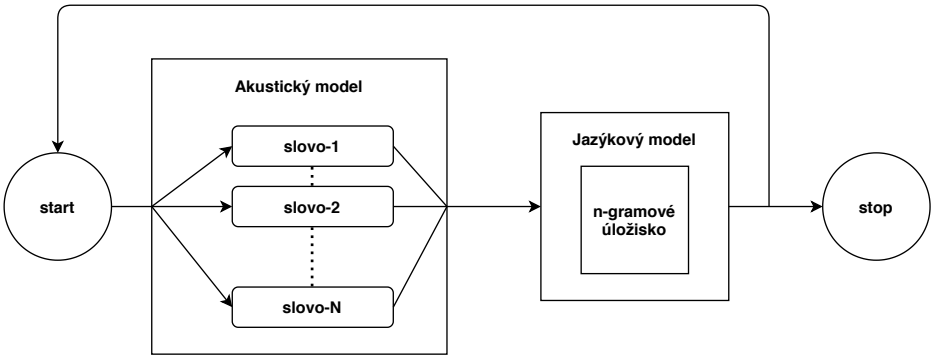
Ak dekodér pracuje so slovníkom, ktorý obsahuje veľký počet záznamov, ani po aplikácií Viterbiho kritéria nemusí byť počet tokenov dostatočne malý pre spracovanie v reálnom čase. Je teda nutné zakomponovať metódy, ktorými je tento počet možné redukovať. V tejto práci sú použité metódy ponechania N najlepších stavov [7] a beam pruning [7].

Metóda ponechania N najlepších stavov zohľadňuje skóre tokenov a v rozpoznávacej sieti ponecháva len N tokenov s najlepším skóre pričom ostatné sú zahodené. Pri beam pruningu je najskôr určené priepustné pásmo na základe najlepšieho a najhoršieho uchovávaného skóre a všetky tokeny, ktorých skóre do tohto pásma nespadá sú zahodené.

2.4.5 Rozpoznávanie spojitej reči

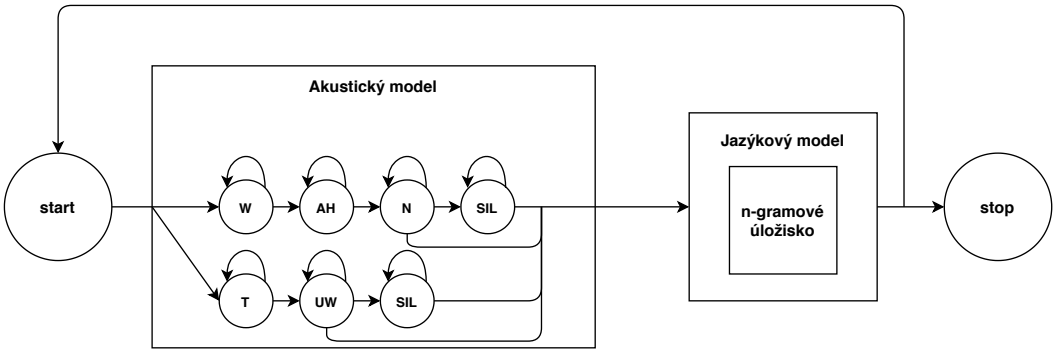
Na rozdiel od rozpoznávania jednotlivých slov, rozpoznávanie spojitej reči musí riešiť problém prechodu medzi jednotlivými slovami. Na začiatku rozpoznávania je do počiatočného uzla rozpoznávacej siete vložený prázdny token. Počiatočný uzol pracuje s pravdepodobnosťou ticha. Token je následne z počiatočného stavu expandovaný do akustického modelu tvoreného uni-gramovou rozpoznávacou sieťou. V stave, kedy token opúšťa rozpoznávaciu sieť, je jeho skóre aktualizované za pomoci n-gramového jazykového modelu. Keďže v jednom kroku rozpoznávania môže sieť opustiť počet tokenov, rovný počtu slov v slovníku, je z nich vybraný token s najlepším skóre. Vybraný token je expandovaný na začiatok rozpoznávacej siete a taktiež do jej koncového uzla. Rozpoznávanie vety je dokončené v momente, kedy dekodér dostane signál pre ukončenie rozpoznávania. Výsledok rozpoznávania je dostupný v podobe tokenu s históriou slov v konečnom uzle. Vďaka priebežnému expandovaniu do koncového uzla siete je počas rozpoznávania dostupný aj priebežný prepis reči.

Pri rozpoznávaní spojitej reči môže byť slovo nasledované ďalším slovom alebo krátkym tichom. Ticho môže reprezentovať významovú pauzu vo vete, avšak môže byť spôsobené aj pomalým tempom rozpravy rečníka. Najjednoduchším spôsobom, ako adresovať tento problém, je druhá varianta totožného slova, ktorej je na konci pridaný uzol reprezentujúci ticho. Avšak, tento postup by znamenal dvojnásobné množstvo slov a tým pádom aj



Obr. 2.15: Rozpoznávanie spojitej reči za pomoci uni-gramovej rozpoznávacej siete a dynamického vyhľadávania n-gramových pravdepodobností.

dvojnásobné množstvo tokenov na spracovanie. Optimálnejším spôsobom je pridanie uzla reprezentujúceho ticho za každé slovo. Za stav, pri ktorom token opúšťa model, je následne považovaný konečný stav ticha, ale aj stav pred ním. Pridané uzly tak reprezentujú ticho, ktoré sa môže vyskytovať medzi slovami pri rozpoznávaní spojitej reči.



Obr. 2.16: Rozpoznávanie spojitej reči. Akustický model obohatený o možný stav ticha medzi jednotlivými slovami.

Kapitola 3

Implementácia

Táto kapitola popisuje implementáciu rozpoznávača reči vo webovom prehliadači. Čitateľ sa v nej oboznámi s technológiami využitými v aplikácií, využitými knižnicami a funkcionalitou jednotlivých častí implementácie, ktoré sú nutné pre chod tejto aplikácie. Objasňuje spôsob implementácie jednotlivých častí rozpoznávača. Pre implementáciu bol využitý jazyk Typescript a pre akceleráciu maticových operácií knižnica Tensorflow.

3.1 Ciele implementácie

Hlavným cieľom pre vytvorenie tejto aplikácie je poskytnutie nástroja na rozpoznávanie reči vo webovom prehliadači, a tým sprístupniť túto funkcionalitu bez potreby odosielania spracovávaných dát na vzdialené servery so špecializovanou výpočtovou technikou. Vytvorená knižnica pracuje s najmodernejšími technológiami pre tvorbu webových aplikácií a využíva dostupné prostriedky pre grafickú akceleráciu výpočtov pre dosiahnutie rozpoznávania reči v reálnom čase. Vytvorené grafické, užívateľské prostredie aplikácie je oddelené od funkcionality knižnice, a tak je knižnica pripravená pre jednoduché znovupoužitie.

3.2 Použité technológie

Webové aplikácie, ktorých program prebieha na strane klienta bez komunikácie so vzdialeným serverom, sú väčšinou implementované v jazyku Javascript. Existujú aj možnosti implementácie v iných jazykoch, no tie sú na koniec aj tak prekladané do Javascriptu alebo spojené s Javascriptom cez rôzne API.

Web API

Rozhranie webových aplikácií, ponúkané webovými prehliadačmi, je využívané na rôzne úlohy, ako manipulovanie webových elementov alebo prehrávanie a nahrávanie zvuku. Funkcie poskytované týmto rozhraním sú špecifikované distribúciou webového prehliadača a taktiež jeho verziou. Implementácia tohto projektu využíva webové API na vykonávanie viacerých úloh, no niektoré špecifické problémy majú vlastnú implementáciu z dôvodu kompatibility.

Node.js

Node.js[1] je najpoužívanejším prostredím pre interpretáciu a vykonávanie Javascriptu. Prostredie je pripravené pre implementáciu širokého spektra webových aplikácií. Ponúka ľahkú konfiguráciu a množstvo open-sourcových balíčkov uľahčujúcich prácu s webovým prehliadačom.

WebStorm

Webstorm je najpopulárnejšie vývojové prostredie (ďalej len IDE), podporujúce všetky jazyky používané pre vývoj webových aplikácií. Podporuje všetky najpopulárnejšie operačné systémy. Taktiež ponúka prostriedky na interaktívnu tvorbu webového grafického prostredia.

Typescript

Typescript[2] je voľne dostupný programovací jazyk vytvorený a udržiavaný Microsoftom. Je určený na vývoj webových aplikácií na strane klienta, ale aj na strane serveru. Je to syntakticky striktná nadstavba Javascriptu, ktorá mu dodáva statické typovanie.

Typescript je navrhnutý na vývoj veľkých aplikácií, ktoré sú následne prekladané do Javascriptu. Keďže je typescript nadstavba Javascriptu, existujúce Javascriptové programy a knižnice sú plne využiteľné aj v Typescripte.

Tensorflow

Tensorflow[3] je voľne dostupný framework vyvíjaný spoločnosťou Google. Jej hlavné zameranie je strojové učenie a iné štatistické úlohy. Tensorflow ukladá vektory a matice do špeciálnych objektov zvaných tensory, nad ktorými ponúka akceleráciu operácií za pomoci dostupných grafických prostriedkov.

V tejto implementácií používame verziu Tensorflow.js, ktorá je uspôsobená pre webové prehliadače a podporuje taktiež vybrané mobilné webové prehliadače. Pri načítaní si framework podľa konfigurácie vytvorí špecifické prostredie pre svoj beh. Podporovaných je viacero prostredí, ktoré implementujú vlastný spôsob ukladania tensorov a vykonávania matematických operácií.

