Fakulta elektrotechnická
Katedra radioelektroniky

Diplomová práce

# WEBový systém pro rozpoznávání řečníka na bázi i-vektorů a x-vektorů

Bc. Marek Vavřínek

**Studijní program:**
Elektronika a komunikace

**Specializace:**
Audiovizuální technika a zpracování signálů

**Vedoucí práce:**
doc. Ing. Petr Pollák, CSc.

**Praha, 2024**

# ZADÁNÍ DIPLOMOVÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vavřínek** Jméno: **Marek** Osobní číslo: **492091**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra radioelektroniky**

Studijní program: **Elektronika a komunikace**

Specializace: **Audiovizuální technika a zpracování signálů**

## II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

**WEBový systém pro rozpoznávání řečníka na bázi i-vektorů a x-vektorů**

Název diplomové práce anglicky:

**WEB System of Speaker Recognition Based on i-vectors and x-vectors**

Pokyny pro vypracování:

1. Seznamte se s problematikou rozpoznání řečníka na bázi i-vektorů (GMM-UGM) a x-vektorů (DNN) a realizujte systém s využitím nástrojů KALDI a dostupných vzorových receptů pro úlohu rozpoznávání řečníka.
2. Navržené systémy na bázi i-vektorů a x-vektorů otestujte na evaluačních datech z dostupných řečových databází z různých prostředí.
3. Vypracujte návrh WEBové aplikace pro demostrační účely úlohy rozpoznávání řečníka. Výsledná aplikace by měla umožňovat zápis nového řečníka, přidání zápisových dat pro existujícího řečníka a následně identifikaci neznámého řečníka na základě řečových dat nahrávaných přes vytvořené WEBové rozhraní.
4. Otestujte orientačně funkčnost navrženého systému v reálném on-line provozu.

Seznam doporučené literatury:

[1] J. Psutka, L. Miller, J. Matoušek, V. Radová: Mluvíme s počítačem česky. Academia, 2006.
[2] D. Yu, L. Deng. Automatic Speech Recognition A Deep Learning Approach. Springer-Verlag London. 2015
[3] D. Povey et al, The Kaldi Speech Recognition Toolkit. In Proc. of IEEE 2011 ASRU, Hawai, US, 2011.

Jméno a pracoviště vedoucí(ho) diplomové práce:

**doc. Ing. Petr Pollák, CSc. katedra teorie obvodů FEL**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **03.02.2024** Termín odevzdání diplomové práce: **24.05.2024**

Platnost zadání diplomové práce: **21.09.2025**

| doc. Ing. Petr Pollák, CSc. | doc. Ing. Stanislav Vítek, Ph.D. | prof. Mgr. Petr Páta, Ph.D. |
|---|---|---|
| podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | podpis děkana(ky) |

## III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.

Datum převzetí zadání Podpis studenta



© VUT v Praze, Design: VUT v Praze, VIC

**Prohlášení:**

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Praha 21. května 2024 Bc. Marek Vavřínek

**Poděkování:**

Rád bych poděkoval vedoucímu práce doc. Ing. Petru Pollákovi, CSc. za odborné vedení této práce a všem, kteří přispěli svým hlasem do testování výsledného on-line systému.

Bc. Marek Vavřínek

**Abstrakt**

Předložená práce se zabývá návrhem systému pro rozpoznávání řečníků a aplikace s webovým rozhraním pro demonstraci funkčnosti systémů rozpoznávání řečníka. Vlastní rozpoznávání řečníků je realizováno pomocí klasického řešení na základě GMM a tzv. i-vektorů, a následně pomocí moderního přístupu využívajícího neuronovou síť a tzv. x-vektorů. Oba systémy jsou implementované nástrojovou sadou Kaldi, natrénované a následně otestované na datech z databáze SPEECON. Webová aplikace je rozdělena na serverovou část a uživatelské rozhraní. Serverová část aplikace je vyvinuta s použitím frameworku Django, na který je systém Kaldi napojený pomocí kontejnerového řešení Docker, zatímco webové uživatelské rozhraní je vytvořeno pomocí frameworku React. Výsledný webový systém je otestovaný na různých počítačových platformách a použitém hardwaru v reálném on-line provozu. Dle zpětné vazby od uživatelů při používání aplikace nenastaly žádné problémy a systém fungoval správně. Dále byly orientačně potvrzeny výsledky funkčnosti systémů rozpoznávání řečníků.



### Klíčová slova:

SRE, GMM, DNN, i-vektor, x-vektor, Kaldi, React, Django



**Abstract**

This thesis focuses on the design of speaker recognition system and web interface application for demonstrating functionality speaker recognition systems. The speaker recognition itself is first implemented using classical approach based on GMMs and so-called i-vectors, and subsequently through a modern approach utilizing neural network and so-called x-vectors. Both systems are implemented using the Kaldi toolkit and are trained and then tested on data from the SPEECON database. The server part of the final application is developed using the Django framework, with the Kaldi system connected via Docker container solution, while the web user interface is created using the React framework. The resulting web system is tested on various computing platforms and hardware in real-time online operation. According to user feedback, there were no issues when using the application, and the system functioned correctly. Additionally, the performance results of the speaker recognition systems were indicatively confirmed.



### Keywords:

SRE, GMM, DNN, i-vector, x-vector, Kaldi, React, Django

# Obsah

**Seznam obrázků** **xiii**

**Seznam kódů** **xv**

**Seznam použitých zkratek** **xvii**

**1 Úvod** **1**

**2 Obecný základ rozpoznávání řečníka** **3**

2.1 Rozpoznávání řečníka . . . . . . . . . . . . . . . . . . . . . . . . . 3
2.1.1 Verifikace mluvčího . . . . . . . . . . . . . . . . . . . . . . 4
2.1.2 Identifikace mluvčího . . . . . . . . . . . . . . . . . . . . . 4
2.2 Systém na bázi i-vektorů . . . . . . . . . . . . . . . . . . . . . . . 5
2.2.1 MFCC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.2.2 Univerzální model pozadí . . . . . . . . . . . . . . . . . . . 6
2.2.3 Reprezentace pomocí i-vektorů . . . . . . . . . . . . . . . . 6
2.2.4 Míry podobnosti . . . . . . . . . . . . . . . . . . . . . . . . 7
2.3 Systém na bázi x-vektorů . . . . . . . . . . . . . . . . . . . . . . . 8
2.3.1 Reprezentace pomocí x-vektorů . . . . . . . . . . . . . . . . 8
2.3.2 Architektura DNN . . . . . . . . . . . . . . . . . . . . . . . 9
2.4 Hodnocení systémů SRE . . . . . . . . . . . . . . . . . . . . . . . . 9

**3 Implementace SRE v Kaldi** **11**

3.1 Nástroje Kaldi . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
3.2 Implementace systému na bázi i-vektorů . . . . . . . . . . . . . . . 13
3.2.1 Příprava dat . . . . . . . . . . . . . . . . . . . . . . . . . . 13
3.2.2 Výpočet MFCC a VAD . . . . . . . . . . . . . . . . . . . . 13
3.2.3 Trénování UBM a i-vektor extraktoru . . . . . . . . . . . . . 14
3.2.4 Extrakce i-vektorů . . . . . . . . . . . . . . . . . . . . . . . 15
3.2.5 Výpočet míry podobnosti . . . . . . . . . . . . . . . . . . . 15
3.2.6 Verifikace dané metody . . . . . . . . . . . . . . . . . . . . 16
3.2.7 Implementace výsledného systému pro použití v praxi . . . . . 17
3.3 Implementace systému na bázi x-vektorů . . . . . . . . . . . . . . . 19
3.3.1 Příprava příznakových koeficientů pro trénování . . . . . . . 19
3.3.2 Vytváření vzorů pro trénování . . . . . . . . . . . . . . . . . 20
3.3.3 Trénování DNN . . . . . . . . . . . . . . . . . . . . . . . . 20
3.3.4 Extrakce x-vektorů . . . . . . . . . . . . . . . . . . . . . . . 21
3.3.5 Implementace výsledného systému v praxi . . . . . . . . . . . 21
3.4 Experimentální část . . . . . . . . . . . . . . . . . . . . . . . . . . 23
3.4.1 Použitá data pro trénování . . . . . . . . . . . . . . . . . . . 23
3.4.2 Vyhodnocení i-vektor extraktoru . . . . . . . . . . . . . . . . 24

3.4.3 Vyhodnocení DNN extrahující x-vektory . . . 25
3.4.4 Porovnání systémů . . . 25

**4 Webová aplikace** **27**
4.1 Backend . . . 27
4.1.1 Django . . . 27
4.1.2 Vytvoření aplikace SREDEMO . . . 28
4.1.3 Modely . . . 29
4.1.4 Pohledy . . . 30
4.2 Frontend . . . 33
4.2.1 React . . . 33
4.2.2 Vytvoření uživatelského rozhraní . . . 33
4.2.3 Nahrávání zvukových nahrávek . . . 34
4.3 Nasazení na server . . . 35
4.3.1 Apache2 . . . 35
4.3.2 Docker . . . 36
4.3.3 Struktura databáze . . . 37
4.4 Vyhodnocení funkčnosti aplikace . . . 38
4.4.1 Popis testování aplikace . . . 38
4.4.2 Nastavení prahu . . . 39
4.4.3 Orientační výsledky vyhodnocení . . . 40
4.5 Jak změnit systém SRE v aplikaci . . . 41

**5 Uživatelské rozhraní webové aplikace** **43**
5.1 Popis uživatelského rozhraní . . . 43
5.2 Popis rozhraní administrátora . . . 45

**6 Závěr** **47**

**Literatura a použité zdroje** **49**

# Seznam obrázků


1 Porovnání skóre pro případ odmítnutí a přijetí . . . . . . . . . . . . . . 5
2 Blokové schéma výpočtu *MFCC* . . . . . . . . . . . . . . . . . . . . . 6
3 Rozložení skóre pro neoprávněné a oprávněné soudy. . . . . . . . . . . 10
4 Zavedená konvence adresářové struktury v Kaldi . . . . . . . . . . . . 12
5 Schéma postupu pro trénování i-vektor extraktoru . . . . . . . . . . . 13
6 Porovnání napočítaného skóre . . . . . . . . . . . . . . . . . . . . . . 16
7 Schéma výsledného systému na rozpoznání řečníků pro použití v praxi 17
8 Schéma postupu pro trénování x-vektor *DNN* . . . . . . . . . . . . . 19
9 Schéma nahrávání audio souborů na server . . . . . . . . . . . . . . . 30
10 *SREDEMO* - Schéma struktury databáze . . . . . . . . . . . . . . . . 37
11 Nastavení verifikačního prahu dle vyhodnocení na reálných datech . . 39
12 *SREDEMO* - Rozhraní nahrávání promluv . . . . . . . . . . . . . . . 44
13 *SREDEMO* - Výsledek rozpoznání se zpětnou vazbou . . . . . . . . . 44
14 *SREDEMO* - Zobrazení zapsaných řečníků . . . . . . . . . . . . . . . 45
15 *SREDEMO* - Zobrazení provedených rozpoznávání . . . . . . . . . . . 46
16 *SREDEMO* - Zobrazení a možnost úpravy vzorových vět . . . . . . . 46

# Seznam kódů

1 Výpis výsledku verifikace pro nastavený práh . . . . . . . . . . . . . . . 17
2 Přidaný kód do skriptu `allocate_egs.py` na řádku 262 [9] . . . . . . 20
3 Instalace frameworku `Django` na Linux server . . . . . . . . . . . . . 28
4 Obsah souboru `requirements.txt` . . . . . . . . . . . . . . . . . . . . . 28
5 Vytvoření `Django` projektu a aplikací `frontend` a `api` . . . . . . . . . 28
6 Zjednodušený příklad modelu v souboru `models.py` . . . . . . . . . . 29
7 Pohled pro přidávání nových vět a mazání vět . . . . . . . . . . . . 31
8 Pohled pro odesílání náhodných vět na základě požadavku . . . . . . 32
9 Založení projektu `Reactu` a nainstalování knihoven . . . . . . . . . . 34
10 Instalace a spuštění *WSGI* modulu pro `Apache2` . . . . . . . . . . . . 35
11 Přidaná konfigurace `Apache2` . . . . . . . . . . . . . . . . . . . . . . . 35
12 Instalace `Dockeru` a spuštění kontejneru . . . . . . . . . . . . . . . . 36
13 Příklad spuštění i-vektorového systému v jazyce `Python` . . . . . . . . 36
14 Výpis s přepisem promluvy a informacemi o řečníkovi . . . . . . . . . 38
15 Zjednodušený příklad úpravy funkce pro spuštění nové metody . . . . 42

# Seznam použitých zkratek

**CMVN** Cepstral Mean and Variance Normalization

**DNN** Deep Neural Network - hluboká neuronová síť

**EER** Equal Error Rate - míra stejné chyby

**EM** Expectation Maximization

**GMM** Gaussian Mixture Model - model Gaussovských směsí

**LDA** lineární diskriminační analýza

**MVC** Model-View-Controller - modelově-pohledové řízení

**MFCC** mel-frekvenční kepstrální koeficienty

**PLDA** pravděpodobnostní lineární diskriminační analýza

**SRE** Speaker Recognition Evalution - rozpoznávání řečníka

**UBM** Universal Background Model - univerzální model pozadí

**UI** User Interface - uživatelské rozhraní

**VAD** Voice Activity Detection - detekce řečové aktivity

**WSGI** Web Server Gateway Interface

# 1 Úvod

V 19. století se začaly objevovat první teoretické zmínky o automatických strojích, které by dokázaly provádět úkony jako člověk. Pokud by tyto stroje dokázaly počítat matematické příklady a pracovat s textem tak, jak to dokáží lidé, jistě by se musely dostat i do bodu, kdy by dokázaly ovládat prvky komunikace jako člověk a lidé by je pak mohli ovládat hlasem [1]. K samotné hlasové komunikaci patří mimo vlastního obsahu zprávy i informace o tom, kdo je autorem dané zprávy. Tuto informaci jsou si lidé nevědomky schopni automaticky domyslet podle hlasu osoby, kterou slyší. Avšak pro počítače byl do nedávna toto složitý úkol. V posledních desítkách let se však technologie dostala do takového bodu, kdy dokáže provádět složité úkony, jako je převod textu na řeč a s tím spojené rozpoznání řečníka na základě jeho hlasu.