Graficky akcelerované prostredie - WebGL

Toto prostredie je v súčasnej dobe pre webové prehliadače najsilnejšie v rámci výpočtovej sily. Toho je dosiahnuté vďaka využitiu grafickej akcelerácie vo webovom prehliadači. Tensory sú ukladané ako grafické textúry a operácie nad nimi sú prevádzané pomocou grafických shaderov, čo zaručuje až výrazné zrýchlenie oproti prevedeniu týchto operácií za pomoci procesoru. Graficky akcelerované prostredie avšak vyžaduje manuálne spravovanie pamäte, ktorá obsahuje tensory.

[1] https://nodejs.org/
[2] https://www.typescriptlang.org/
[3] https://www.tensorflow.org/

CPU prostredie

Toto prostredie je vykonávané v čistom Javascripte za pomoci procesoru, a tak je aj jeho výkon v porovnaní s grafickým prostredím nižší. Avšak, väčšina moderných webových prehliadačov, vrátane mobilných verzií, podporuje grafickú akceleráciu.

Vue.js

Vue.js[4] je knižnica používaná pre tvorbu webového grafického užívateľského rozhrania (GUI) a jednostránkových webových aplikácií. Je plne kompatibilná s Typescriptom a ponúka rýchle a jednoduché nastavenie. Taktiež ponúka mnoho open-sourcových rozšírení. V tejto práci bolo použité rozšírenie Onsen-Vue[5], ktoré uľahčuje rozmiestnenie komponentov GUI a zaručuje ich responzívnosť pri zmene rozlíšenia.

Git

Git je v dnešnej dobe najviac používaný verzovací systém na svete. Je určený predovšetkým pre veľké tímy, ktoré sa podieľajú na spoločnom projekte. Napriek tomu je Git veľmi užitočným nástrojom pre ľubovolný projekt a možno ho použiť na zálohovanie, rekapituláciu zmien a prenos súborov medzi zariadeniami. Ako hosťovací server bol použitý GitHub[6] hlavne kvôli skúsenostiam z predošlých školských, ale aj osobných projektov.

3.3 Štruktúra knižnice

Program je vytvorený ako knižnica pre použitie vo webovom prehliadači. Pri jej tvorbe bolo dbané na to, aby bola využiteľná aj v iných webových aplikáciach a zakomponovaná do iných grafických užívateľských prostredí.

Knižnica je rozdelená do niekoľkých logických celkov, z ktorých každý zaobstaráva jednu z úloh potrebných pre prevod reči na text.

Recording

Obsahuje triedy využité pri nahrávaní zvukovej stopy. Taktiež obsahuje dátové štruktúry používané na spracovanie a prenos audio dát.

Preprocessing

Táto zložka obsahuje všetky triedy, ktoré sa starajú o extrakciu príznakov zo zvukového signálu a dátové štruktúry, v ktorých sú príznaky skladované a prenášané.

Classification

Obsahuje triedy starajúce sa o predikciu pravdepodobností jednotlivých foném v rámcoch. Ďalej, triedy, ktoré sa starajú o pripravovanie vstupu neurónovej siete z prijatých mel bank filtrov a ich normalizáciu.

[4] https://vuejs.org/
[5] https://onsen.io/
[6] https://github.com/

Decoding

Zložka decoding obsahuje zložky s implementáciou dynamického dekodéru. Hlavnou triedou je súbor Decoder.ts, ktorá zaobaľuje funkcionalitu všetkých ostatných tried.

Workers

Zahrňuje súbory, ktoré sa starajú o vytváranie Javascriptových workerov (threadov) a funkcionalitu pre komunikáciu medzi nimi.

3.3.1 Viacvláknové spracovanie

Architektúra knižnice je rozdelená do viacerých modulov, ktoré vykonávajú operácie súčasne na vlastných vláknach, čo zaručuje súbežné spracovanie hlavných častí rozpoznávača. Vláknové spracovanie je implementované pomocou Javascriptových workerov (vlákien). Javascriptové vlákna obsahujú len veľmi primitívnu funkcionalitu zdieľanej pamäte, ktorá je pre implementáciu zložitej funkcionality, ako je rozpoznávač reči, nepoužiteľná. Všetky dáta sú tak prenášané medzi vláknami za pomoci natívnych správ, ktoré vyžadujú kopírovanie pamäte zdielaného objektu. Tento fakt má za následok zhoršenie rýchlosti rozpoznávania. Aj napriek tomuto značnému obmedzeniu sa v tejto implementácií podarilo dosiahnuť rozpoznávania v reálnom čase. Obrázok, 3.1 znázorňuje vzťah a komunikáciu jednotlivých vlákien.

Po importovaní knižnice do webovej aplikácie sú dostupné metódy, ktoré spustia a ukončia nahrávanie audio signálu. Trieda Controller taktiež obsahuje verejnú premennú typu string, do ktorej bude dekodér odosielať výstup. Na túto premennú bude možné pripojiť element grafického užívateľského prostredia pre zobrazovanie výsledného textu.

3.4 Nahrávanie zvukového signálu

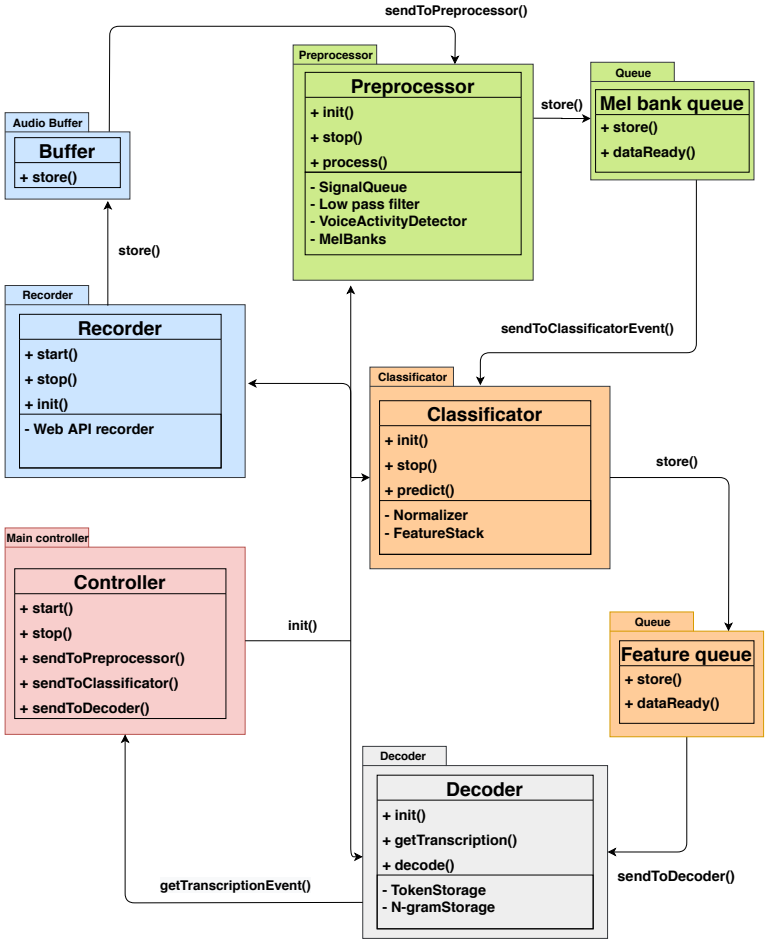
Nahrávanie zvukového signálu je zabezpečené pomocou modulu Recorder. Je počiatočným bodom rozpoznávača a zároveň zabezpečuje inicializáciu a komunikáciu jednotlivých vlákien. Implementuje funkcionalitu pre získanie a prípravu dát pre vlákno extrakcie príznakov.

3.4.1 Rekordér

Modul pre získavanie zvukového signálu je implementovaný za pomoci web API poskytovaného prehliadačom. Audio je ukladané do bufferu vo vzorkovacej frekvencií, ktorá je natívna pre nahrávacie zariadenie a bitovej hĺbke 16 bitov. V momente, kedy je buffer naplnený, je vyvolaná udalosť, ktorá odosiela získané dáta do vlákna pre extrakciu príznakov. Pred predaním dát na ďalšie spracovanie je signál prevzorkovaný na požadovanú frekvenciu 8000 Hz. API v niektorých prípadoch poskytuje interný nástroj na prevzorkovanie signálu. Ak tento nástroj nie je dostupný, modul používa vlastný nástroj vytvorený pre tento účel.

Spracovanie audio súboru

Pre spracovanie audio nahrávky zo súboru je nutná kontrola jej formátu. Táto implementácia podporuje iba súbory vo formáte WAV. Súbory musia spĺňovať nasledujúcu špecifikáciu: audio obsahuje iba jeden kanál, dáta sú uložené vo formáte PCM, bitová hĺbka je 16 bitov a vzorkovacia frekvencie je 8000 Hz. Nahrávka je následne odoslaná do vlákna pre spracovanie príznakov.



Obr. 3.1: Architektúra knižnice. Jednotlivé vlákna sú označené vlastnou farbou. Moduly, ktoré obsahujú vlákna sú zobrazené na diagrame spolu s ich základnou komunikáciou.

3.5 Extrakcia príznakov

Extrakcia príznakov prebieha súbežne s nahrávaním zvukových dát. Toho je dosiahnuté vďaka implementácií extrakcie na vlastnom vlákne.

Rámcovanie

Dáta sú do vlákna extrakcie príznakov prenášané za pomoci triedy AudioData. Po obdržaní sú dáta rozdelené do audio rámcov s dĺžkou 25 milisekúnd a prekryvom 15 milisekúnd. Pre účely udržania dát jednotlivých rámcov a následná práca s nimi bola vytvorená trieda AudioFrame.

Hammingovo okno

Nasledujúcim krokom je aplikácia Hammingovho okna (2.2.2). Koeficienty Hammingovho okna sú počítané pri inicializácií rozpoznávača, a tak je nutné ich počítať iba raz. Framework Tensorflow taktiež ponúka funkcionalitu aplikácie okennej funkcie, no nakoľko ide o jednoduchú operáciu násobenia, rozdiel vo výkone s implementáciou rozpoznávača je minimálny.

Rýchla Fourierova transformácia

Aplikácia rýchlej Fourierovej transformácie je opäť zaistená pomocou frameworku Tensorflow. Je tak učinené z dôvodu optimalizácie. Knižnica aj napriek tomu obsahuje vlastnú implementáciu v triede FFT a tiež podporu komplexných čísiel v triede ComplexNumber. Vstupné dáta Fourierovej transformácie majú dĺžku 256 vzoriek. Dáta sú zložené zo vzoriek audio rámca (200 vzoriek) a doplnené o 56 nulových hodnôt.

Detekcia rečovej aktivity

Dôvody, pre detekciu rečovej aktivity, sú dva. Prvým dôvodom je lepšia adaptácia rozpoznávača na rečníka, obmedzením rámcov obsahujúcich šum. Druhým je deaktivácia neurónovej siete a dekodéru v momente, kedy nie je detekovaná reč. Vďaka tomu bolo možné výrazne znížiť časovú náročnosť spracovania.

Detekcia je implementovaná pomerne jednoduchým spôsobom. Silové spektrum reči, ktoré je výsledkom Fourierovej transformácie, je sčítané a následne vydelené vzorkovacou frekvenciou audio signálu. Ďalej je nutné určiť hranicu, kedy je rámec považovaný za aktívny. Pri testovaní bolo zistené, že kvalita vstupného zariadenia môže v signále generovať šum, ktorý znemožňuje nastavenie fixnej hranice. Rozpoznávač teda ponúka funkcionalitu na úpravu hranice detekovania rečovej aktivity.

$$f(s) = \begin{cases} \text{reč} & \text{ak} \quad \sum_{i=0}^{N} s_i > \text{treshold} \\ \text{ticho} & \text{inak} \end{cases} \tag{3.1}$$

Pri určovaní aktivity, iba pre jednotlivé rámce, by veľmi často dochádzalo k označeniu veľmi malých časových úsekov za aktívne. Experimentálne boli teda určené hranice, minimálne časové limity, po ktorých detektor zmení svoj stav. Kvôli tejto úprave je v detektore vytvorený zásobník predošlých rámcov, vďaka ktorému nedochádza ku strate dát pri oneskorených prechodoch zo stavu ticha na reč.

Aplikácia Mel bank filtrov

Druhým výpočtovo náročnejším krokom extrakcie príznakov je aplikácia Mel bank filtrov. K jej akcelerácií je využitý framework Tensorflow - podľa prostredia prehliadača. Parametre filtrov sú rovnako, ako pri Hammingovom okne, počítané pri inicializácií rozpoznávača a tento výpočet nie je nutné opakovať. Postup výpočtu jedného filtra je znázornený nasledujúcim algoritmom 1.

Algorithm 1 Algoritmus aplikácie mel bank filtru

```
FFT = FFT vstup
idx = index mel bank filtrov
for i = 0,1, ..., dĺžka(FFT) do
    res = res + mels[idx][i] * (FFT[i].real + FFT[i].imag)
end for
if res == 0 then
    res = 0
else
    res = log(res)
end if
return res
```

3.5.1 Uniformná normalizácia

Uniformná normalizácia je implementovaná pomocou triedy Normalizer. Táto trieda nepočíta priemer zo všetkých predošlých rámcov, no využíva takzvaný pohyblivý priemer, kedy vzorky staršie, ako stanovená hranica, nie sú relevantné.

Táto implementácia používa optimalizovanú uniformnú normalizáciu, vďaka ktorej nie je nutné ukladať do pamäte hodnoty všetkých predošlých rámcov. Normalizácia je prevádzaná na základe vážených priemerov uložených v dvoch vektoroch a jednej premennej predstavujúcej váhu.

Algorithm 2 Algoritmus aplikácie exponenciálnej normalizácie

```
banks = mel banky aktuálneho rámca
position = position + 1
for i = 0,1, ..., dĺžka(banks) do
    current[i] = current[i] + banks[i]
    banks[i] = banks[i] - (current[i] + (N - position) * previous[i]) / N
    if position >= N then
        previos = current / N
        current = [0,0...0]
        position = 0
    end if
end for
return banks
```

3.5.2 Exponenciálna normalizácia

Druhou implementovanou metódou je exponenciálna normalizácia. Jej výhodou je, že staršie vzorky ovplyvňujú aktuálny rámec s menšou váhou. Implementácia tejto normalizácie je pomerne jednoduchá. Za pomoci rovnice 2.10 je nutné získanie potrebných parametrov. Taktiež je nutné udržiavať vektor hodnôt z predošlej iterácie.

Algorithm 3 Algoritmus aplikácie exponenciálnej normalizácie

```
banks = mel banky aktuálneho rámca
for i = 0,1, ..., dĺžka(banks) do
    prev[i] = prev[i] * multiplier + banks[i] / divider
    banks[i] = banks[i] - prev[i]
end for
return banks
```

3.6 Klasifikátor foném

Klasifikácia získaných audio príznakov na fonémy je zaistená neurónovou sieťou. Práca sa nezaoberá jej tréningom, nakoľko vopred vytrénovaná sieť bola dodaná vedúcim práce. Práca používa neurónové siete, ktoré rozpoznávajú jednostavové alebo trojstavové fonémy. Trojstavové fonémy rozdeľujú jednotlivé fonémy na tri stavy, ktoré by mali zaručiť väčšiu presnosť klasifikácie. Tento prístup má však za následok trojnásobné množstvo uzlov rozpoznávacej siete popísanej v podkapitole 2.4.2. Štruktúry použitých sietí sú zobrazené v tabuľke 3.1 a 3.2.

Aktivačná funkcia oboch sietí je sigmoida, ktorá je popísaná v podkapitole 2.3. Ako možno vidieť, sieť, ktorá rozpoznáva jednostavové fonémy, obsahuje takzvanú bottleneck vrstvu, ktorá nevyužíva žiadnu aktivačnú funkciu. Na konečnú vrstvu oboch sietí je aplikovaná funkcia softmax, ktorá je taktiež popísaná v podkapitole 2.3.

Tabuľka 3.1: Štruktúra použitej neurónovej siete pre jednostavové fonémy

Vrstva	**Rozmer**	**Aktivačná funkcia**
Vstupná	360	
1	500	sigmoid
2	500	sigmoid
3	80	
4	500	sigmoid
5	46	softmax

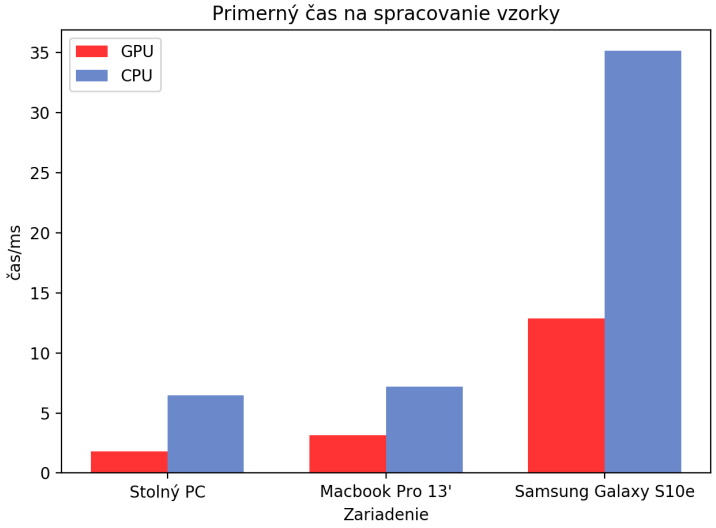
Tabuľka 3.2: Štruktúra použitej neurónovej siete pre trojstavové fonémy

Vrstva	**Rozmer**	**Aktivačná funkcia**
Vstupná	360	
1	700	sigmoid
2	700	sigmoid
3	700	sigmoid
4	700	sigmoid
5	152	softmax

Keďže dekodér, ktorý tvorí finálnu časť rozpoznávača, pracuje s logaritmami pravdepodobností, hodnoty získané po výpočte siete sú zlogaritmované.

Vstupom do neurónovej siete sú normalizované výsledky mel bank filtrov. O ukladanie predspracovaných dát a ich prípravu pre neurónovú sieť sa stará trieda FeatureStack. Keďže neurónová sieť neprijíma na svoj vstup len hodnoty aktuálneho rámca, ale aj okruh rámcov s rádiusom o veľkosti sedem rámcov. Dáta je naviac nutné preorganizovať, pretože sieť predpokladá, že kanály jednotlivých mel bánk budú nasledovať za sebou.