V současnosti tak vznikají automatické systémy, které jsou schopné z promluvy neznámého řečníka extrahovat jeho hlas a z jeho hlasu pak získat informaci, kdo je autorem dané zprávy. Jinak řečeno, tyto systémy jsou schopny rozpoznat člověka po hlase, přestože rozpoznání hlasu není zdaleka nejlepší způsob, jak jednoznačně identifikovat jedince. Například rozpoznání dané osoby podle její jedinečné duhovky nebo otisků prstů je mnohem přesnější. V některých případech však jiná možnost než identifikace na základě hlasu není možná.

Během let se začaly objevovat různé systémy identifikace řečníka, příkladem může být automatické rozpoznání na základě hlasu na technické podpoře během telefonního hovoru. Hlasové rozpoznání pochopitelně není jedinou formou identifikace a je doplněno například kontrolní otázkou. Představuje však jistou formu zjednodušení a urychlení procesu a i pomyslný krok vpřed do budoucnosti, do modernější technologie. Tento systém může dobře fungovat díky velkým výpočetním výkonům serverů na straně technické podpory dané firmy.

V posledních letech se však výkon spotřební elektroniky natolik zvýšil, že i obyčejné mobilní telefony jsou schopné provádět složité operace, jako je například převod řeči na text při diktování krátké zprávy nebo dlouhých poznámek. Takový systém v řadě případů může být doplněn o identifikaci řečníka z toho důvodu, aby byly zpracovány pouze zprávy diktované vlastníkem telefonu a ne řeč kolemjdoucích na ulici nebo hovořících v metru.

Také v moderních vozidlech se stále častěji využívá hlasové ovládání, aby řidič během jízdy nemusel odvracet oči od silničního provozu a mohl bezpečně ovládat některé prvky svého vozu. V těchto automatických systémech hlasové rozpoznání řečníka poslouchá pouze příkazy od řidiče, a může tak zároveň představovat bezpečnostní opatření, aby spolucestující nenarušovali řidiči jeho ovládání přístrojů, nebo dokonce palubní desky auta.

V současnosti zaznamenaly také výrazný krok vpřed technologie pro virtuální a rozšířenou realitu a tzv. *spacial computing* systémy. Takové technologie již fungují jako samostatný počítač a k ovládání nevyužívají ani klávesnice, ani myši. Jednou z možností, jak zadat textový vstup, je pomocí hlasu. Opět zde dochází k rozpoznání

hlasu, aby zařízení nereagovalo na cizí jedince, ale aby jako hlasový vstup používalo výhradně hlas svého majitele.

Všechny tyto příklady zmiňují praktické použití technologie rozpoznání řečníka po hlase jako podpůrný systém pro systém převodu řeči na text, nebo pro hlasové ovládání. Pravděpodobně v těchto příkladech dané systémy porovnávají hlas pouze s uloženým vzorem majitele, tedy jedné osoby, případně několika málo osob, a ne se stovkami, nebo dokonce tisíci vzory hlasů jako v případě příkladu telefonické podpory veliké firmy. Díky tomu tyto rozpoznávače mohou být implementované i na jednodušších výpočetních strojích, jako jsou zmiňované mobilní telefony.

Tato práce se zabývá výše zmíněnou problematikou rozpoznání řečníka na základě jeho hlasu a popisuje systémy plnící tuto funkci v kapitole 2. V rámci práce jsou implementovány dva rozdílné systémy na rozpoznání řečníka. První systém využívá tzv. *i-vektory*, což je jednodušší, ale přesto dobře fungující řešení, které je snáze implementovatelné. Druhý systém je založen na tzv. *x-vektorech*, což je modernější přístup pro rozpoznávání řečníků využívající hlubokou neuronovou síť. Oba tyto systémy jsou realizovány pomocí nástrojové sady Kaldi. V kapitole 3 je popsáno toto řešení.

Druhou částí této práce je vytvoření webového systému pro nahrávání promluv řečníků a propojení se zmíněnými systémy na rozpoznání řečníků. Cílem je vytvoření funkčního demonstračního systému na rozpoznání řečníků, který je pojmenován *SREDEMO*. Tento systém je navržen tak, aby fungoval jako webové rozhraní, umožňující uživatelům snadný přístup a testování bez potřeby instalace dodatečného softwaru na jejich zařízení. Kromě toho bude *SREDEMO* sloužit v budoucnosti jako nástroj pro sběr a ukládání promluv od různých řečníků do databáze. Popis návrhu webového rozhraní a souvisejícího systému je popsán v kapitole 4, přičemž popis výsledného systému a jeho funkcí je v kapitole 5.

# 2 Obecný základ rozpoznávání řečníka

Tato kapitola popisuje teoretický základ systémů pro rozpoznávání řečníků a poskytuje pojmy spojené s touto technologií, které jsou nezbytné pro pochopení dalších souvislostí, ale i teoretický základ konkrétních metod s důrazem na systémy na bázi i-vektorů a x-vektorů, jež jsou následně implementovány v dalších kapitolách, a ukazuje, jak lze jednotlivé realizace výsledných systémů mezi sebou porovnávat.

## 2.1 Rozpoznávání řečníka

Úloha, při které je snahou rozpoznat osobu podle nějakých charakteristických rysů v jejím hlase, se nazývá rozpoznávání řečníka (*SRE* - Speaker Recognition Evalution). Charakteristické rysy v lidském hlase mohou být fyzikální, které jsou závislé na tvaru a fyzikálních rozměrech hlasového ústrojí a jiný člověk je nemůže imitovat, nebo naučené, které jsou dány způsobem mluvy daného řečníka a které jsou často pro identifikaci mluvčího jako celek důležitější, ale jsou naopak snadno imitovatelné.

Metody, které se snaží úlohu *SRE* realizovat, je možné rozdělit na subjektivní a objektivní. Subjektivní metody rozpoznávání řečníků často provádějí experti z oblasti fonetiky a lingvistiky. Patří mezi ně například forenzní lingvistika, poslechové metody a spektrografické metody. Využívá se analýza slovní zásoby, četnost používaných vět anebo grafické vyjádření zvukového signálu ve formě grafu spektra. Na rozdíl od objektivních metod se zde nepoužívají jednoznačné matematické postupy a nemůže být ani zajištěna konzistence výsledků pro dlouhé časové horizonty. Na druhé straně objektivní metody jsou postaveny na přesných matematických postupech, které dokáží z nahrávek vyextrahovat hlas a z něj jeho jedinečné charakteristiky. Tyto metody lze považovat za daleko přesnější, ale zároveň jsou mnohem náročnější na realizaci a na výpočetní výkon hardwaru.

Úloha rozpoznávání mluvčího může být textově závislá, částečně textově závislá, či textově nezávislá [2]. V textově závislých rozpoznávačích se provádí porovnávání nahrávek, ve kterých všichni mluvčí odříkávají shodný obsah, kterým může být nějaká věta, nebo číselná posloupnost, zatímco v textově nezávislém systému mluvčí mohou pronést libovolnou větu, často se však používají vzorové věty z rozsáhlých seznamů, aby nahrávky obsahovaly přirozenou mluvu dané osoby a ne jen citoslovce přemýšlení. Dále dle [2] se úloha rozpoznání řečníka dělí na úlohu identifikace v uzavřené množině a v otevřené množině, lze však ještě rozlišovat třetí typ úlohy, a tím je čistá verifikace.

### 2.1.1 Verifikace mluvčího

V kontextu *SRE* je úloha čisté verifikace jednou ze základních metod rozpoznávání. Při uvažování nejjednodušší formy je v systému zapsaný pouze jeden řečník. Během plnění požadavku o rozpoznávání pak systém pouze provádí ověření (verifikaci), že osoba zadávající požadavek o rozpoznání je tatáž osoba zapsaná v systému. Vyhodnocení se provádí na základě vypočtení míry podobnosti (skóre) mezi zapsaným řečníkem a neznámou osobou. Vypočtená hodnota míry podobnosti se pak porovná s nastaveným prahem. Pokud je skóre nižší než daný práh, neznámá osoba není považována za řečníka zapsaného v systému a je zamítnuta (`reject`), pokud však skóre je vyšší než daný práh, osoba byla rozpoznána a je akceptována (`accept`).

V systému ale může být zapsáno i více osob než jedna. V takovém případě rozpoznávaná osoba kromě nahrávky svého hlasu zadává i například své jméno, nebo identifikační číslo. Pak systém opět provádí verifikaci pouze s jedním zápisem v databázi. Oproti úlohám identifikace tedy systém během zpracování nemusí počítat míry podobnosti se všemi zapsanými osobami. Samotné rozpoznání pak může proběhnout velmi rychle.

Zmíněný práh musí být vhodně nastaven tak, aby kdokoliv, kdo není shodná osoba s řečníkem zapsaným v systému, byl jednoznačně zamítnut. Například v zabezpečovacích systémech je lepší práh zvolit přísnější, aby nežádoucí osoba nebyla vpuštěna, i za cenu, že v některých případech zapsaní řečníci mohou být občas neprávem zamítnuti. V takovém případě totiž mohou rozpoznání provést znovu.

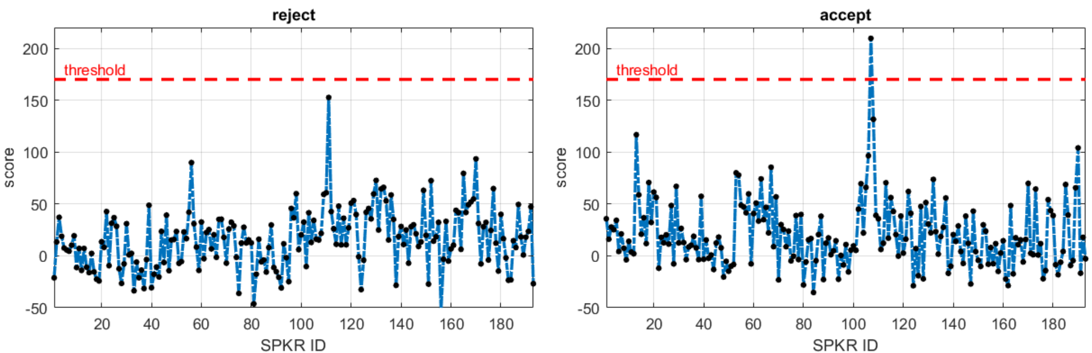
### 2.1.2 Identifikace mluvčího

Úloha identifikace se dělí na dvě různé úlohy podle typu množiny zapsaných řečníků. První a jednodušší je rozpoznávání na uzavřené množině. Jedná se o systémy, kde je dopředu jasné, že všechny rozpoznávané osoby jsou jistě zapsané v databázi a nemůže se stát, že neznámý mluvčí, kterého je třeba rozpoznat, není v zapsané množině. Opakem toho je rozpoznání na otevřené množině, tedy neznámý mluvčí nemusí být v zapsané množině řečníků.

Jednodušší varianta je označována jako identifikace na uzavřené množině. Jak již bylo naznačeno, systém předpokládá, že neznámá osoba je zapsaná v databázi, a je pouze potřeba určit (identifikovat), kdo daná osoba je. V takovém případě neznámá osoba pouze nahrává své promluvy a systém vypočítá míry podobnosti se všemi zapsanými jedinci v databázi. Samotná identifikace je pak dána jako prosté hledání maximálního skóre. Tedy výsledkem rozpoznání je určení maximální shody rozpoznávané osoby se zapsanými řečníky v databázi. Takový systém vždy jako výsledek dává právě jeden odpovídající zápis.

O trochu komplikovanější variantu identifikace, která je prováděná na otevřené množině, je možné označovat jako kombinaci identifikace a verifikace. V této úloze, stejně jako v předchozí, systém počítá skóre se všemi zápisy v databázi a snaží se najít nejlepší shodu. Zpočátku tedy předpokládá, že rozpoznávaná osoba je zapsaná v databázi, a hledá opět maximální skóre. Protože se však provádí rozpoznání na otevřené množině, nalezená největší shoda nemusí být správný výsledek. Následuje tedy úloha verifikace, kdy systém v tuto chvíli má ověřit, jestli rozpoznávaná osoba je opravdu shodná s nalezenou v databázi. Stejně jako v úloze čisté verifi-

kace se tak provede porovnáním s prahem. Zde je rozdíl oproti identifikaci. Pokud je dané skóre vyšší než nastavený práh, je řečník akceptován (`accept`) a výsledkem rozpoznání je jeho nalezená dvojice v databázi. Systém tak identifikuje danou osobu v množině zapsaných řečníků a zároveň provede ověření, že se nejedná o neznámou osobu, která není zapsaná v množině. Pokud je však maximální skóre nižší než nastavený práh, daný mluvčí je zamítnut (`reject`) a výsledkem identifikace není ničí jméno, neboť daný řečník není zapsaný v množině. Názorná ukázka fungování zmíněného systému je vidět na obrázku 1.



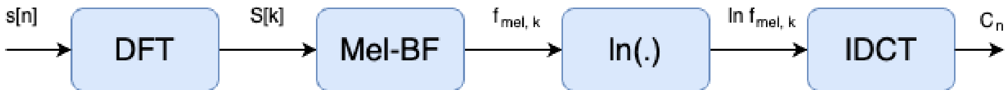

**Obrázek 1:** Porovnání dvou možných výsledků v identifikaci. Případ `reject`: největší shoda u řečníka 112 nepřesáhla práh. Případ `accept`: řečník 107 byl rozpoznán.

## 2.2 Systém na bázi i-vektorů

Systémy na bázi Gaussovských modelů směsí (*GMM* - Gaussian Mixture Model) byly dlouho standardní metodou pro rozpoznávání řečníků a stále se hojně využívají. Tyto systémy pracují s modelem *GMM*, který popisuje spojité rozložení pravděpodobnosti akustických rysů řečníka, nebo akustického modelu pozadí. Cílem těchto systémů je najít v databázi takovou shodu, která co nejpřesněji svou hustotou pravděpodobnosti odpovídá analyzovanému rozložení *MFCC* (mel-frekvenční kepstrální koeficienty) příznaků daného řečníka.

### 2.2.1 MFCC

Mel-frekvenční kepstrální koeficienty (*MFCC*) jsou koeficienty, které se používají k reprezentaci spektrálního obsahu lidské řeči. Tyto koeficienty jsou založeny na psychoakustických vlastnostech lidského sluchu a jsou navrženy tak, aby co nejlépe modelovaly lidské vnímání zvuku. Výpočet *MFCC* zahrnuje několik kroků, včetně předzpracování signálu, rozdělení signálu na rámce, výpočtu spektra, vytvoření banky melových filtrů, výpočtu logaritmu energie v každém filtru a nakonec aplikaci inverzní diskrétní kosinové transformace (*IDCT*) pro získání koeficientů *MFCC*. Blokové schéma výpočtu *MFCC* může být vyjádřeno pomocí obrázku 2. *MFCC* jsou následně použity jako vstupní příznaky pro další fáze zpracování řečových signálů, jako je například rozpoznávání řeči nebo klasifikace mluvčích.




**Obrázek 2:** Blokové schéma výpočtu *MFCC*

### 2.2.2 Univerzální model pozadí

Nejprve je třeba sestavit model každého mluvčího, aby systém pak mohl jednotlivé mluvčí porovnávat a určit shodu. Takový model se obecně sestavuje iteračním algoritmem *EM* (Expectation-Maximization), tedy pomocí hledání maximální věrohodnosti. Tento proces je ovšem výpočetně náročný a provádět ho pro každého řečníka by bylo velmi zdlouhavé a neefektivní. Z toho důvodu se často používá systém, při kterém se nejprve vytvoří univerzální model pozadí (*UBM* - Universal Background Model), který reprezentuje možné rozložení příznaků řečníků. Protože je trénován na velkém množství mluvčích, je na jednotlivém řečníkovi nezávislý. Pak se pro každého mluvčího jen tento *UBM* adaptuje, což je výrazně rychlejší proces.

Pro větší přesnost je dobré při trénování *UBM* používat typově shodné nahrávky s nahrávkami, které se budou pomocí vzniklého *UBM* rozpoznávat. Například použití shodných mikrofonů a zvukových řetězců. Dále pro větší přesnost dle [2] je možné natrénovat *UBM* samostatně pro ženy a samostatně pro muže a pak tyto modely spojit v jeden. Tím se zamezí vychýlení modelu v rámci pohlaví.

### 2.2.3 Reprezentace pomocí i-vektorů

Systému využívajícímu adaptaci modelu *UBM* se říká *UBM-GMM*. Takový systém reprezentuje jednotlivé promluvy $r$ mluvčího $s$ pomocí supervektorů $\mathbf{m}_{r,s}$, které shrnují různé akustické vlastnosti hlasu mluvčího a jsou odvozeny z modelu *UBM* a *GMM*. Tyto supervektory jsou pak využity k identifikaci mluvčího v rámci procesu rozpoznávání řečníků a jejich dimenze odpovídá násobku dimenze příznakového vektoru *MFCC* koeficientů a počtu komponent *GMM* modelu. Protože supervektor $\mathbf{m}_{r,s}$ má velkou dimenzi a obsahuje mnoho nadbytečných informací, hledají se takové vektory, které budou mít menší dimenzi, ale stále budou správně reprezentovat daného mluvčího. Těmito vektory jsou i-vektory $\mathbf{x}_{r,s}$, které je možné extrahovat podle vztahu

$$\mathbf{m}_{r,s} = \mu + \mathbf{T}\mathbf{x}_{r,s} \tag{1}$$

za znalosti transformační matice $\mathbf{T}$ pro redukci dimenze supervektorů $\mathbf{m}_{r,s}$ a supervektoru středních hodnot $\mu$ (střední hodnoty *UBM* modelu). Jednotlivé promluvy daného řečníka jsou pak popisovány samostatnými i-vektory. Pokud je nějaká promluva příliš dlouhá, může být rozdělena na části, které popisuje více i-vektorů. Během porovnávání různých řečníků se obvykle vypočítává průměr ze všech i-vektorů patřící k jednomu mluvčímu, aby daný mluvčí byl reprezentován pouze jedním i-vektorem, který je následně porovnáván s i-vektory reprezentujícími jednotlivé jiné řečníky.

### 2.2.4 Míry podobnosti

Porovnávání řečníků se provádí na základě míry podobnosti. Jedná se o číselnou hodnotu, které se obecně říká skóre, přičemž větší hodnota znamená větší podobnost mezi dvěma mluvčími. V systému rozpoznání řečníka se provádí výpočet mezi vstupním vektorem reprezentující neznámého mluvčího a daným vektorem zapsaného řečníka v databázi. Daným vektorem může být jak i-vektor, tak x-vektor. Při procházení celé databáze zapsaných řečníků se pak mění pouze druhý vektor reprezentující řečníka ve výpočtu. Během testování v uzavřené množině řečníků je pak identifikace dána jako prosté maximum ze všech napočítaných skóre.

Pro *SRE* na bázi i-vektorů, resp. x-vektorů, se využívají tři metody výpočtu skóre (míry podobnosti). Nejjednodušší z nich je kosinová vzdálenost, která se počítá podle následujícího vztahu

$$d_{cos} = \frac{\mathbf{x_1} \cdot \mathbf{x_2}}{|\mathbf{x_1}| \cdot |\mathbf{x_2}|} \; . \tag{2}$$

Jedná se o velmi jednoduchý a výpočetně nenáročný způsob výpočtu skóre, který může v některých případech, při kterých je dostatečné množství dostupných dat, fungovat velmi dobře. Pro přesnější vyhodnocování se nehodí, protože se snižujícím počtem promluv se rychle zvyšuje chybovost. Z tohoto důvodu jsou dále popsány další dvě metody výpočtu skóre, které zlepšují rozlišitelnost jednotlivých řečníků.

První z těchto metod výpočtu míry podobnosti je Lineární diskriminační analýza (*LDA*), která využívá lineární transformaci vektorů dané dimenze do takového podprostoru nižší dimenze, ve kterém se zlepší rozlišitelnost jednotlivých tříd, které je třeba klasifikovat. V tomto případě danými třídami klasifikace se rozumí jednotliví mluvčí. Lepší rozlišitelnost je pak dána menším rozptylem uvnitř tříd a větším rozptylem mezi jednotlivými třídami. Samotná transformace vektorů je dána transformační maticí, která je pro danou úlohu potřeba natrénovat na velkém vzorku dat. Trénování transformační matice $\mathbf{A}$ je dané jako optimalizace následujícího vztahu

$$J(\mathbf{A}) = \mathrm{tr}\Big((\mathbf{A}^T \mathbf{\Sigma}_W \mathbf{A})^{-1} (\mathbf{A}^T \mathbf{\Sigma}_B \mathbf{A})\Big) \; , \tag{3}$$

kde operátor $\mathrm{tr}(\cdot)$ je součet prvků na diagonále, $\mathbf{\Sigma}_W$ je celková kovariance uvnitř tříd počítaná přes všechny třídy a $\mathbf{\Sigma}_B$ je kovariance mezi třídami [3]. Postup výpočtu skóre pomocí *LDA* je pak následovně. Nejprve se vektory reprezentující daného řečníka transformují do nižší dimenze a následně se vypočítá kosinová vzdálenost. Vedlejším efektem velké redukce dimenze vektorů je, že výpočet kosinové vzdálenosti je pak výrazně urychlen, Tím pádem celková náročnost výpočtu skóre dle *LDA* nemusí být o moc větší než samotné kosinové vzdálenosti.

Druhá metoda, která zlepšuje rozlišitelnost jednotlivých řečníků, je pravděpodobnostní lineární diskriminační analýza (*PLDA*). Ta na rozdíl od *LDA* nesnižuje dimenzi vektorů, ale používá pravděpodobnostní model vektoru promluvy $s$ mluvčího $r$, který vychází z faktorové analýzy dle následujícího vztahu

$$\mathbf{x}_{r,s} = \mu + \mathbf{V}\mathbf{y}_s + \mathbf{U}\mathbf{w}_{r,s} + \epsilon_{r,s} \; . \tag{4}$$

Model obsahuje dvě složky, kterými jsou charakteristika daného mluvčího, která je pro všechny dané nahrávky stejná a představují ji první dva členy ve (4), a charakteristika proměnných akustických podmínek prostředí a analogové části nahrávacího

zařízení, kterou představují druhé dva členy. Rozložení pravděpodobnosti proměnných $\mathbf{y}_s$ a $\mathbf{w}_{r,s}$ je normální, a proto není třeba tyto parametry modelu trénovat. Ostatní parametry $\mu$, $\mathbf{V}$, $\mathbf{U}$ a $\epsilon_{r,s}$ je třeba získat natrénováním z velkého množství promluv pomocí *EM* algoritmu. Čím větší počet nahrávek je dostupný u daného řečníka, tím efektivnější jsou výsledky při rozpoznání oproti základním metodám výpočtu skóre, jako je kosinová vzdálenost [4].

Při rozpoznávání jsou porovnávány dvě sady nahrávek $\mathbf{O}_1$ a $\mathbf{O}_2$, o kterých zaručeně víme, že všechny nahrávky v dané sadě pocházejí od stejného mluvčího. Hypotéza $H_s$ říká, že obě sady nahrávek pocházejí od stejného mluvčího, zatímco hypotéza $H_d$ říká opak. Cílem je rozhodnout, která hypotéza je pravděpodobnější. Toho je docíleno pomocí výsledné podobnosti, která je pak dána podle vztahu (5), viz [5].

$$S_{\text{PLDA}} = \log \frac{P(\mathbf{O}_1, \mathbf{O}_2 | H_s)}{P(\mathbf{O}_1, \mathbf{O}_2 | H_d)} \tag{5}$$

## 2.3 Systém na bázi x-vektorů

Hluboké neuronové sítě (*DNN* - Deep Neural Network) se staly klíčovým prvkem v moderním rozpoznávání řečníků díky své schopnosti efektivně zpracovávat data a reprezentovat složité vzory v nich. Oproti tradičním metodám, jako je například systém založený na *GMM* na bázi i-vektorů, nabízejí systémy na bázi *DNN* větší flexibilitu a schopnost lépe zachytit vztahy v akustických rámcích představující identitu řečníka. Tím se stávají přesnějšími a robustnějšími v různých prostředích za předpokladu, že byly správně natrénované na velkém objemu dat.

Však ve všech případech nemusí být možné vyvinout systém na bázi *DNN* lépe fungující než klasické systémy na bázi *GMM*, navíc se nesmí opomenout mnohem složitější realizace *DNN* systémů a jejich náročnost na rozsáhlou databázi potřebnou pro proces trénování, a proto je třeba stále brát v potaz předchozí technologie a nesměřovat vždy rovnou k použití *DNN*.

### 2.3.1 Reprezentace pomocí x-vektorů

Podobně jako *GMM* systém na bázi i-vektorů ze vstupních *MFCC* koeficientů jednotlivých promluv vytváří reprezentaci ve formě i-vektorů, systém *SRE* využívající *DNN* mapuje promluvy na vektorovou reprezentaci mluvčího, tj. x-vektory. V obou případech mohou být jednotlivé promluvy proměnné délky a výsledné vektory jsou s fixní dimenzí. S x-vektory je pak možné pracovat podobně jako s i-vektory, tedy je možné použít stejné nástroje na výpočet skóre a porovnávání vektorů reprezentující řečníky, jako je například kosinová vzdálenost, *LDA* a *PLDA*.

Rozdíl oproti předchozímu systému v reprezentaci řečníka spočívá tedy ve způsobu, jakým jsou získány reprezentující vektory. U *DNN* systémů se již nepoužívá nic podobného adaptaci *UBM* na reprezentaci daného řečníka, nýbrž natrénovaná síť provádí kompletní transformaci příznakových koeficientů na výsledný x-vektor.

### 2.3.2 Architektura DNN

Architektura *DNN* pro výpočet x-vektorů se skládá z několika vrstev, které spolu tvoří složitý systém schopný zpracovávat a extrahovat informace ze vstupních dat. Celkový počet parametrů v *DNN* pro *SRE* systémy může dosahovat jednotek až desítek miliónů. Jednotlivé vrstvy systému na bázi x-vektorů jsou vidět v tabulce 1.

Vstupní vrstva přijímá data nejčastěji příznaky jako *MFCC* koeficienty stejně jako v případě *GMM* systému, nebo jinak upravené časové rámce zvukového signálu. Následují skryté vrstvy, které provádějí lineární a nelineární transformace vstupních dat. Tyto transformace umožňují síti extrahovat abstraktní informace z dat. Každá skrytá vrstva má svůj vlastní váhový vektor, který určuje, jak jsou vstupní data váhována a jak jsou transformována. Tyto váhové vektory jsou trénovány pomocí algoritmů učení, jako je zpětné šíření chyby, aby se minimalizovala chyba predikce a maximalizovala přesnost sítě [6]. Výstupní vrstva, kterou je v případě [7] vrstva *segment6*, převádí výstupy poslední skryté vrstvy na výsledné x-vektory, které jsou použity pro rozpoznání konkrétního řečníka.