Pre akceleráciu výpočtov neurónovej siete je využitý framework Tensorflow. Za použitia jeho grafickej akcelerácie bolo dosiahnuté až 4-násobné zrýchlenie oproti výpočtu za pomoci procesoru. Možnosť využiť grafickú akceleráciu vo webovom prehliadači sa však líši na základe distribúcie a verzie webového prehliadača. Výsledná rýchlosť akcelerácie taktiež záleží od výpočtového výkonu zariadenia, na ktorom je prehliadač spustený.



Obr. 3.2: Graf znázorňuje priemerný čas na spracovanie jedného rámca neurónovou sieťou za pomoci frameworku Tensroflow. Test bol vykonaný na stolnom počítači s dedikovanou grafickou kartou, prenosnom laptope s integrovanou grafickou kartou a mobilnom zariadení. Neurónová sieť použitá pre test obsahovala v skrytej vrstve 1200 perceptronov.

3.7 Dekodér

Dekodér je najzložitejšia a najviac výpočtovo náročná časť rozpoznávača. Z tohto dôvodu je nutné dôkladne navrhnúť dátové štruktúry a ich metódy, ktoré s nimi pracujú. Ďalej je nutné navrhnúť efektívny spôsob implementácie algoritmu token passing popísaného v podkapitole 2.4.4. Ako posledná je nutná implementácia metód na optimalizáciu rozpoznávača, ako aj spôsob ukladania rozoznaných slov.

3.7.1 Uni-gramová rozpoznávacia sieť

Uni-gramová rozpoznávacia sieť je výpočtovo najnáročnejší komponent dekodéru. Algoritmus token passingu je priamo ovplyvnený jej implementáciou, keďže pracuje nad štruktúrou reprezentujúcou túto rozpoznávaciu sieť. Počas vývoja sa experimentovalo s viacerými druhmi implementácie rozpoznávacej siete, vrátane maticovej implementácie a implementácie za pomoci natívnych Javascriptových polí. Tieto možnosti sa ukázali ako výpočtovo neefektívne. Optimálnou metódou sa ukázal spôsob, s využitím jednosmerne viazaného zoznamu, použitý aj v práci [10]. Keďže Javascript neposkytuje možnosť výberu predávania pamäte referenciou alebo hodnotou, premenné, ktoré je nutné predávať referenciou na miesto uloženia v pamäti, bolo nutné zaobaliť do tried a pracovať s ich inštanciami. Uzol jednosmerne viazaného listu je reprezentovaný triedou:

```
class StorageUnit {
    public token: Token;
    public position: number;
    public prevRow: StorageUnit | null;
};
```

Pre každé slovo v slovníku je teda vytvorený vlastný jednosmerne viazaný zoznam, ktorého uzly reprezentujú rozloženie slova na fonémy. Jednotlivé zoznamy sú prístupné cez referenciu na ich prvý uzol, ktorá je uložená v Javascriptovom dynamickom poli:

```
class TokenStorage {
  public data: Array<StorageUnit | null>;
 }
```

Porovnanie časovej náročnosti implementácie za pomoci natívnych polí a jednosmerne viazaných zoznamov je zobrazené v tabuľke 3.3. Rozdieli pri testoch sa líšili na základe veľkosti slovníku a počtu udržiavaných tokenov, no implementácia za pomoci jednosmerne viazaných zoznamov sa ukázala ako výhodnejšia.

Tabuľka 3.3: Čas potrebný na spracovanie 90 sekúnd dlhej nahrávky. Slovník pri meraní času obsahoval 10000 slov

Implementácia	**Počet udržiavaných tokenov**	**čas/s**
Viazané zoznamy	4000	63.87
	2000	48.54
Natívne polia	4000	81.36
	2000	56.21

3.7.2 Token

Token je najpodstatnejšia štruktúra rozpoznávača. V každom kroku rozpoznávania je ich vytváraných a spracovávaných až niekoľko tisíc. Mala by teda obsahovať len nevyhnutné informácie a ich veľkosť by mala byť obmedzená na minimum. Každý token teda nesie informáciu svojho skóre a taktiež informáciu o svojej doterajšej ceste. Štruktúra tokenu v tejto impelementácií vyzerá nasledovne:

```
export class Token {
  public score: number = 0;
  public path: Link | null = null;
}
```

3.7.3 História tokenu

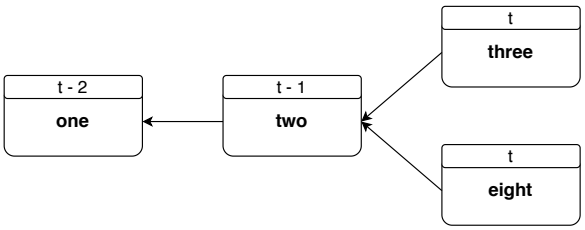
Pri rozpoznávaní spojitej reči je finálny prepis tvorený históriou cesty tokenu s najlepším skóre. Z tohto dôvodu je potrebné vytvoriť efektívny spôsob ukladania tejto informácie. História je reprezentovaná identifikátorom slova, ktorý je do tokenu vložený po opustení rozpoznávacej siete. Pri počiatočnej implementácií boli identifikátory slov ukladané do natívneho dynamického pola. Táto varianta však nebola pamäťovo ani výpočtovo optimálna. V neskorších fázach rozpoznávania boli jej výsledkom veľké pamäťové a aj výpočtové nároky pri kopírovaní tokenov.

Z tohto dôvodu bolo ukladanie histórie tokenu implementované ako jednosmerne viazaný zoznam. Vďaka jednosmernému viazanému zoznamu sa pri kopírovaní ukladá iba prvý

uzol zoznamu, čo výrazne znižuje nároky na pamäť a čas potrebný na kopírovanie tokenu. Štruktúra uzla tohto zoznamu vyzerá nasledovne:

```
export class WordLink {
  public wordId: number;
  public previous: WordLink | null = null;
  public sectionSplit: boolean = false;
 }
```

Záznamy histórie môžu byť zdieľané v histórií viacerých tokenov a vďaka tomu vytvárajú stromovú štruktúru, ktorej koreňovým uzlom je prvé rozpoznané slovo.



Obr. 3.3: Grafické znázornenie stromovej štruktúry histórie rozpoznaných slov

3.7.4 Token passing

Jeden z najpoužívanejších algoritmov pre zistenie najpravdepodobnejšej cesty cez HMM je token passing. Je ideálny pre použitie pri rozpoznávaní spojitej reči, ale aj pri rozpoznávaní jednotlivých slov. Algoritmus je popísaný v algoritme 5. Algoritmus bol prevzatý z [9] a následne upravený pre použitie s vlastnou dátovou štruktúrou.

Algorithm 4 Token passing pre rozpoznávaciu sieť tvorenú jednosmerne viazanými zoznamami

```
1:  toEmit = Array()
2:  for i = 0,1, ..., dĺžka(TokenStorageRows) do
3:      node = TokenStorageRows[i]
4:      prev = null
5:      while node != null do
6:          if node.position === EMIT then
7:              emited = updateScoreAndEmit(node)
8:              toEmit.push(emited)
9:          else
10:             updateScoreAndExpand(node)
11:         end if
12:     end while
13: end for
14: best = getBest(toEmit)
```

Algortimus iteruje nad všetkými cestami rozpoznávacej siete. Každá cesta je spracovaná prechodom cez jednosmerne viazaný zoznam. Na základe pozície uzla vo viazanom zozname sa určuje operácia nad tokenom, ktorý obsahuje. Ak, je pozícia rovná počtu fonémov slova,

ktoré cesta siete predstavuje, pravdepodobnosť tokenu je aktualizovaná a token opúšťa sieť. Naopak, ak, je pozícia menšia, token je aktualizovaný a jeho kópie sú odoslané do všetkých prepojených uzlov. Tokeny, ktoré opustili sieť, sú zahodené až na token s najlepším skóre, ktorý predstavuje najpravdepodobnejšiu cestu, a teda vo svojej histórií nesie aktuálny prepis reči na text.

Viterbiho kritérium

Dôležitou častou token passingu je aj Viterbiho kritérium, ktoré je aplikované pri každom kopírovaní tokenu do prepojeného uzla viazaného zoznamu. Ak uzol, do ktorého sa snažíme vložiť kopírovaný token už obsahuje iný token, ich skóre sú porovnané a token s horším skóre je zahodený.

3.7.5 N-gramové úložisko

Ako už bolo spomenuté v podkapitole 2.4.3, jazykový model v tejto implementácií je tvorený n-gramami. Pri implementácií n-gramového úložiska bolo potrebné brať do úvahy výslednú veľkosť úložiska, ale aj rýchlosť vyhľadávania v tejto dátovej štruktúre. Prvá implementácia úložiska bola založená na princípe popísanom v práci [8]. Tento spôsob používa vektor so samotnými n-gramami a druhý pomocný vektor s indexami používanými pri ich vyhľadávaní. Samotné vyhľadávanie je potom vykonávané za pomoci algoritmu binárneho vyhľadávania.

Druhý z implementovaných metód využíva natívne vlastnosti Javascriptového interpreteru. N-gramy sú uložené v Javascriptovom objekte a ich hodnoty indexované unikátnym kľúčom. Kľúče sú tvorené spojením indexu slov jednotlivých n-gramov. Interpret natívne ukladá a vyhľadáva v objektoch tohto typu za pomoci princípu hashovacej tabuľky. Tento princíp je v Javascripte veľmi často využívaný, a tak aj dobre optimalizovaný. To sa potvrdilo aj v testoch, kde sa toto riešenie ukázalo vzhľadom na výpočtový výkon lepšie.

Tabuľka 3.4: Porovnanie dvoch spôsobov implementácie n-gramového úložiska. 250-tisíc náhodných vyhľadávaní v bigramovom úložisku obsahujúcom 200-tisíc záznamov.

Implementácia	**Priemerný čas náhodného vyhľadávania/[s]**
Binárne vyhľadávanie	1.83
Vyhľadávanie v JS objekte	1.26

Vyhľadávanie v JavaScriptovom objekte

Javascriptový objekt bol zaobalený do triedy, ktorá implementuje metódy asociatívného pola. Bi-gramové a tri-gramové záznamy sú uložené samostatných inštanciách tejto triedy. Kľúč jednotnotlivých záznamov v týchto objektoch je tvorený spojením unikátnych identifkátorov slov, ktorým záznam prihlieha a následne implicitne zákodóvaný do unikátneho hashu. Hodnoty záznamov sú tvorené štruktúrami ktoré vyzerajú nasledovne:

```
export class BiGram {
  public score: number;
  public backoff: number;
}
```

```
export class TriGram {
  public score: number;
}
```

Pre vyhľadávanie n-gramov za pomoci popísaných asociatívnych polí je použitý nasledovný algoritmus:

Algorithm 5 Algoritmus použitý pre vyhľadanie v tri-gramovovom úložisku.

```
ngrams = ngramStorage
for token in emitted do
    actual = token.path
    prev1 = path.prev
    prev2 = path.prev.prev
    if prev1 == 0 then
        if ngram.bigrams[actual, prev1] then
            token.score += ngram.bigrams[actual, prev1].score
        else
            token.score += (actual.score + actual.backoff)
        end if
        continue
    end if
    if ngram.trigrams[actual, prev1, prev2] then
        token.score += ngram.bigrams[actual, prev1, prev2].score
        continue
    end if
    if ngram.bigrams[prev1, prev2] then
        backoff = ngram.bigrams[prev1, prev2].backoff
    end if
    if ngram.bigrams[actual, prev1] then
        token.score += (ngram.bigrams[actual, prev1] + backoff)
    else
        token.score += (actual.score + prev1.score + backoff)
    end if
end for
```

3.7.6 Optimalizačné metódy

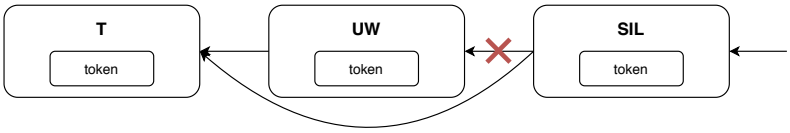
Pri použití rozsiahleho slovníku, ani po aplikácii Viterbiho kritéria, nemusí byť počet tokenov v rozpoznávacej sieti dostatočne malý pre spracovanie v reálnom čase. Počet tokenov spracovaných v každom kroku je teda nutné zredukovať pomocou optimalizačných metód popísaných v podkapitole 2.4.4.

Tokeny sú zoradené a je ponechaných iba N tokenov s najlepším skóre. Na zoradenie táto implementácia využíva voľne dostupnú knižnicu fast-sort[7]. Jej algoritmus je vysoko optimalizovaný, a tak nebolo nutné implementovať vlastné techniky zoradenia. Skóre posledného tokenu v zoradenom zozname je určené, ako hranica prípustného skóre. Od tokenu s najlepším skóre je odčítaná hodnota šírky lúča používaného v algortime beam pruning.

[7] https://www.npmjs.com/package/fast-sort

Ak, je hodnota prípustného skóre väčšia, ako upravená hodnota najlepšieho tokenu, je ním nahradená.

Hranica prípustného skóre je následne použitá pri prechode rozpoznávacou sieťou. Tokeny, ktorých skóre je pod prípustnou hranicou, sú zo siete odstránené. Vďaka implementácií rozpoznávacej siete za pomoci jednosmerne viazaných zoznamov je operácia odstránenia tokenu realizovaná odstránením uzla zoznamu, ktorý tento token obsahuje. Znázornenie odstránenia tokenu z rozpoznávacej siete je znázornené na obrázku 3.4.



Obr. 3.4: Odstránenie tokenu, ktorého skóre nespĺňa kritérium pre ponechanie v rozpoznávacej sieti. Po odstránení referencie na nežiadúci uzol je pamäť uvoľnená nasledujúcom cykle garbage collectoru.

3.8 Načítavanie modelov

Jedným zo zásadných implementačných problémov sa stalo načítavanie používaných modelov. Keďže modely neurónových sietí a jazykové modely môžu v textovom formáte dosahovať až stovky megabytov, ich načítavanie pri otvorení aplikácie vo webovom prehliadači nebolo dostatočne rýchle. Väčšina týchto dát je tvorená číslami s pohyblivou rádovou čiarkou. Vďaka tomu je možné dáta uložiť v binárnej podobe, čo drasticky znižuje ich veľkosť.

Javascript však nie je schopný načítavať statické binárne súbory z lokálneho úložiska. Kvôli tomuto obmedzeniu bol za pomoci Node.js a frameworku Express[8] vytvorený jednoduchý server. Tento server poskytuje vytvorenej knižnici všetky potrebné dáta v binárnej podobe alebo vo formáte JSON. Vďaka tomu sa podarilo výrazne znížiť čas načítavania aplikácie vo webovom prehliadači.

Neurónová sieť

Model neurónovej siete je tvorený jej jednotlivými vrstvami. Každá vrstva je tvorená váhami jej perceptronov. Všetky tieto dáta sa dajú reprezentovať v dátovom type **float32** vďaka čomu je ich zakódovanie do binárnej podoby a ich následne rozšifrovanie pomerne jednoduché. Každá vrstva je v tejto implementácií zakódovaná do vlastného binárneho súboru. Toto riešenie vyžaduje viac požiadaviek na server poskytujúci modelové dáta, no drasticky znižuje čas ich rozšifrovania na strane klienta.

Tabuľka 3.5: Porovnanie veľkosti modelu neurónovej siete v textovom a binárnom formáte

Veľkosť skrytej vrstvy	Textový formát [MB]	Binárny formát [MB]
500	7.2	2.1
700	32.5	7.3
1200	47.6	9.5

[8] https://expressjs.com/

Jazykový model

Jazykový model je dátovo najväčším zo všetkých použitých modelov. Jednotlivé záznamy sú však tvorené identifikátormi jednotlivých slov a n-gramovým skóre. Vďaka plne číselnej reprezentácií je model taktiež možné jednoducho zakódovať do jeho binárnej podoby. N-gramové pravdepodobnosti sú, tak, ako vrstvy neurónovej siete, reprezentované typom **float32**. Najväčší použitý slovník v tejto implementácií obsahuje pätnásťtisíc slov. Jeho identifikátory teda nie je nutné kódovať do 32 bitovej reprezentácie a postačujúcim dátovým typom je **uint16**.

Tabuľka 3.6: Porovnanie veľkosti jazykového modelu v textovom a binárnom formáte

Veľkosť slovníku	**Textový formát [MB]**	**Binárny formát [MB]**
7500	77.1	34.6
10000	85.5	37.4
12500	90.3	39.4
15000	94.5	40.9

3.9 Zhrnutie implementácie

Počas tejto práce bol vytvorený plne funkčný rozpoznávač reči založený na dynamickom dekodéri. Nahrávanie zvukového signálu je implementované za pomoci Web API poskytovaného webovými prehliadačmi. Získané dáta sú predspracované a rozdelené do rámcov, na ktoré je aplikovaná rýchla Fourierova transformácia za pomoci frameworku Tensorflow. Silové spektrum jednotlivých rámcov je použité pri detekcií rečovej aktivity a následne sú z neho vyrátané banky Mel filtrov. Tieto banky sú zaslané na predikciu do neurónovej siete, ktorá je graficky akcelerovaná frameworkom Tensorflow. Výsledné predikcie pravdepodobností výskytu jednotlivých foném sú spracovávané dynamickým dekodérom. Dekodér je tvorený akustickým uni-gramovým modelom, ktorý využíva algoritmus token passing a jazykovým modelom, ktorý obsahuje n-gramy. N-gramy sú aplikované po dynamickom vyhľadávaní v n-gramovom úložisku. Modely sú načítavané zo serveru vytvoreného pre tento účel v binárnej podobe alebo vo formáte JSON. Celý proces rozpoznávania je rozdelený do štyroch častí, ktoré sa vykonávajú paralelne na vlastných vláknach. Vďaka tomu bolo možné dosiahnuť rozpoznávania reči v reálnom čase. Na prezentáciu funkcionality bola vytvorená demo webová aplikácia s jednoduchým užívateľským prostredím.

Kapitola 4

Testovanie a vyhodnotenie

V tejto kapitole práca rozoberá techniku využitú na vyhodnotenie úspešnosti rozpoznávania. Taktiež poskytuje samotné vyhodnotenie implementácie za pomoci tejto metódy. Jednotlivé časti rozpoznávača boli samostatne testované v priebehu implementácie. Na verifikáciu priebežných výsledkov bol použitý referenčný script v jazyku Python dodaný vedúcim práce.

4.1 Word error rate

Pre vyhodnotenie úspešnosti rozpoznávania bol využitý takzvaný **Word error rate** (WER). Je to jedna z najpoužívanejších metód používaná na vyhodnotenie automatického rozpoznávania reči. Výpočet WER sa riadi rovnicou 4.1.

$$WER = \frac{S + D + I}{N} \tag{4.1}$$

Kde S je počet substitúcií, D je počet zmazaných slov, I počet vložených slov a N je počet referenčných slov. Meranie WER bolo prevádzané na vytvorenej demo aplikácií popísanej nižšie. Vyhodnotenie dát bolo realizované za pomoci voľne dostupného nástroja **asr-evaluation**[1].