Například v [7] je popsáno, že prvních 5 vrstev pracuje s řečovými rámci a s malým časovým kontextem, který pro 1 aktuální rámec je tvořen 15 rámci. Pak následuje vrstva statistického seskupování (pooling), která spojuje všechny výstupy na úrovni rámců z předchozí vrstvy a vypočítává jejich průměr a standardní odchylku, po níž následuje vrstva, která extrahuje x-vektory.

| **Layer** | **Layer context** | **Total context** | **Input x output** |
|---|---|---|---|
| frame1 | $[t - 2, t + 2]$ | 5 | 120x512 |
| frame2 | $\{t - 2, t, t + 2\}$ | 9 | 1536x512 |
| frame3 | $\{t - 3, t, t + 3\}$ | 15 | 1536x512 |
| frame4 | $\{t\}$ | 15 | 512x512 |
| frame5 | $\{t\}$ | 15 | 512x1500 |
| stats pooling | $[0, T]$ | $T$ | 1500Tx3000 |
| segment6 | $\{0\}$ | $T$ | 3000x512 |
| segment7 | $\{0\}$ | $T$ | 512x512 |
| softmax | $\{0\}$ | $T$ | 512x$N$ |

**Tabulka 1:** Struktura DNN, x-vektory jsou extrahovány na vrstvě *segment6*. Počet řečníků na trénování je reprezentován hodnotou **N** ve vrstvě *softmax*. [7]

## 2.4 Hodnocení systémů SRE

Jednotlivé systémy provádějící úlohu rozpoznání řečníků lze mezi sebou porovnávat různými způsoby. Ve spojitosti se systémem pracujícím v uzavřené množině je situace jednoduchá a často stačí prosté vyjádření chyby identifikace ve formě poměru počtu chybných identifikací ku celkovému počtu provedených identifikací, viz následující vztah:

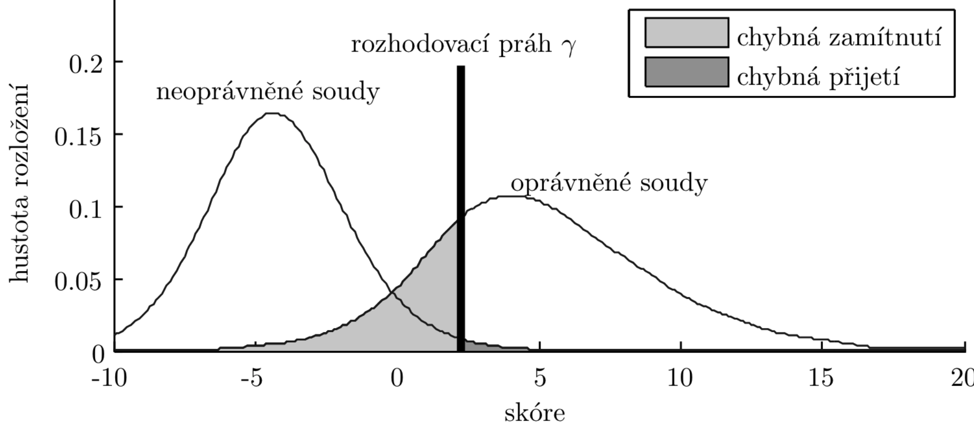
$$P_{err} = \frac{N_{err}}{N_{total}} \tag{6}$$

U systémů realizujících identifikaci v otevřené množině je však situace složitější. Zde totiž můžou nastat dva chybné stavy a prostá chyba identifikace je nedostačujícím popisem hodnocení. Chybě, při které je zapsaný řečník neprávem zamítnut, se říká míra chybného odmítnutí (*FRR* - False Rejection Rate) a popisuje, jaká část identifikací byla systémem zamítnuta, protože výsledné skóre nepřesáhlo nastavený práh, ačkoliv se jednalo o správnou osobu. Oproti tomu lze určit druhou chybu, při které je daný řečník nesprávně rozeznán a která je označována jako míra chybného přijetí (*FAR* - False Acceptance Rate).

$$FRR = \frac{N_{\mathrm{miss}}}{N_{\mathrm{tar}}} \cdot 100 \qquad\qquad FAR = \frac{N_{\mathrm{FA}}}{N_{\mathrm{non}}} \cdot 100\ , \tag{7}$$

kde $N_{\mathrm{miss}}$ je počet soudů, ve kterých došlo k chybnému zamítnutí, a $N_{\mathrm{tar}}$ je počet předložených oprávněných soudů. Obdobně $N_{\mathrm{FA}}$ je počet soudů, ve kterých došlo k chybnému přijetí, a $N_{\mathrm{non}}$ je počet předložených neoprávněných soudů. Dvě rozdílné míry chybovosti však komplikují vzájemné porovnání úspěšnosti dvou různých systémů, případně možnost posoudit zlepšení úspěšnosti systému při změně jednoho z parametrů, a proto je nejčastější vyjádření kvality systému pomocí jednočíselné hodnoty míry stejné chyby (*EER* - Equal Error Rate). Ta vyjadřuje, jaké budou míry chybného přijetí a chybného odmítnutí pro stav, kdy je práh verifikace nastaven tak, aby *FRR* a *FAR* byly stejně velké. Čím menší je výsledné *EER*, tím systém pracuje lépe. Na obrázku 3 je vidět grafické vyjádření oprávněných a neoprávněných soudů systému společně s chybami zamítnutí a přijetí pro rozhodovací práh.

Avšak v reálném použití není vhodné mít nastavený práh systému tak, aby byly chyby *FAR* a *FRR* shodné. Ideální je nastavení prahu takové, aby byla míra chybného přijetí nulová. Pak již nezáleží na tom, jestli míra chybného zamítnutí je nějaká malá nenulová hodnota. I přes tento fakt je hodnota *EER* dobrým jednočíselným ukazatelem, jak systém pracuje.




**Obrázek 3:** Rozložení skóre pro neoprávněné (vlevo) a oprávněné (vpravo) soudy. Světle šedá plocha vlevo od rozhodovacího prahu reprezentuje míru *FRR* a tmavá plocha vpravo míru *FAR* [3]

# 3 Implementace SRE v Kaldi

V této kapitole jsou popsané zvolené implementace systému *SRE*. V první části je rozebrán *GMM* systém na bázi *i-vektorů*, druhá část je věnovaná řešení systému na bázi *DNN* pracující s tzv. *x-vektory*. V obou případech byly systémy implementovány pomocí nástrojové sady Kaldi (*The Kaldi Speech Recognition Toolkit*) [8], která je vyvíjena pod licencí Apache verze 2.0 jako *Open-Source Software*. Je tedy možné volně využívat dostupné nástroje ve vlastních aplikacích.

## 3.1 Nástroje Kaldi

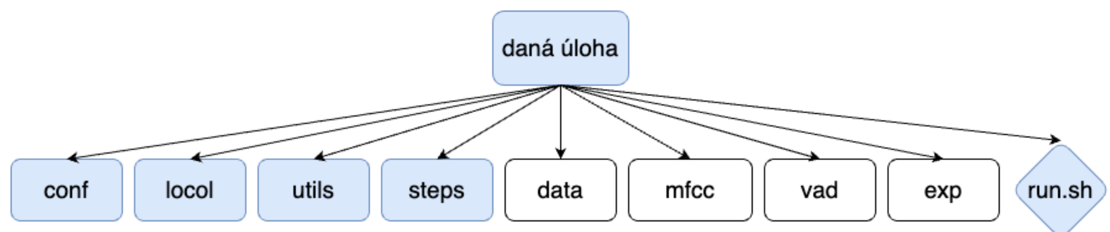
Knihovna Kaldi vznikla v roce 2009 na univerzitě Johna Hopkinse a stala se oblíbenou volbou programátorů v oblasti výzkumu systémů na zpracování řeči. V roce 2010 do projektu přispěli i akademici z Vysokého učení technického v Brně. Do tohoto projektu stále komunita přidává nové funkce a rozšiřuje soubor nástrojů. Kromě veškerých nástrojů pro zmíněné použití obsahuje knihovna Kaldi i vzorové skripty na mnoho jednotlivých podoblastí v tomto oboru. V terminologii Kaldi se vzorovým skriptům říká recepty. Vzorové skripty jsou součástí Kaldi, jako příklady, jak s knihovnou a danými nástroji pracovat. Běžnou praxí v této oblasti je použití daného receptu jako základu pro svou práci. Dostupný skript je však třeba modifikovat a následně na něm stavět dál.

Využívat vzorové příklady jako základ pro svou práci však nemusí být jednoduché. Autorům těchto receptů se podařilo dosáhnout nějakých výsledků při práci na jejich počítačích s jejich databází zvukových nahrávek a je velmi pravděpodobné, že na jiných počítačových systémech s jinými zvukovými databázemi nebudou skripty fungovat správně. Každý uživatel s tímto musí počítat a skript musí nejprve upravit pro svůj systém a pro svou databázi a až pak může na receptu stavět dále, což vyžaduje důkladnou znalost principů *SRE*.

Při řešení implementace obou zmíněných systémů jsem využil vzorové skripty, které jsou součástí instalace Kaldi. Pro *DNN* systém na bázi x-vektorů jsem se inspiroval receptem `sre16/v2` a začal jsem na jeho základu stavět svůj systém, zatímco *GMM* systém na bázi i-vektorů jsem vytvářel s využitím vzorového skriptu `sre10/v1`. Při práci na *GMM* systému jsem dále využil skripty, jejichž autorem je Michael Záruba, který se ve své diplomové práci [4] zabýval podobným systémem na bázi i-vektorů.

Jednotlivé části Kaldi jsou psané v různých jazycích. Výpočetně náročnější operace, které se zabývají například matematickými transformacemi, jsou napsané v objektovém jazyce `C++`, zatímco skripty zabývající se trénováním neuronových sítí jsou napsané v interpretovaném jazyce `Python`. Méně náročné operace jsou sestavené jako skripty v jazyce `Perl` nebo v interpretovaném terminálovém jazyce `BASH`. Mezi tyto méně náročné operace v Kaldi patří například hlavní spouštěcí skript, který volá jednotlivé procedury, ale i například části, které připravují data. Nejen kvůli

této rozmanitosti programovacích jazyků, ale i kvůli faktu, že je knihovna vyvíjena jako *Open-Source Software*, je náročné ji instalovat přímo na daný výpočetní stroj. Z tohoto důvodu je jednodušší použít Kaldi v kontejnerovém formátu, například v Dockeru nebo v jiném virtualizačním nástroji. Tento přístup zajišťuje, že daná aplikace běží spolehlivě a konzistentně napříč různými platformami, resp. operačními systémy.

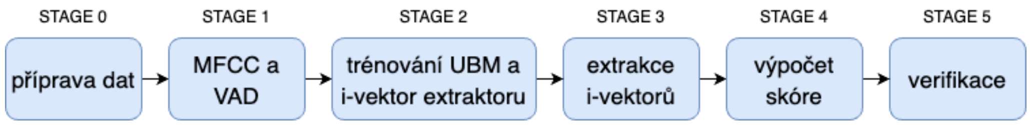



**Obrázek 4:** Zavedená konvence adresářové struktury v Kaldi

V Kaldi je zavedená konvence pro adresářovou strukturu každého receptu, resp. úlohy. Ve složce je několik podadresářů a hlavní skript `run.sh`, kterým se spouští samotná úloha, viz obrázek 4. V podadresáři `conf` se nachází statické konfigurace k dané úloze. Vedle něho ve složce `local` uživatel umisťuje své vlastní skripty, které jsou nezbytné pro danou úlohu a specifické pro použití dané databáze apod. Na rozdíl od toho v adresáři `utils` jsou připravené pomocné skripty, které pracují se strukturovanými textovými soubory, které jsou typické pro práci v Kaldi. Dále složka `steps` obsahuje jednotlivé výpočetní kroky. Za podobným účelem bývá v úlohách *SRE* ještě přítomna složka `sid`, která v jiných úlohách nemusí být přítomna. Nakonec složky `data`, `mfcc`, `vad` a `exp` se vytvářejí během běhu programu. Ve složce `data` se uchovávají pomocné informace o zvukových nahrávkách v textových souborech, adresáře `mfcc` a `vad` obsahují napočítané dodatečné informace z jednotlivých nahrávek a v adresáři `exp` vznikají výsledky daného experimentu úlohy. Na obrázku 4 jsou tyto čtyři poslední složky označeny bíle.

## 3.2 Implementace systému na bázi i-vektorů

Principem tohoto systému je natrénování extraktoru, který bude schopný z nahrávek daného řečníka extrahovat i-vektory, které následně budou sloužit pro porovnávání a tedy pro rozpoznávání řečníků. Návrh tohoto systému je složen z 6 kroků v Kaldi označovaných `STAGES`. Schéma postupu je znázorněno na obrázku 5. V následujících sekcích jsou tyto kroky postupně rozebrány. Po úspěšném natrénování extraktoru a jeho verifikaci následuje v sekci 3.2.7 implementace výsledného systému pro použití v praxi. Na závěr v kapitole 3.4 je uvedeno vyhodnocení dosaženého systému a jeho porovnání se systémem na bázi *x-vektorů*.




**Obrázek 5:** Schéma postupu pro trénování i-vektor extraktoru

### 3.2.1 Příprava dat

Prvním krokem u většiny skriptů využívajících nástroje Kaldi je příprava dat. Tento první krok je označován jako `STAGE 0`. V rámci přípravy dat jsou lokalizovány všechny zvukové nahrávky jednotlivých řečníků. Jedná se o méně náročné operace, které jsou často prováděné ve skriptovacím jazyce `Perl`, nebo `BASH`. Daný soubor audio nahrávek je následně členěn na tři sety dat. Soubor nahrávek `train` bude v následujících krocích sloužit k trénování extraktoru, soubory `enroll` a `test` budou po natrénování extraktoru sloužit k verifikaci systému.

Skripty, které provádějí přípravu dat, nepřemisťují původní nahrávky z databáze, ale vytvářejí textové soubory, které popisují umístění souborů a jak s nimi pracovat. V tomto kroku vzniklé textové soubory obsahují seznamy jednotlivých souborů s promluvami a cestami k umístění těchto souborů. Případně může být tento seznam doplněn i o příkazy, jak nahrávky například převzorkovat. Jedná se o soubory `wav.scp`, které jsou vytvářeny pro každý set samostatně a ukládány do samostatných složek dle názvu setu do složky `data` dle obrázku 4. Dále jsou vytvářeny soubory `utt2spk`, `spk2utt`, které obsahují informace o tom, jaké promluvy patří k danému řečníkovi, a obráceně, kdo je řečníkem daných promluv. Posledním krokem přípravy dat je vytvoření souboru `trials`. Tento seznam slouží jako podklad pro výpočet skóre a pro verifikaci natrénovaného extraktoru.