4.1.1 Testovací dataset

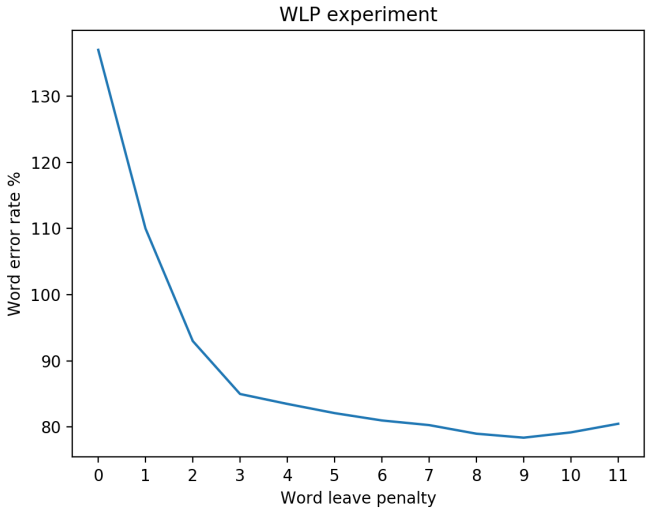
Pre testovanie úspešnosti bol použitý dátový set TED-LIUM[5]. Testovacie nahrávky boli upravené na formát WAV a prevzorkované na požadovanú frekvenciu 8000 Hz. Následne bolo vybraných desať nahrávok, ktorých dĺžka bola orezaná na 120 sekúnd. Výsledná testovacia sada teda pozostávala z 1200 sekúnd audio nahrávok.

4.1.2 Konfigurácia parametrov

Pred vyhodnotením nad celým testovacím dátovým setom bolo nutné experimentálne zistiť vhodné parametre ovplyvňujúce úspešnosť rozpoznávača. Konkrétne sa jedná o hodnoty `WORD_LEAVE_PENALTY` (WLP) a `LANGUAGE_SCALE` (LS). Prvá z týchto hodnôt reprezentuje penalizáciu tokenu pri opustení uni-gramovej rozpoznávacej siete. LS je koeficient vplyvu jazykového modelu na výsledky rozpoznávania.

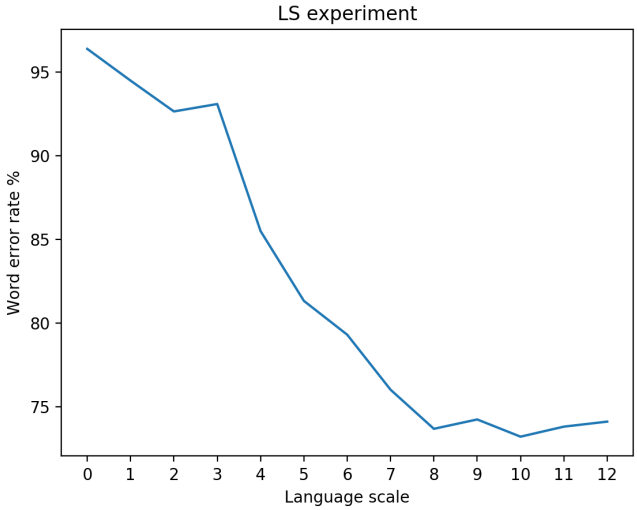
[1] https://github.com/belambert/asr-evaluation

Pri testovaní týchto dvoch hodnôt bol použitý slovník o veľkosti 10000 slov. WLP bolo určené za pomoci experimentu, pri ktorom bola opakovane vyhodnotená jedna z testovacích nahrávok. Interval posunu WLP bol rovný hodnote 1. Nasledujúci graf zobrazuje vývoj WER pri priebehu experimentu.



Obr. 4.1: Ovplyvnenie výsledkov rozpoznávania parametrom `WORD_LEAVE_PENALTY`.

Experiment ukázal, že použitie tohto parametru je nevyhnutné a jeho optimálna hodnota je 9. Pri vyšších hodnotách sa výsledky začali pomaly zhoršovať a tak bol experiment ukončený. Pre zistenie optimálnej hodnoty `LANGUAGE_SCALE` bol prevedený totožný experiment. Experiment je opäť graficky znázornený na nasledujúcom grafe.



Obr. 4.2: Ovplyvnenie výsledkov rozpoznávania parametrom `LANGUAGE_SCALE`.

Určenie správnej hodnoty, ktorá určuje ovplyvnenie jazykovým modelom, sa opäť ukázala ako nevyhnutná. Pri jej správnom určení, v tomto prípade na hodnotu 10, je možné výrazne zlepšiť výsledky rozpoznávania.

4.1.3 Výsledky testovania

Pre vyhodnotenie úspešnosti rozpoznávača boli použité neurónové siete s rôznym počtom perceptronov, v ich skrytej vrstve v kombinácií s viacerými slovníkmi, ktoré obsahovali rôzny počet slov. Vyhodnotenie je zobrazené v tabuľke 4.1.

Tabuľka 4.1: Vyhodnotenie testovacej dátovej sady. Slovník udáva počet slov známy dekodéru, bigrami spolu s trigramami udávajú veľkosť jazykového modelu a peceptrony značia veľkosť skrytej vrstvy neurónovej siete. Beam pruning bol nastavený na hodnotu 200 a počet udržiavaných tokenov v rozpoznávacej sieti na 4000.

Slovník	**Bigramy [M]**	**Trigramy [M]**	**Perceptrony**	**Average WER [%]**
5000	1.13	1.69	500	73.86
			1200	68.78
7500	1.38	1.80	500	74.35
			1200	69.11
10000	1.55	1.87	500	75.17
			1200	69.93
12500	1.69	1.90	500	75.43
			1200	70.13
15000	1.79	1.93	500	75.48
			1200	70.17

Testy boli prevedené aj za pomoci neurónovej siete rozpoznávajúcej trojstavové fonémy. Tento princíp by mal zaručiť presnejšie rozpoznanie foném, a tak aj väčšiu úspešnosť rozpoznávania. Tento predpoklad bol potvrdený výsledným WER. Avšak, použitie trojstavových fónem zapríčiňuje aj trojnásobný počet uzlov v rozpoznávacej sieti. Pri obsiahlom slovníku tento počet môže presiahnuť hranicu dovoľujúcu pre spracovávanie v reálnom čase. Vyhodnotenie za použitia trojstavových foném je zobrazené v tabuľke 4.2.

Tabuľka 4.2: Vyhodnotenie testovacej dátovej sady s neurónovou sieťou rozpoznávajúcou trojstavové fonémy. Slovník udáva počet slov známy dekodéru, bigrami spolu s trigramami udávajú veľkosť jazykového modelu. Neurónová sieť obsahovala v skrytej vrstve 700 perceptronov a nepoužívala takzvanú bottleneck vrstvu. Beam pruning bol nastavený na hodnotu 200 a počet udržiavaných tokenov v rozpoznávacej sieti na 8000.

Slovník	**Bigramy [M]**	**Trigramy [M]**	**Average WER [%]**
5000	1.13	1.69	62.91
7500	1.38	1.80	63.42
10000	1.55	1.87	64.15
12500	1.69	1.90	64.43
15000	1.79	1.93	65.07