### 3.2.2 Výpočet MFCC a VAD

Při práci na systémech zabývajících se zpracováním řeči se často používají jako příznaky *MFCC* koeficienty. U *GMM* systému na bázi i-vektorů tomu není jinak. Proto druhým krokem (`STAGE 1`) je výpočet *MFCC* příznakových koeficientů připravených promluv z předchozího kroku. V Kaldi ve složce `steps` je k tomuto účelu připravený skript `make_mfcc.sh`, který se použije pro výpočet zmíněných *MFCC* koeficientů postupně pro všechny tři množiny řečníků (`train`, `enroll` a `test`).

U použitého skriptu stějně jako u většiny připravených skriptů v Kaldi lze zadat parametr `nj`, který představuje počet dostupných výpočetních jader. Skript tak na základě tohoto parametru rozdělí data pro zpracování na počet podmnožin dle parametru `nj`, které se následně zpracovávají paralelně. Výsledkem každé paralelní úlohy je pak samostatný binární soubor ve složce `mfcc` obsahující příznakové koeficienty. Výhodou tohoto přístupu je veliká rychlost výpočtu na systémech s větším počtem výpočetních jader.

Druhou částí tohoto kroku je výpočet řečové aktivity (*VAD* - Voice Activity Detection). Tato informace je dále využívána v následujících krocích společně s *MFCC*. Pro výpočet *VAD* je využit obdobný skript jako k výpočtu *MFCC*, a to skript `compute_vad_decision.sh` ze složky `sid`. Tento skript využívá jednoduchý algoritmus na bázi logaritmu energie řečového segmentu. Výstupem jsou pak opět samostatné binární soubory obsahující informace o tom, které segmenty v daných nahrávkách jsou řečové a které nikoliv.

### 3.2.3 Trénování UBM a i-vektor extraktoru

Následující třetí krok (`STAGE 2`) je nejdůležitější. V tomto kroku probíhá natrénování výsledného extraktoru i-vektorů. Jedná se o výpočetně velmi náročný proces, při kterém se postupně zpracovávají všechny promluvy z množiny `train`, respektive jejich *MFCC* koeficienty. Proces trénování extraktoru je rozdělen do tří částí, které využívají připravené skripty v Kaldi ze složky `sid`. Všechny připravené skripty opět dokáží provádět výpočty paralelně na více procesorových jádrech najednou a výrazně tak urychlit proces trénování.

V první částí se trénuje univerzální model pozadí (*UBM* - Universal Backgournd Model) s diagonální kovarianční maticí. Tuto úlohu provádí skript ze složky `sid` `train_diag_ubm.sh`, který na základě zpracovaných nahrávek (*MFCC* koeficienty společně s *VAD*) z datasetu `train` vytvoří model s diagonální kovarianční maticí, který uloží do samostatné složky ve složce `exp`. Tento proces je prováděn v několika iteracích pomocí *EM* algoritmu a u výsledného modelu lze nastavit počet *GMM* komponentů, které jsou v původním receptu nastaveny na hodnotu 64, což v této práci zůstalo zachováno.

Během druhé části se natrénuje *UBM* s plnou kovarianční maticí. K tomu je využit skript `train_full_ubm.sh`, který na základě připraveného *UBM* s diagonální maticí a *MFCC* koeficientů společně s *VAD* z množiny `train` natrénuje *UBM* s plnou kovarianční maticí. Opět tento proces je proveden v několika iteracích pomocí *EM* algoritmu a výsledek je uložen do samostatné složky ve složce `exp`.

Závěrem v třetí části je natrénování výsledného extraktoru i-vektorů. Toho je docíleno pomocí skriptu `train_ivector_extractor.sh`, který využívá vytvořený *UBM* s plnou kovarianční maticí. Zde lze nastavit dimenzi výsledných i-vektorů, která ve výchozím nastavení je nastavena na hodnotu 400. Výsledný extraktor je uložen do složky `exp/extractor`.

### 3.2.4 Extrakce i-vektorů

V této chvíli je systém připravený pro použití. V předchozím třetím kroku byl totiž natrénovaný extraktor i-vektorů, který již lze použít v *SRE* systémech, ovšem tento extraktor není otestovaný, jak se chová a jestli je vůbec vyhovující k použití. Tímto se zabývají následující kroky 4 až 6 (`STAGES 3 - 5`).

Ve čtvrtém kroku se tedy použije natrénovaný extraktor a extrahují se i-vektory ze všech tří data setů `train`, `enroll` a `test`. Toho je dosaženo pomocí skriptu `extract_ivectors.sh`, který je součástí distribuce Kaldi ve složce `sid`. Tento skript pro každý data set extrahuje i-vektory z řečových segmentů dle *VAD* z vypočítaných *MFCC* příznaků jednotlivých promluv za použití natrénovaného extraktoru a uloží je do podsložek příslušících k jednotlivým data setům ve složce `exp/ivectors`. Oproti trénovací fázi je toto výrazně rychlejší proces. Opět u tohoto skriptu je možná paralelizace. Dle počtu dostupných výpočetních jader procesoru skript rozdělí zpracování promluv do jednotlivých procesů. V použití v praxi pak tento krok je velmi rychlý, protože se neextrahují i-vektory všech promluv v databázi najednou, ale vždy jen několika málo promluv jednoho řečníka, který se právě zpracovává, a nemusí se tedy provádět jako paralelní výpočet.

### 3.2.5 Výpočet míry podobnosti

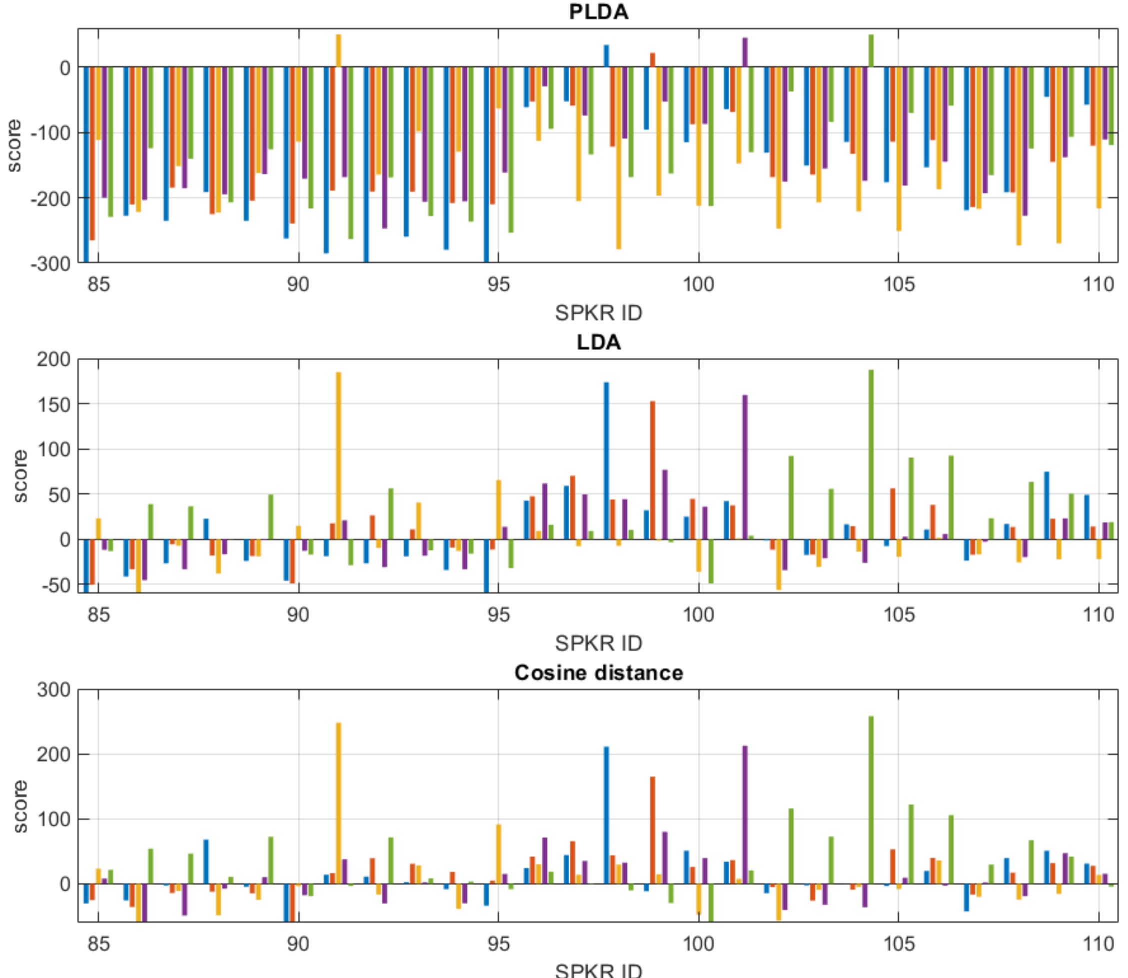
Předposledním pátým krokem (`STAGE 4`) v úloze natrénování extraktoru i-vektorů je samotné porovnávání i-vektorů, jinak řečeno výpočet míry podobnosti (skóre). Cílem tohoto kroku je zjistit, jak je natrénovaný extraktor úspěšný. V použití v praxi pak tento krok představuje samotné porovnávání dvou řečníků a výsledné vyhodnocení, zda se jedná o téhož řečníka, či nikoliv. A tedy představuje výsledné rozpoznání řečníků.

Výpočet skóre je v trénovací úloze implementován třemi různými metodami, aby bylo možné mezi sebou porovnat jednotlivé metody a vybrat tu nejlepší, viz sekce 2.2.4. V Kaldi se míry podobnosti počítají na základě souboru `trials`, který obsahuje postupně všechny dvojice mluvčích, které mají mezi sebou být porovnány, a pomocí připravených nástrojů.

První a nejjednodušší metodou pro zjištění míry podobnosti je kosinová vzdálenost. Ta se počítá pomocí nástroje `ivector-compute-dot-products`, který neobsahuje normalizaci pomocí délek daných vektorů jako ve vztahu (2), ale i přesto je v receptech Kaldi označován jako kosinová vzdálenost.

Další metodou je *LDA*, u které se používá transformační matice, která byla vypočítána na základě nahrávek z množiny `train`. Pro sestavení transformační matice se v Kaldi používá nástroj `ivector-compute-lda`. Při výpočtu skóre se i-vektory nejprve transformují touto maticí a následně se spočítá kosinová vzdálenost.

Poslední metodou výpočtu míry podobnosti je *PLDA*. Ta na rozdíl od *LDA* používá pravděpodobnostní model, který byl natrénován v Kaldi pomocí nástroje `ivector-compute-plda` opět na množině nahrávek `train`. Výpočet skóre dle *PLDA* je odlišný oproti *LDA* a provádí se na to určeným nástrojem `ivector-plda-scoring`. Na obrázku 6 je vidět grafické porovnání všech tří zmíněných metod pro 5 neznámých mluvčích, přičemž v této ukázce každá z nich dává jasný jeden výsledek ve formě maximálního skóre.




**Obrázek 6:** Porovnání napočítaného skóre pomocí metod *PLDA*, *LDA* a kosinová vzdálenost. Určení 5 řečníků (ID: 91, 98, 99, 101 a 104) ze zapsaných 193 řečníků

Všechny tři metody ukládají své výstupy výpočtů skóre do samostatných souborů pro možnost porovnání metod. Závěrem tohoto pátého kroku je výpočet chyb identifikace jednotlivých metod pomocí skriptu `compute_identify_err.sh`. Ten prochází vypočítaná skóre a u každého neznámého mluvčího hledá maximální hodnotu. Pokud maximální hodnota skóre není u shodných mluvčích, inkrementuje čítač chyb. Tento postup je pouze platný u uzavřené množiny řečníků. Pokud se jedná o otevřenou množinu, tedy testovaní mluvčí nemusejí být v zapsané množině, čítač chyb se neinkrementuje. Zda se jedná o otevřenou, nebo uzavřenou množinu se nastavuje během přípravy dat souborem `trials`. Dosažené výsledky jsou uvedeny v kapitole 3.4.

### 3.2.6 Verifikace dané metody

Posledním krokem je otestování natrénovaného extraktoru a tří metod výpočtu skóre (kosinová vzdálenost, *LDA* a *PLDA*) na otevřené množině. Kromě zapsaných mluvčích jsou systému předávány na otestování i řečníci, kteří nejsou zapsáni do dané množiny. Tímto šestým krokem (`STAGE 5`) je verifikace.

Při provádění verifikace je potřeba mít připravený soubor `trials`, který obsahuje jak zapsané, tak nezapsané řečníky. V takovém nastavení pak výsledek chyby identifikace v předchozím kroku vyjde 0 pro nezapsané osoby, protože v testování na otevřené množině se čítač chyb neinkrementuje pro nezapsané osoby. Během verifikace jsou použité výpočty skóre jednotlivých metod z předchozího kroku.

Prvním krokem verifikace dané metody je vytvoření seznamu, který obsahuje dvojice řečníků, pro které bylo napočítáno maximální skóre, a dané maximální skóre. Dále je tento seznam doplněn o informaci, zda se jedná o téhož řečníka (`target`), či nikoliv (`nontarget`). Tato informace v reálném provozu není k dispozici. Systém pro každý řádek provede verifikaci. Na základě maximálního skóre a nastaveného prahu doplní na řádek informaci `accept`, nebo `reject`. Na závěr systém doplní u každého řádku, zda se systému podařilo provést verifikaci správně (`OK`), či nikoliv (`ERR`), podle hodnot `target` a `nontarget`. Příklad výpisu takového výsledku je vidět v kódu 1.

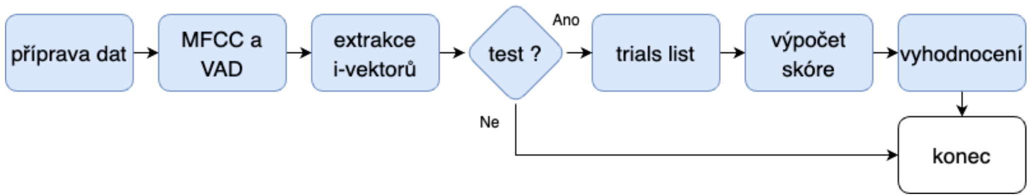
```
1 SA433 SA566 80 nontarget - reject OK
2 SA488 SA488 99 target - reject ERR
3 SA561 SA560 101 nontarget - accept ERR
4 SA564 SA564 120 target - accept OK
```

**Kód 1:** Výpis výsledku verifikace pro nastavený práh 100 u dané metody

V druhém kroku verifikace se na základě maximálního skóre a informace `target` (nebo `nontarget`) vypočítá hodnota *EER* systému, viz sekce 2.4. K tomu v Kaldi slouží nástroj `compute-eer`. Na závěr se pro každou metodu výpočtu míry podobnosti vypíše samostatné *EER*. Dosažené výsledky pro testování na otevřené množině jsou uvedeny v kapitole 3.4.