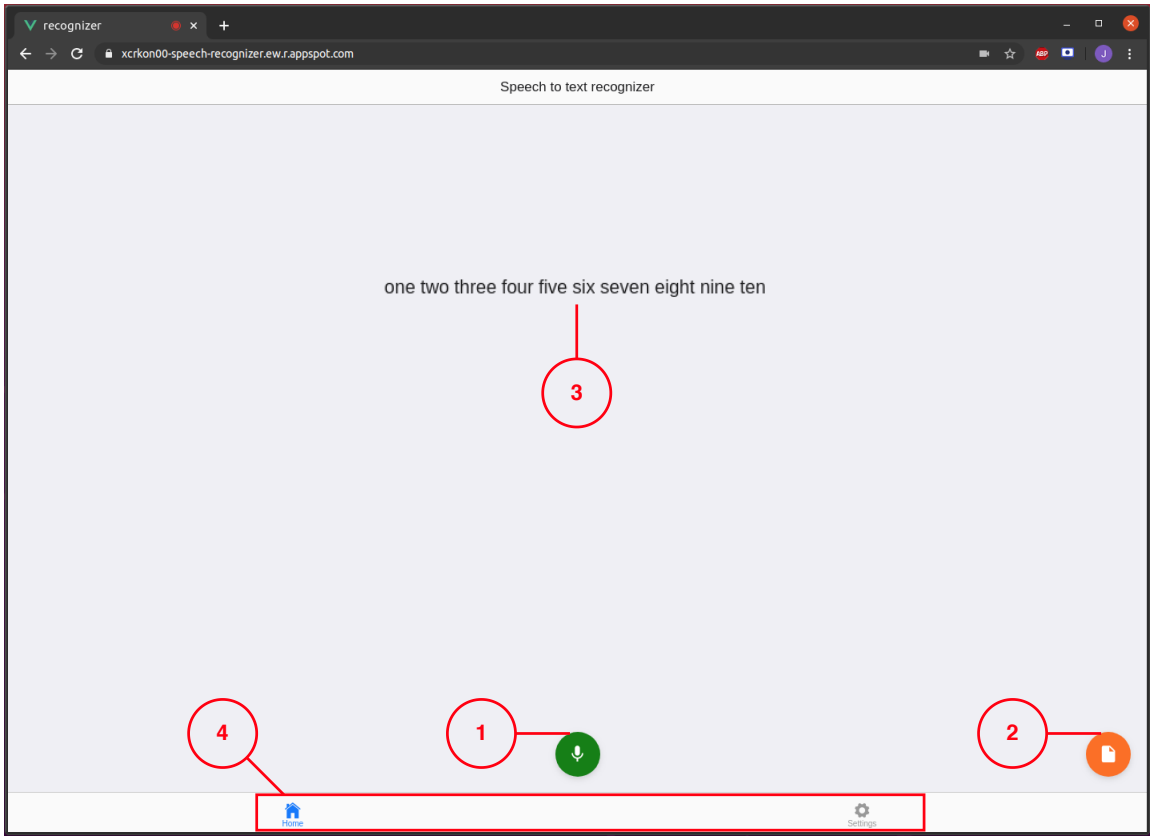
4.2 Demo aplikácia

Pre demonštráciu implementácie a jej vyhodnotenie bola vytvorená jednoduchá webová aplikácia za pomoci knižnice Vue.js popísanej v podkapitole 3.2. Grafické užívateľské prostredie (GUI) je plne adaptívne platforme a rozlíšeniu zariadenia, na ktorom je aplikácia spustená. Vďaka implementácií, za pomoci tejto knižnice, je možné simulovať natívne prvky mobilných zariadení vo webovom prehliadači, a tým zaistiť prívetivé užívateľské rozhranie aj v mobilných webových prehliadačoch.

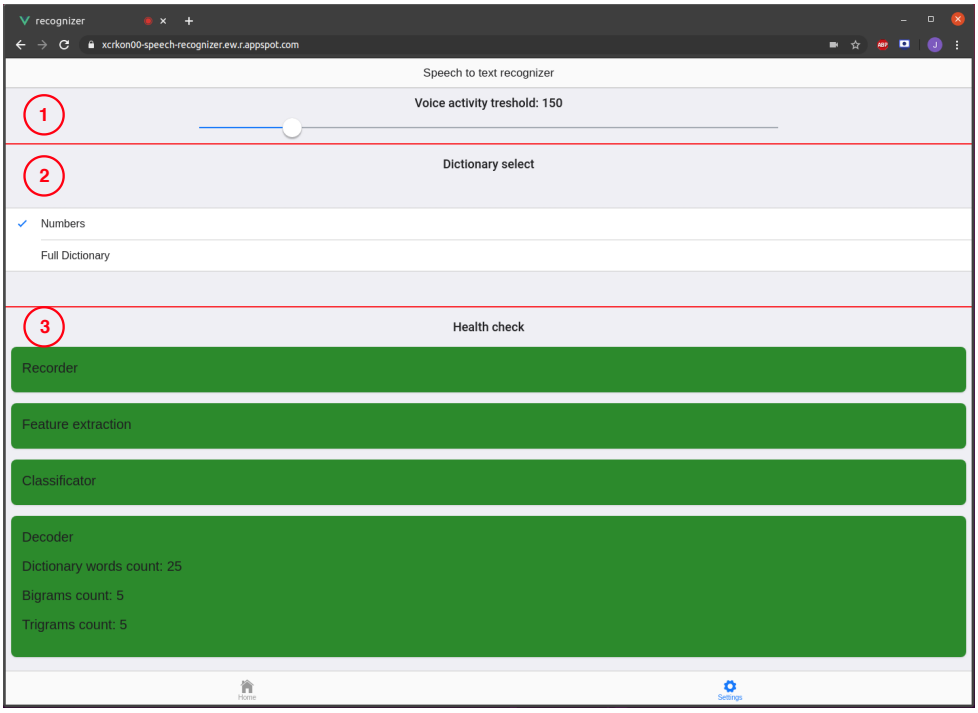
Domovská stránka ponúka hlavnú funkcionalitu rozpoznávača. Aktuálne najpravdepodobnejší prepis je zobrazovaný v reálnom čase a detekcia hlasovej aktivity je indikovaná zmenou farby tlačidla pre spustenie a zastavenie rozpoznávania. Farby tlačidla značia tieto funkcie: zelená - rozpoznávač je vo vypnutom stave, červená - rozpoznávač je v zapnutom stave a nie je detekovaná rečová aktivita, modrá - rozpoznávač je v zapnutom stave a detekuje rečovú aktivitu. Stránka nastavení ponúka možnosť zmeny hranice detekcie rečovej aktivity. Táto funkcionalita bola pridaná hlavne pre testovacie dôvody. Ďalej je na tejto stránke dostupná možnosť výberu slovníka pre dekodér a taktiež informuje o úspešnej alebo neúspešnej inicializácií jednotlivých častí rozpoznávača. Popis rozloženia GUI je možné vidieť na obrázku 4.3 a 4.4. Vzhľad GUI pre platformu Android a iOS je možné nájsť v prílohe B.

Aplikácie bola nasadená do produkcie za pomoci Google Cloud Platfrom[2] (GCP) a je dostupná na adrese:
https://xcrkon00-speech-recognizer.ew.r.appspot.com/

[2] https://cloud.google.com/



Obr. 4.3: Hlavná stránka demo aplikácie v plnej verzií webového prehliadača, kde tlačidlo označené číslom 1 ovláda rozpoznávanie v reálnom čase, tlačidlo označené číslom 2 ponúka možnosť spracovania audio súboru, 3 označuje aktuálny prepis a 4 označuje menu pre prepínanie medzi stránkami aplikácie.



Obr. 4.4: Stránka nastavení, kde 1 označuje blok pre nastavenie hranice detekcie rečovej aktivity, 2 označuje blok pre výber slovníka a 3 označuje kontrolný, ktorý informuje o inicializácií jednotlivých častí rozpoznávača a v prípade dekodéru poskytuje aj informácie o načítaných modeloch.

Kapitola 5

Záver

Cieľom tejto práce bolo implementovať jednoduchý rozpoznávač reči pre webový prehliadač za pomoci dostupných webových technológií a cieľ bol teda splnený.

Teoretická časť práce popisuje reprezentáciu zvukových dát vo výpočtových systémoch a spôsob akým sa s nimi pracuje. Ďalej je v tejto časti vysvetlený princíp extrakcie príznakov a jej jednotlivé kroky. Na popis extrakcie príznakov nadväzuje popis ich klasifikácie za pomoci neurónovej siete. V závere teoretickej časti je objasnená problematika dekódovania za pomoci skrytých Markovových modelov a jazykových modelov, ktoré sú tvorené n-gramovými pravdepodobnosťami.

Implementačná časť uvádza akým spôsobom sú dáta získavané a ďalej spracované. Špecifikuje spôsob akým sú dáta predávané v rámci programu a objasňuje, ako sú reprezentované. Klasifikácia získaných príznakov, ako aj všetky maticové operácie, sú graficky akcelerované za pomoci prostredia WebGL. Grafická akcelerácia je implementovaná pomocou frameworku Tensorflow. Výpočetne najnáročnejšou časťou je dekodér. Jednotlivé časti dekodéru boli implementované rôznymi spôsobmi, ktoré boli otestované a ich výsledky porovnané.

Vďaka viacvláknovému spracovaniu a grafickej akcelerácií bolo dosiahnutého rozpoznávania v reálnom čase. Výpočtové zariadenia s plnými verziami webových prehliadačov v testoch zvládli slovníky o veľkosti 12500 až 15000 slov. Mobilné zariadenie, používajúce mobilnú verziu totožného webového prehliadaču, umožňuje rozpoznávanie v reálnom čase so slovníkom obsahujúcim 1500 až 3000 slov. Úspešnosť rozpoznávača závisí na použitých modeloch a pohybuje sa v rozmedzí WER $63 - 75\%$.

5.1 Nadväzujúca práca

Vzhľadom na zložitosť celého projektu určite exituje mnoho spôsobov vylepšenia vrátane použitých dátových štruktúr a algoritmov, ale aj riešení, ktoré sú špecifické pre použitú platformu. Za zmienku stojí vytvorenie lepšej metódy pre zdieľanie dát medzi vláknami alebo implementácia výpočtovo najnáročnejších častí rozpoznávača pomocou jazyku C++ a následné prevedenie do WebAssembly[1]. Táto optimalizácia by mohla zaručiť výrazné zlepšenie výkonu rozpoznávača. Veľkým prínosom by bol aj ďalší výskum neurónových sietí, ktoré by poskytli lepší odhad pravdepodobností pre dekodér a tým zaručili väčšiu úspešnosť rozpoznávania.

[1] https://webassembly.org/

Literatúra

[1] Bracewell, R.; Bracewell, R.: *The Fourier Transform and Its Applications*. Electrical engineering series, McGraw Hill, 2000, ISBN 9780073039381.
URL https://books.google.cz/books?id=ZNQQAQAAIAAJ

[2] Gelfand, S.: *Essentials of Audiology*. Thieme, 2011, ISBN 9781604061550.
URL https://books.google.cz/books?id=SGdR8uCuRHUC

[3] Katz, S.: Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ročník 35, č. 3, 1987: s. 400–401.

[4] Psutka, J.; Müller, L.; Matoušek, J.; aj.: *Mluvíme s počítačem česky*. Prague: Academia, 2006, ISBN 80-200-1309-1, 752 s.
URL http://www.kky.zcu.cz/en/publications/PsutkaJ_2006_Mluvimes

[5] Rousseau, A.; Deléglise, P.; Estève, Y.: TED-LIUM: an Automatic Speech Recognition dedicated corpus. In *Conference on Language Resources and Evaluation (LREC)*, 2012, s. 125–129.

[6] Stevens, S. S.: A Scale for the Measurement of the Psychological Magnitude Pitch. *Acoustical Society of America Journal*, ročník 8, č. 3, Január 1937: str. 185, doi:10.1121/1.1915893.

[7] Van hamme, H.; Van Aelten, F.: An adaptive-beam pruning technique for continuous speech recognition. 11 1996, ISBN 0-7803-3555-4, s. 2083 – 2086 vol.4, doi:10.1109/ICSLP.1996.607212.

[8] VESELÝ, M.: *Dynamický dekodér pre rozpoznávanie reči*. Diplomová práca, Vysoké učení technické v Brně, Fakulta informačních technologií, 2017.

[9] Young, S.; Russell, N.; Thornton, J.: Token Passing: a Simple Conceptual Model for Connected Speech Recognition Systems. Technická správa, 1989.

[10] Čuba, E.: *Implementace jednoduchého rozpoznávače reči pre android*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2018.
URL https://www.vutbr.cz/studenti/zav-prace/detail/114554

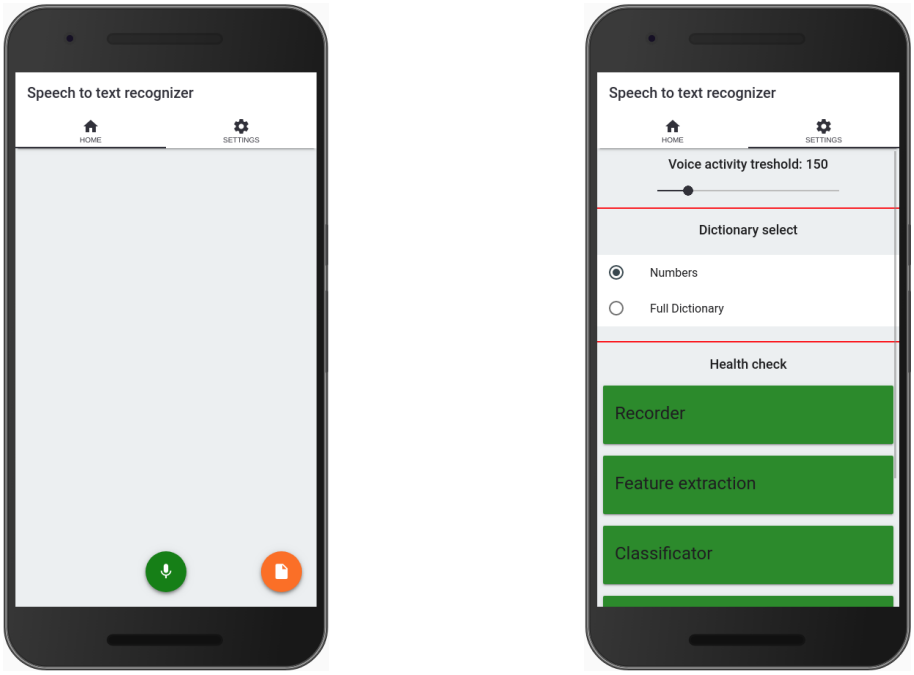
Príloha A

Obsah priloženého pamäťového média

- **thesis/pdf** Výsledný dokument *pdf*
- **thesis/latex** Zdrojové súbory latex na vytvorenie tohto dokumentu.
- **src/speech-regonizer** Zdrojové súbory obsahujúce demo aplikáciu a knižnicu pre rozpoznávanie reči
- **src/assets-server** Zdrojové súbory serveru poskytujúceho jazykové modely a neurónové siete
- **language_models/** Jazykové modely a scripty na ich spracovanie
- **neural_networks/** Neurónové siete a scripty na ich spracovanie
- **reference/** Obsahuje referenčný script v jazyku Python
- **test_set/** Testovací dátový set spolu z extrahovanými príznakmi jednotlivých zvukových súborov
- **video** Demonstračné video
- **poster** Plagát

Príloha B

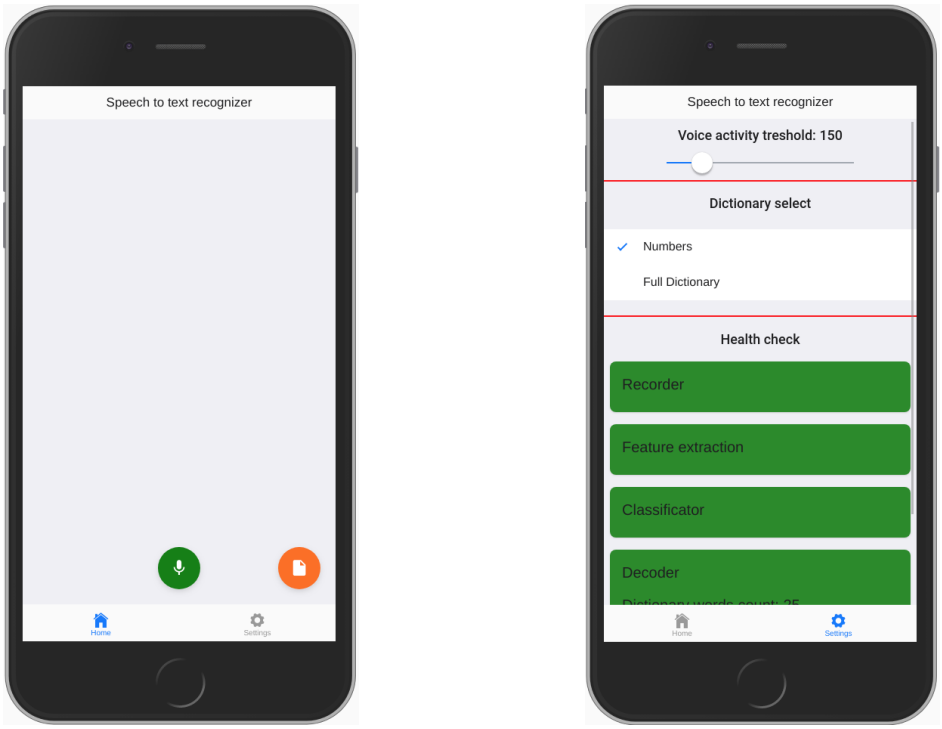
Prototypy užívateľského prostredia



(a) Hlavná stránka

(b) Stránka nastavení

Obr. B.1: Screenshot demo aplikácie v mobilnom prehliadači na platforme Android.



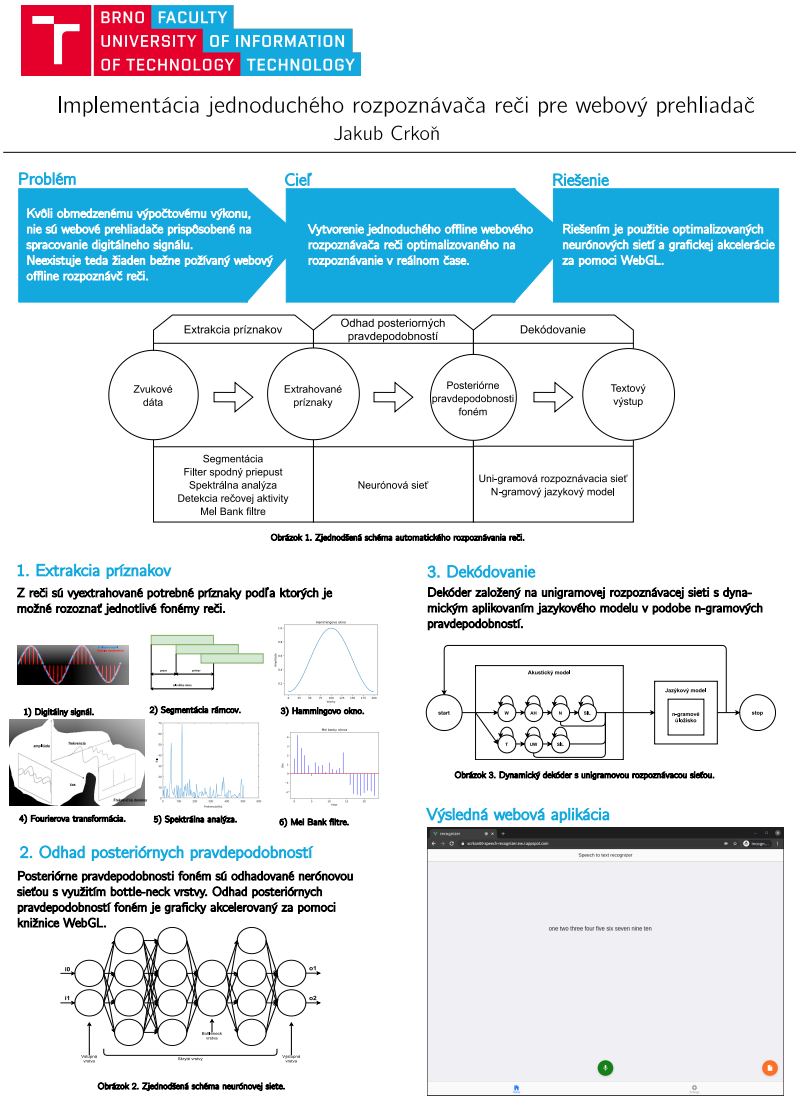
(a) Hlavná stránka

(b) Stránka nastavení

Obr. B.2: Screenshot demo aplikácie v mobilnom prehliadači na platforme iOS.

Príloha C

Plagát



Obr. C.1: Plagát