### 3.2.7 Implementace výsledného systému pro použití v praxi

V následujícím textu bude popsáno zprovoznění systému, který je určený pro praxi a který využívá zmíněný natrénovaný extraktor umístěný ve složce `exp/extractor`. Pro použití v praxi tak byla tato složka přenesena na server, na kterém běží výsledná aplikace *SREDEMO*. Více o vytvoření této aplikace v kapitole 4.




**Obrázek 7:** Schéma výsledného systému na rozpoznání řečníků pro použití v praxi

Implementace výsledného systému je opět provedena pomocí nástrojové sady Kaldi. Jedná se o vytvoření skriptu s podobnými kroky jako v trénovací úloze. Schéma výsledného skriptu je znázorněno na obrázku 7. Rozdíl oproti trénovací části spočívá v rozdělení na dvě úlohy, na úlohu zápisu řečníka do databáze a na úlohu rozpoznání testovaného řečníka. Dalším rozdílem je, že každá z těchto úloh není spouštěna na rozsáhlé databázi dat, ale jen na zpracování jednoho řečníka. Tedy

v úloze zápisu do databáze jsou skriptu předány informace jako argumenty ID nově zapsaného řečníka a typ úlohy `enroll` a skript zpracuje pouze nahrávky jedno řečníka. V případě úlohy rozpoznání řečníka jsou skriptu předány argumenty ID testovaného mluvčího a typ úlohy `test` a skript zpracuje pouze nahrávky jednoho neznámého řečníka a porovná je s již zpracovanými údaji zapsaných řečníků. Obě tyto úlohy počítají minimální množství informací v daný okamžik a dávají výsledek velmi rychle.

Prvním krokem, stejně jako v trénovací úloze, je příprava dat. Pro tento krok jsem napsal skript `SREDEMO_SPKR_prep.sh`, který připraví data pro zpracování v Kaldi. Obecně při práci s Kaldi je konvence, že příprava dat se provádí do složky `data`, která je ve složce projektu. V případě aplikace *SREDEMO* však z praktických důvodů toto muselo být změněno, protože by se tak míchala data jednotlivých řečníků v jedné složce. Z tohoto důvodu a z bezpečnostních důvodů jsou veškerá data řečníků v samostatných složkách mimo umístění Kaldi skriptů. Každý řečník má tak svoji složku, ve které mimo jiné se ukládají výstupy z implementace výsledného systému *SRE*. Více o struktuře databáze aplikace *SREDEMO* je v kapitole 4.3.3.

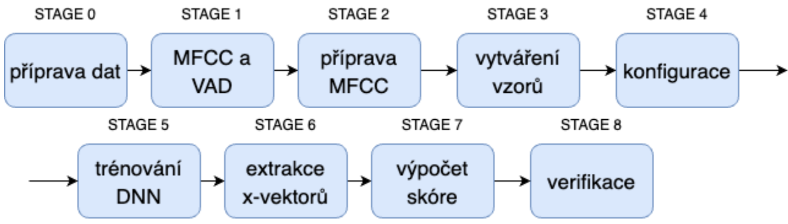
V druhém kroku jsou počítané *MFCC* koeficienty a *VAD* pomocí skriptů, které jsou součástí Kaldi. Opět výstupy jsou nastaveny tak, aby se ukládaly do samostatné složky daného řečníka a nemíchaly se tak s daty jiných mluvčích.

Extraktor je v tuto dobu již natrénovaný a tak dalším krokem skriptu je extrakce i-vektorů daného řečníka, které se ukládají opět do složky příslušící dané osobě. V případě typu úlohy zápis řečníka (`enroll`) tímto skript končí. V případě úlohy rozpoznání neznámého mluvčího (`test`) následuje ještě jeden poslední krok.

Pokud typ úlohy je nastaven jako `test`, proběhne čtvrtý krok, kterým je výpočet skóre. V tomto kroku se nejprve připraví seznam zapsaných řečníků do databáze včetně informace o umístění jejich reprezentujících i-vektorů a pak tzv. `trials` list, který popisuje dvojice testovaných řečníků. V tomto seznamu je tedy v jednom sloupci vždy ID testovaného mluvčího a v druhém sloupci se postupně střídají ID jednotlivých zapsaných řečníků. Pak následuje výpočet skóre a vyhodnocení. Celkový výsledek pro každého testovaného mluvčího se uloží v jeho příslušné složce do souboru `output`.

## 3.3 Implementace systému na bázi x-vektorů

Princip této úlohy je podobný jako v předchozí kapitole, avšak nejedná se o natrénování extraktoru, ale hluboké neuronové sítě, která je schopná z nahrávek daného řečníka extrahovat x-vektory, které následně budou sloužit pro rozpoznávání řečníků podobně jako i-vektory. Návrh tohoto systému je složen z 9 kroků (`STAGES 0 - 8`). Schéma postupu je znázorněno na obrázku 8. První dva kroky (příprava dat a výpočet *MFCC* a *VAD*) jsou totožné jako v případě implementace systému na bázi i-vektorů, viz kapitola 3.2, a proto v této části nejsou zmíněné. Ostatní kroky, které jsou u tohoto systému rozdílné, jsou postupně rozebrány. Na závěr v kapitole 3.4 je uvedeno vyhodnocení dosaženého systému a jeho porovnání se systémem na bázi i-vektorů.




**Obrázek 8:** Schéma postupu pro trénování x-vektor *DNN*

### 3.3.1 Příprava příznakových koeficientů pro trénování

Po přípravě dat a výpočtu *MFCC* a *VAD* tří souborů `train`, `enroll` a `test` je ještě potřeba připravit příznakové *MFCC* koeficienty na trénování neuronové sítě. Tím se zabývá tento třetí krok (`STAGE 2`). Úprava je však tentokrát prováděná pouze pro jeden soubor nahrávek `train`, protože pouze ten je použit pro trénování *DNN*.

První úpravou koeficientů je aplikace normalizace průměru a rozptylu kepstrálních koeficientů (*CMVN* - Cepstral Mean and Variance Normalization) a odstranění neřečových segmentů z *MFCC* koeficientů. Tuto úlohu provádí skript k tomu určený `prepare_feats_for_egs.sh`, který je součástí distribuce úlohy `v2` v Kaldi. Skript na základě informace z *VAD* a pomocí *CMVN* odstraní z *MFCC* koeficientů segmenty, které obsahují ticho. Výsledné příznakové koeficienty uloží do samostatné složky `train_no_sil` ve složce `exp`. K těmto nově vzniklým příznakovým koeficientům skript vytvoří i příslušný adresář ve složce `data` obsahující odpovídající soubory stejného typu jako z kroku přípravy dat. Výsledkem je, že dataset `train` má dvoje *MFCC* koeficienty, jednou obsahující ticho a jednou bez neřečových segmentů, a zabírá tak skoro dvojnásobek místa na disku.

Po odstranění neřečových segmentů je vhodná další úprava příznakových koeficientů. Nejprve se odstraňují takové příznaky, které po odstranění ticha jsou příliš krátké. Použitelné jsou pouze promluvy, které po odstranění ticha jsou stále delší než 5 sekund. Pokud je daná promluva kratší, měla by být vyňata z dat určených pro trénování *DNN*. Po odstranění nežádoucích promluv se provádí filtrování řečníků. Zde každý řečník by měl mít minimálně 8 promluv. Mluvčí, kteří mají po předchozích úpravách méně promluv, by měli být vyřazeni.

### 3.3.2 Vytváření vzorů pro trénování

Nyní, když jsou připravená data k trénování, nastává nejnáročnější proces, kterým je samotné trénování neuronové sítě. To realizuje skript `run_xvector.sh`, který je součástí receptu `v2`. Během trénovacího procesu je třeba provést mnoho náročných operací, které jsou rozdělené na tři oddělené kroky (`STAGES 3 - 5`), přičemž vlastní trénování neuronové sítě je v kroku `STAGE 5`. Tímto způsobem vývojář může snadno spouštět jednotlivé kroky (`STAGE`) a kontrolovat tak postup programu.

V prvním zmíněném kroku skript `get_egs.sh` ze složky `sid` vytváří trénovací příklady pro neuronovou síť, které jsou tvořeny částmi připravených promluv z předchozího kroku. V rámci této operace se vytvářejí archivy, které obsahují trénovací vzory. V každém archivu se vytvářejí všechny vzory se stejnou délkou, avšak v různých archivech mohou být různé délky vzorů. Přičemž v každém archivu se jeden mluvčí objevuje vícekrát. To se nastavuje parametrem `num-repeats`. V této práci byla zachována původní hodnota parametru 35.

Další nastavení parametrů tohoto skriptu je závislé na použité databázi nahrávek řečníků. Z tohoto důvodu autor skriptu doporučuje spouštět zmíněný krok vícekrát s různým nastavením a pozorovat, kolik archivů se vytvoří. Vhodný počet by měl být mezi 50 a 150 archivy. S původním nastavením v tomto kroku může vzniknout například pouze 1 archiv. Je tak potřeba snižovat parametr `frames_per_iter`, který představuje cílený počet segmentů dat v jednom archivu, což vede na navýšení archivů na požadované počty. Celá popsaná operace je prováděná pouze na výpočetních jádrech procesoru a lze urychlit paralelním výpočtem na více jádrech najednou.

Během spouštění této fáze se ukázalo, že skript `allocate_egs.py`, který je v tomto kroku volán, obsahuje chybu. Na fóru autor skriptu uvádí, že si je chyby vědom, ale že stále není v distribuci Kaldi opravená, a jak danou chybu opravit. Ve skriptu na řádku 262 se nachází cyklus `while`, který obsahuje špatnou podmínku pro trvání cyklu, která způsobuje, že program může běžet v nekonečném cyklu, ve kterém nic nevykonává. Po jednoduché modifikaci zmíněné podmínky začal skript pracovat správně, viz kód 2.

```
1 nutt_spkr = len(spk2utt[spkr])
2 break_loop = 0
3 while utt_len < length and break_loop < nutt_spkr:
4     utt = get_random_utt(spkr, spk2utt, length)
5     utt_len = utt2len[utt]
6     break_loop += 1
7 if break_loop < nutt_spkr:
8     offset = get_random_offset(utt_len, length)
9     this_egs.append( (utt, offset) )
```

**Kód 2:** Přidaný kód do skriptu `allocate_egs.py` na řádku 262 [9]

### 3.3.3 Trénování DNN

Další krok je označený jako `STAGE 4`, ve kterém se vytvářejí konfigurace neuronové sítě. Jedná se o samotné nastavení struktury *DNN*, které se ukládá do konfiguračních souborů ve složce `configs`, která je součástí cílové složky obsahující veškerá

data ohledně sítě umístěné v adresáři `exp` dle Kaldi konvence. Konfigurační soubory nejsou použité jen během trénovací fáze, ale i během provádění extrakce x-vektorů za pomocí natrénované *DNN*. V této práci bylo zachováno původní nastavení *DNN*, které je vidět v tabulce 1.

Posledním krokem (`STAGE 5`) ze skriptu `run_xvector.sh` je samotné trénování *DNN*. Tuto úlohu provádí skript `train_raw_dnn.py` ze složky `steps`. Trénování neuronové sítě je výpočetně velmi náročný proces, a proto je tato operace navržena tak, aby šla spustit na grafických kartách, které se na tento typ úloh více hodí než procesor. Výpočet je ovšem možný i na jádrech procesoru, avšak trvá pak mnohonásobně déle, i když je nastavený větší počet výpočetních jader parametrem `num-jobs-final`. Pro použití grafické karty k trénování *DNN* slouží parametr `use-gpu=True`, i zde je možná paralelizace mezi více grafickými kartami nastavením zmíněného parametru `num-jobs-final`. Původní nastavení tohoto parametru je 8, pokud však výpočet s tímto nastavením je spuštěn na menším množství grafických karet, přehltí se výpočetní paměť některé karty a program skončí s chybou, proto je nutné nastavit tento parametr na odpovídající počet grafických karet, které jsou dostupné pro trénování.

### 3.3.4 Extrakce x-vektorů

Nejsložitější část úlohy je již hotová. V předchozím kroku byla natrénována *DNN* pro extrakci x-vektorů. Nyní jen zbývá natrénovanou síť otestovat, stejně jako byl otestovaný extraktor i-vektorů. V tomto kroku (`STAGE 6`) se tedy provádí extrakce x-vektorů ze všech tří množin `train`, `enroll` a `test`.

Extrakce probíhá pomocí skriptu `extract_xvectors.sh`, který je součástí `nnet3` distribuce Kaldi ve složce `sid`. Tento skript je použit samostatně pro každou množinu na extrakci x-vektorů z vypočítaných *MFCC* příznaků jednotlivých promluv, které ukládá do podsložek příslušících jednotlivým množinám ve složce `exp/xvectors`. Skript může být spuštěn jak na procesoru, tak na grafické kartě, stejně jako skript pro trénování neuronové sítě, což se ovlivňuje parametrem `use-gpu=True`. Vzhledem k tomu, že na výsledném serveru bude aplikace provádět výpočty pouze na procesoru, extrakce je v tomto kroku také prováděna pouze na procesoru, aby se tak ověřila funkčnost pro výslednou aplikaci *SREDEMO*.

Po extrakci x-vektorů následují kroky s označením `STAGE 7` a `STAGE 8`, jedná se o výpočet skóre a verifikaci. Protože tyto kroky jsou totožné jako u systému využívajícího i-vektory, jsou v této kapitole vynechány.

### 3.3.5 Implementace výsledného systému v praxi

Výsledný x-vektorový systém (`v2`) v praxi je téměř shodný s implementací systému na základě i-vektorů (`v1`), která je popsaná v sekci 3.2.7. Změna schématu na obrázku 7 by byla pouze taková, že místo i-vektor extrakce by byla v případě `v2` extrakce x-vektorů.

Prvním krokem je opět příprava dat do samostatné složky řečníka a druhým krokem je výpočet *MFCC* a *VAD*. Tyto kroky jsou totožné jako ve skriptu `v1`. Za předpokladu, že při zápisu nového mluvčího vždy nejprve proběhne skript `v1` a s ním vytvoření i-vektorů reprezentující řečníka, je zbytečné tyto kroky provádět

znovu a mohla by rovnou po v1 následovat extrakce x-vektorů. Protože ale skripty v1 a v2 jsou koncipované tak, aby dle vstupních parametrů dokázaly provést zápis i rozpoznání, zmíněné kroky nejsou ve v2 vynechány. Při rozpoznávání se totiž spouští jen jeden ze skriptů.

Následuje třetí krok, ve kterém se provede extrakce x-vektorů daného mluvčího. Ty se ukládají do samostatného adresáře xvectors umístěného ve složce připadající mluvčímu, zatímco i-vektory jsou již v adresáři ivectors, pokud se provádí zápis řečníka. Stejně jako v případě skriptu v1, pokud je typ úlohy enroll, úloha zde končí. Pokud je typ zadán jako test, provede se finální krok, jímž je výpočet skóre a výsledné vyhodnocení. Tento krok je opět shodný jako v případě v1.

Obě úlohy (v1 a v2) mají realizovanou implementaci jako samostatné systémy, které jsou od sebe oddělené ve stejnojmenných adresářích v1 a v2. Takto je zachovaná konvence Kaldi a spouštění jednotlivých systémů se liší pouze ve změně cesty k jinému spouštěcímu souboru. Pro případ výsledné aplikace *SREDEMO* by bylo možné i řešení, ve kterém by existoval pouze jeden skript, který by prováděl jak extrakci i-vektorů, tak x-vektorů, včetně výsledného vyhodnocení. Tento přístup jsem ale zavrhl, protože mít oba systémy kompletní a od sebe oddělené je praktičtější, neboť mohou pracovat nezávisle na sobě. Obě implementace výsledných systémů včetně skriptů pro trénování extraktorů a *DNN* jsou dostupné v příloze této práce.

## 3.4 Experimentální část

V předchozích podkapitolách byl popsán proces trénování extraktoru i-vektorů a jeho verifikace a stejně tak implementace a testování systému na bázi x-vektorů. Po sestavení funkčních skriptů v prostředí Kaldi následuje experimentální část, ve které probíhá opakované trénování a testování natrénovaných extraktorů za účelem nalezení optimálního extraktoru společně s jeho detailním otestováním.

V experimentální fázi vývoje byla využita databáze *SPEECON* [10] jako základ potřebných dat jak pro trénování systémů, tak i pro výsledné vyhodnocení natrénovaných extraktorů. Pokud by se během takového postupu zjistilo, že na trénování je k dispozici nedostatečné množství dat, alternativní přístup by mohl být takový, že pro trénování se využije jiná databáze než pro otestování. Případně by bylo možné i spojit více databází do jednoho většího souboru. Více o použitých promluvách v následující sekci.

Vývoj a trénování pro oba popsané systémy bylo prováděno na výpočetních zdrojích MetaCentra poskytnutých projektem e-INFRA CZ (ID:90254). Jedná se o výpočetní gridovou infrastrukturu, která poskytuje velký paralelní výpočetní výkon. Díky tomu trénování probíhalo výrazně rychleji než na běžném počítači a nemusel být řešen ani objem dat na úložišti.

### 3.4.1 Použitá data pro trénování

Použitá databáze *SPEECON* obsahuje velké množství promluv u každého řečníka společně s textovým přepisem, protože je určená k použití při vývoji rozpoznávačů textů. Při vývoji systémů rozpoznávání řečníků je důležité mít k dispozici dostatečně velký počet různých řečníků, což databáze *SPEECON* poskytuje také. V rámci kroku přípravy dat se tak vybírá jen část promluv u každého mluvčího, na kterých probíhá jak vlastní trénovaní extraktoru, tak ověření výsledného systému.

Pro soubor řečníků `train` bylo použito 9 promluv obsahujících celé věty pro každého řečníka pro i-vektorový systém a pro x-vektorový systém bylo použito vět 40, přičemž průměrná celá věta obsahuje více než 5 sekund záznamu mluveného slova. Na těchto promluvách proběhlo natrénování extraktoru (resp. *DNN*) a jeho ověření pak proběhlo pomocí souborů `enroll` a `test` s rozdílnými promluvami oproti `train`, přičemž počet promluv v těchto druhých souborech se měnil v rámci experimentální fáze. Počty řečníků ve zmíněných dvou množinách jsou dále uvedeny v textu, přičemž soubor `test` vždy obsahuje shodné řečníky jako `enroll`, ale jiné promluvy, a další řečníky navíc, které `enroll` neobsahuje. Tímto způsobem bylo otestováno chování systémů v závislosti na celkovém množství promluv potřebných pro dobré fungování systému.

Ve vývoji x-vektorového systému v kroku `STAGE 2` bylo záměrně vynecháno filtrování dat za účelem odstranění kratších promluv a řečníků s malým počtem promluv. Důvodem je, že ve fázi přípravy dat byly vybrány z databáze *SPEECON* pouze dlouhé promluvy, které všechny měly dostačující délku. Zároveň, protože databáze *SPEECON* je obsáhlá, co se týče početnosti promluv u jednoho řečníka, mohlo se v prvním kroku (`STAGE 0`) nastavit velké množství promluv pro každého mluvčího. Tedy i druhá podmínka na minimum 8 promluv je zaručeně splněna.

### 3.4.2 Vyhodnocení i-vektor extraktoru

Natrénování extraktoru i-vektorů a jeho vyhodnocení proběhlo opakovaně několikrát za sebou. V jednotlivých pokusech se měnil počet promluv určených pro trénování a počet řečníků, respektive se měnily skupiny nahrávek, které jsou v databázi *SPEECON* označovány jako `ENVIROMENT`. Některé tyto skupiny jsou více zarušeny šumem a hlukem pozadí než jiné, které jsou naopak velmi čisté a nahrávány v klidném prostředí.

Po natrénování daného extraktoru i-vektorů bylo provedeno opakované spouštění extrakce i-vektorů s různým nastavením počtů promluv v množině zapsaných řečníků (`enroll`) a v množině určené k rozpoznání (`test`). Pro dané nastavení bylo provedeno vždy vyhodnocení chyby identifikace a *EER*.

V tabulce 2 jsou uvedeny výsledky vyhodnocení pouze extraktoru, který byl použit ve výsledné aplikaci *SREDEMO*. Těchto výsledků bylo dosaženo s extraktorem, který byl natrénován na 357 řečnících a každý řečník byl představován 9 promluvami. Použitá množina obsahovala jak čisté nahrávky, tak zašuměné nahrávky, díky čemuž je výsledný extraktor robustnější. Chyba identifikace pak byla vyhodnocována na 86 zapsaných řečnících a *EER* bylo vyhodnoceno na 196 řečnících v souboru `test`, počty jednotlivých promluv v souborech `enroll` a `test` jsou uvedené ve zmíněné tabulce.

| | | **chyba identifikace [%]** | | | **EER [%]** | | |
|---|---|---|---|---|---|---|---|
| **enroll** | **test** | **cos** | **LDA** | **PLDA** | **cos** | **LDA** | **PLDA** |
| 7 | 6 | 0 | 0 | 0 | 2,083 | 1,042 | 0 |
| 7 | 4 | 0 | 0 | 0 | 4,167 | 3,125 | 0 |
| 5 | 4 | 0 | 0 | 0 | 3,125 | 3,125 | 0 |
| 5 | 3 | 0 | 0 | 0 | 6,25 | 4,167 | 0 |
| 4 | 3 | 0 | 0 | 0 | 7,292 | 5,208 | 0 |
| 4 | 2 | 0 | 0 | 0 | 7,292 | 9,375 | 1,042 |

**Tabulka 2:** *SREDEMO* - Výsledky natrénovaného i-vektor extraktoru

| | | **chyba identifikace [%]** | | | **EER [%]** | | |
|---|---|---|---|---|---|---|---|
| **enroll** | **test** | **cos** | **LDA** | **PLDA** | **cos** | **LDA** | **PLDA** |
| 12 | 12 | 90 | 0 | 0 | 25 | 0 | 0 |
| 6 | 5 | 88 | 0 | 0 | 40 | 0 | 0 |
| 5 | 4 | 88 | 0 | 0 | 40 | 0 | 0 |
| 4 | 3 | 88 | 0 | 0 | 20 | 0 | 0 |
| 4 | 2 | 86 | 0 | 0 | 33 | 2,3 | 0 |
| 3 | 2 | 90 | 0 | 0 | 25 | 2,3 | 0 |

**Tabulka 3:** *SREDEMO* - Výsledky natrénované *DNN* extrahující x-vektory

Dle výsledků uvedených v tabulce 2 je možné konstatovat, že natrénovaný extraktor v kombinaci s libovolnou ze tří metod vyhodnocení rozpoznání (kosinová vzdálenost, *LDA*, *PLDA*) se skvěle hodí na identifikaci na uzavřené množině, neboť

všechny tři metody přinášejí chybu identifikace nulovou i pro nižší počty použitých promluv pro zápis a rozpoznávání.

V druhém vyhodnocení v pravé části tabulky je možné pozorovat, že kosinová vzdálenost dává nejhorší hodnoty *EER* a tedy nejhorší výsledky identifikace na otevřené množině, které se s klesajícím počtem promluv zhoršují, zatímco metoda *PLDA* dává perfektní výsledky ve většině případech.

### 3.4.3 Vyhodnocení DNN extrahující x-vektory

Opakované trénování a vyhodnocování *DNN* proběhlo shodným způsobem jako v případě extraktoru i-vektorů, viz předchozí sekce. Systém na bázi DNN je však výrazně složitější na natrénování a zároveň vyžaduje více dostupných dat k trénování. Podle literatury [7] jeho výhodou je však přesněji fungující systém *SRE*. Avšak pro shodné nastavení trénování jako pro extraktor i-vektorů dávalo *DNN* výrazně horší výsledky. Důvodem je pravděpodobně zmíněná náročnost na rozsáhlost dat. Z toho důvodu bylo výsledné *DNN* použité v aplikaci *SREDEMO* natrénováno na množině obsahující pouze čisté nahrávky ve formě 157 řečníků s 40 promluvami. Toto nastavení přineslo požadované výsledky, které byly vyhodnoceny také pouze na čistých nahrávkách, avšak tím se snížila robustnost systému vůči nekvalitním nahrávkám, což v této fázi experimentů nebylo zjištěno.

Z výsledků uvedených v tabulce 3 je patrné, že metodu vyhodnocení rozpoznávání kosinovou vzdálenost není možné používat v kombinaci s touto natrénovanou *DNN*. Chyba identifikace je pro tuto metodu extrémně velká i pro velké počty promluv a hodnoty *EER* jsou taktéž nepřijatelné, zatímco metody *LDA* a *PLDA* dávají nulovou chybu identifikace na uzavřené množině 43 řečníků v souboru `enroll` a vynikající hodnoty *EER* otestované na 86 řečnících v množině `test`.

### 3.4.4 Porovnání systémů

Dle výsledků uvedených v tabulce 2 a v tabulce 3 je možné říct, že se podařilo natrénovat dva dobře fungující systémy *SRE*, které dávají vynikající výsledky, pokud jsou použity v kombinaci s *PLDA*, přičemž systém na bázi *DNN* dle očekávání je lepší než systém na bázi *GMM*.

Nebyla ovšem provedena detailní analýza obou systémů, jak se chovají na různorodých datech, nebo trénování na spojení více databází do jedné. Tato práce by jistě mohla být obsahem celé jiné diplomové práce, avšak tato diplomová práce se zabývá vytvořením webové aplikace obsahující tyto systémy, a proto zde nezbývá prostor na detailnější analýzy a na trénování na dalších databázích. Ovšem tato práce přináší důležitý výsledek, kterým je fakt, že databáze *SPEECON* obsahuje dostatečné množství dat na trénování *DNN* za účelem vytvoření systému *SRE*.

# 4 Webová aplikace

V této kapitole je popsán postup návrhu webové aplikace. Často je vidět v technické praxi podobný postup řešení složitých problémů, a to pomocí rozdělení na několik méně složitých podproblémů. V případě webových aplikací je základní rozdělení aplikace na frontend a backend. Pro grafický návrh frontendu aplikace jsem si vybral framework `React`, protože je to v současnosti často používaný systém, který umí efektivně vykreslovat uživatelské rozhraní (*UI* - User Interface). Pro backend část aplikace jsem si vybral framework `Django`, protože se často užívá v kombinaci s `Reactem` a protože se programuje v jazyce `Python`. Ten je uživatelsky jednoduchý a hodí se i pro spouštění skriptů knihovny Kaldi starající se o SRE systém.

## 4.1 Backend

Ačkoliv frontend je ta část aplikace, se kterou uživatel interaguje, je důležité začít práci od backendu, což je ta část aplikace, kterou běžný uživatel přímo nevidí, ale je samotným jádrem aplikace, které vykonává požadavky včetně odesílání frontendu uživateli. Často se skládá ze tří hlavních částí: serveru, aplikace a databáze. Pro backend technologie se často používají programovací jazyky jako jsou `PHP`, `Ruby`, `Python` a další [11]. Avšak vytvářet celý backend v čistém programovacím jazyce by bylo velice zdlouhavé, a proto samotný vývoj může být značně urychlen pomocí použití dostupných knihoven, které obsahují většinu již hotových potřebných nástrojů. Příkladem může být framework `Django`.

### 4.1.1 Django

V roce 2005 byla publikovaná první veřejná verze frameworku `Django` a pojmenovaná po belgicko-francouzském kytaristovi Django Reinhardtovi. V současnosti tento framework má přes 1000 přispěvatelů a více jak 3000 balíčků, které jsou určeny přímo pro práci s `Djangem`. Již od původní verze `Django` využívá modelově-pohledové řízení (*MVC* - Model-View-Controller) pro práci s relačními databázemi a stává se častou volbou při vývoji webových aplikací celé řady organizací jako je *Instagram*, *Pinterest*, *National Geographic* a dalších [12].

Výslednou aplikaci SREDEMO založenou na frameworku `Django` jsem vytvářel na svém počítači s Windows 10 a následně jsem ji přenesl na Linux server. Vývoj v rámci Windowsu nebo Linuxu je velmi podobný a liší se maximálně v pár příkazech při instalaci. Pro vývoj i pro nasazení aplikace bylo potřeba kromě jazyku `Python` nainstalovat na počítače framework `Django`, který vyžaduje, aby spuštěná aplikace, která je vytvořená pomocí této knihovny, byla oddělená od systému počítače. Toho je docíleno pomocí spuštění ve virtuálním prostředí `Pythonu`. Toto virtuální prostředí tak bylo také potřeba nainstalovat.

```
# instalace balicku virtualnich prostredi
sudo apt install python3.11-venv
# vytvoreni virtualniho prostredi
sudo python3 -m venv sreenv
# aktivace virtualiho prostredi
source sreenv/bin/activate
# instalace balicku pokud neni dostatecne opravneni
sudo /var/.../sreenv/bin/python -m pip install -r Speakly/requirements.txt
```

**Kód 3:** Instalace frameworku `Django` na Linux server

Příkazy v kódu 3 popisují instalaci na serveru Linux Debian 12, kde již byl předinstalovaný `Python 3.11.2` a nebylo nutno ho tedy instalovat. Následovala instalace virtuálního prostředí a vytvoření virtuálního prostředí s názvem `sreenv`. Nakonec do tohoto virtuálního prostředí byly nainstalované knihovny související s frameworkem `Django` ze souboru `requirements.txt`. Obsah tohoto souboru lze vidět v kódu 4.

```
asgiref==3.7.2
Django==4.2.6
django-cors-headers==4.3.0
djangorestframework==3.14.0
pytz==2023.3.post1
sqlparse==0.4.4
tzdata==2023.3
```

**Kód 4:** Obsah souboru `requirements.txt`

### 4.1.2 Vytvoření aplikace SREDEMO

Samotná aplikace je vytvořena jako jeden `Django` projekt s názvem `Speakly`, který hraje roli pouze v adresářové struktuře. Projekt obsahuje dvě části, které `Django` označuje jako aplikace. První aplikace s názvem `frontend` se stará o vykreslování frontendu na straně uživatele a druhá aplikace `api` se stará o backend na straně serveru. Následující kód 5 popisuje vytvoření projektu se zmíněnými aplikacemi. V aplikaci `frontend` je pak v následujícím kroku vytvořen `React` projekt. Více o `Reactu` v kapitole 4.2.

```
# vytvoreni projektu s nazvem Speakly
django-admin startproject Speakly
# vytvoreni aplikace s nazvem api
python manage.py startapp api
# vytvoreni aplikace s nazvem frontend
python manage.py startapp frontend
```

**Kód 5:** Vytvoření `Django` projektu a aplikací `frontend` a `api`

Po zadání příkazů z kódu 5 se vytvoří prázdný projekt obsahující připravené soubory pro rychlý začátek vývoje celé aplikace. Jeden z hlavních souborů projektu je soubor `settings.py`, ve kterém vývojář definuje většinu nastavení aplikace. V rámci aplikace *SREDEMO* tak bylo potřeba nastavit propojení s dvěma vytvořenými aplikacemi (`frontend` a `api`), URL adresy pro statické soubory a k složce `media` (tam

se ukládají přijaté soubory), cesty k systémovým složkám projektu, cestu k databázovému souboru typu `SQLite3` a další nastavení aplikace. Avšak jedno z nejdůležitějších nastavení je nastavení `DEBUG`.

Tento parametr ovlivňuje chování frameworku `Django` a vypisování chybových zpráv. Je určený pro snadnější hledání chyb při lokálním testování, ale představuje velký bezpečnostní problém při produkčním nasazení. Je tedy jisté, že při lokálním vývoji aplikace je toto nastavení povolené a při produkčním nasazení vždy zakázané. Toho lze využít pro zjednodušení přenastavování projektu při nasazení na produkční server. Proto jsem řadu parametrů v projektu nastavoval ve dvojicích, ze kterých se vždy vybere daný parametr dvojice podle zmíněné proměnné `DEBUG`. Takto při nasazování na server není třeba měnit všechny parametry jednotlivě, ale stačí globálně zakázat `DEBUG`. Mezi zmíněné parametry patří například cesty k souborům, URL adresy, nebo i kusy kódů, které mohou být spouštěny jen v produkčním nasazení na serveru.

### 4.1.3 Modely

Framework `Django` řeší na pozadí chod aplikace, databázi a zabezpečení. Při vytváření malých projektů, jako je aplikace *SREDEMO*, je práce vývojáře zredukovaná na práci na tzv. modelech a pohledech. Každá aplikace v `Django` projektu má pak své vlastní soubory `models.py` a `views.py` pro definování modelů a pohledů.

```
1  class Record(models.Model):
2      spkr_id = models.CharField(max_length=8, unique=True, default=gen_spkr_id)
3      name = models.CharField(max_length=50)
4      location = models.CharField(max_length=50)
5      gender = models.CharField(max_length=30)
6      age = models.PositiveIntegerField(default=0)
7      recorded_sentence = models.CharField(max_length=200)
8      recorded_file = models.FileField(upload_to=get_upload_path)
9      created_at = models.DateTimeField(auto_now_add=True)
10     class Meta:
11         db_table='Record'
```

**Kód 6:** Zjednodušený příklad modelu v souboru `models.py` pro uložení informací o řečníkovi

V souboru `models.py` se definují jednotlivé modely zápisů do databáze `SQLite3`. Jsou tak nezbytné pro předávání a ukládání informací. Každý model může mít mnoho atributů jako je jméno, pohlaví, věk, zvukové soubory a další. Zároveň se zde nastavují pravidla pro jednotlivé atributy jako je maximální délka, unikátnost, místo uložení souboru a další.

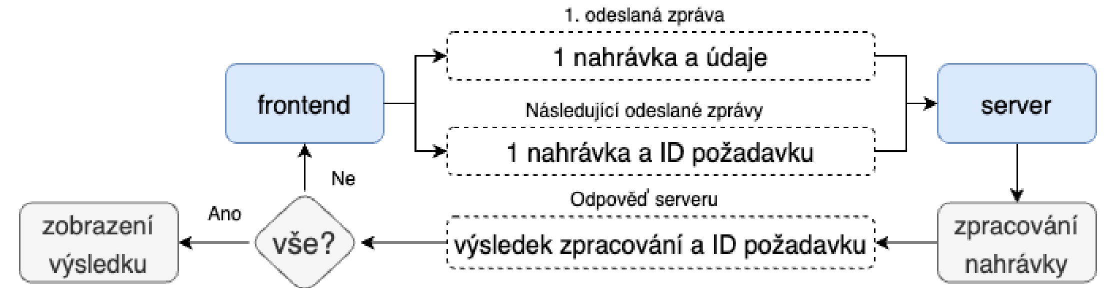
Podobná struktura modelu, jako je v kódu 6, byla použita v aplikaci `api` pro práci s daty souvisejícími se zápisem řečníků do databáze. Dále `api` obsahuje další model, který pracuje s daty týkajícími se úlohy rozpoznávání řečníků. Tedy v aplikaci `api` jsou dva modely pro dvě různé operace, zatímco aplikace `frontend` nepracuje s žádnými daty, a proto neobsahuje žádné modely.

Součástí návrhů modelů může být i práce na unikátních funkcích, které vykonávají nějaké úkony s daty, které jsou specifické pro danou aplikaci. V aplikaci *SREDEMO* je tak implementována jedna funkce `gen_spkr_id()`, která při vytvá-

ření nového zápisu do databáze vygeneruje unikátní ID řečníka, které má náhodný charakter. Při generování tohoto ID se tak kontroluje, aby nikdo z mluvčích, kteří jsou zapsaní v databázi, toto ID neměl a zároveň se kontroluje, aby ani žádný zápis v databázi ve složce `deleted` (viz kapitola 4.3.3) nebyl shodný.

### 4.1.4 Pohledy

V pohledech se definuje, jak se má přistupovat k datům a jak se mají reprezentovat na základě požadavků od uživatele. Pohled aplikace `frontend` je velmi jednoduchý a obsahuje pouze příkaz pro předání souboru `index.html`, protože tato aplikace zajišťuje jen jeden úkol, kterým je předání statických souborů pro vykreslení *UI* uživateli. Tyto statické soubory jsou vygenerovány frameworkem `React`. Pohled aplikace `api` je však mnohem složitější.




**Obrázek 9:** Schéma nahrávání audio souborů na server, komunikace začíná blokem frontend

Soubor `views.py` aplikace `api` obsahuje několik pohledů. Jeden z nich je například pohled na zápis nového řečníka do databáze. Tento pohled definuje, co se má stát v případě, že přijde požadavek na uložení informací do databáze od *UI*. Prvním krokem je vždy zkontrolování, zda jsou příchozí data validní, tedy že jejich struktura odpovídá struktuře modelu. Tento krok představuje zabezpečení pro příchozí požadavky a `Django` ho vynucuje tak, že pokud validace není provedena, není umožněn zápis informací. Pak následuje uložení přijatých informací do databáze společně s vytvořením unikátního ID řečníka. Dále se provede uložení přijatých audio souborů do samostatné složky příslušící danému řečníkovi. Pokud přijaté audio soubory mají vzorkovací kmitočet vyšší než 16 kHz, provede se převzorkování na tuto vzorkovací frekvenci. Tento proces je implementován pomocí knihovny `pydub`, která obsahuje pokročilé nástroje pro práci se zvukovými soubory. Následuje vytvoření textových souborů obsahující informace o nahrávkách a dané osobě. Stejné informace jsou uloženy i v databázi aplikace. Tímto dvojím způsobem jsou informace ukládány tak, že výsledná databáze při jejím přenosu nevyžaduje extrahování informací z aplikace. Posledním krokem je spuštění BASH skriptů nástroje Kaldi, které provedou výpočet i-vektorů (nebo x-vektorů). Na závěr pohled vrátí informaci frontendu, jestli se zápis povedl, či nikoliv.

V případě požadavku na rozpoznání řečníka je struktura pohledu velmi podobná. Hlavní rozdíl mezi zápisem a rozpoznáním řečníka je přímo implementován ve skriptu využívajícím nástroje Kaldi a tedy v pohledu je jedinou rozdílností to, jak se skript volá. Zbytek pohledu je principiálně shodný s předchozím. Jedná se

o zkontrolování a uložení přijatých souborů do správné složky. Na konci však pohled vrátí frontendu výsledek rozpoznávání.

Při provádění požadavku na nový zápis do databáze nebo na rozpoznání mluvčího se musí v daném požadavku přenést všechny audio nahrávky z frontendu na backend. Servery však mají nastavenou nějakou maximální hodnotu objemu dat pro jeden požadavek. Když uživatel nahraje více delších promluv, které dohromady překročí tento limit, není možné je pak jedním požadavkem odeslat na backend. Z tohoto důvodu jsou oba pohledy pro zápis mluvčího i pro rozpoznání řečníka implementovány tak, že po zpracování daného požadavku vrátí frontendu unikátní ID, pod kterým byl požadavek zpracován. Pokud pak frontend pošle další požadavek obsahující další nahrávky společně s tímto unikátním ID, backend přidá nahrávky do předchozího požadavku. Maximálně takto může dojít k deseti požadavkům za sebou, které obsahují samostatné nahrávky, protože na straně frontendu je nastaveno jako maximální množství nahraných audio stop deset. Schéma popsané komunikace je vidět na obrázku 9.

```
class NewSentenceView(APIView):
    def post(self, request):
        sentence_data = JSONParser().parse(request)
        sen_file_path = os.path.join(settings.MEDIA_ROOT,'sentences')
        if sen_file_path and os.path.isfile(sen_file_path):
            with open(sen_file_path, 'a', encoding='utf-8') as file:
                file.write(sentence_data['newSentence'] + '\n')
            return JsonResponse(status=status.HTTP_201_CREATED)
        else:
            return JsonResponse({'error': 'Sentences not found.'}, status=404)

    def delete(self, request):
        delete_data = JSONParser().parse(request)
        line_number = delete_data.get('line_to_delete')
        sen_file_path = os.path.join(settings.MEDIA_ROOT,'sentences')
        temp_file_path = sen_file_path + '.tmp'
        if sen_file_path and os.path.isfile(sen_file_path):
            with open(sen_file_path, 'r', encoding='utf-8') as original_file:
                with open(temp_file_path, 'w', encoding='utf-8') as temp_file:
                    for i, line in enumerate(original_file, start=1):
                        if i != line_number:
                            temp_file.write(line)
            shutil.move(temp_file_path, sentences_file_path)
            return JsonResponse( status=200)
        else:
            return JsonResponse( status=404)
```

**Kód 7:** Pohled pro přidávání nových vět a mazání vět v administrativní části aplikace

Kromě pohledů na zápis a rozpoznání řečníka, které jsou poměrně komplikované, jsou implementovány i další jednoduché pohledy. Pohled `FeedbackView` slouží k zachycení zpětné vazby uživatele po dokončení rozpoznání řečníka, zda ho systém správně rozpoznal. V kódu 7 je vidět pohled `NewSentenceView`, který obsahuje dvě metody pro mazání a přidávání nových vět v administrativní části aplikace, kde dále pohled `ShowListView`, který také obsahuje dvě metody, umožňuje nahlédnutí a mazání v databázi. Přístup do administrativní části aplikace povoluje jednoduchý

pohled `LoginView`, který kontroluje, že přihlašovací údaje v požadavku jsou shodné s údaji uloženými v systému aplikace.

`RandomSentenceView` je dalším jednoduchým pohledem, který na základě požadavku frontendu vrátí náhodně vybranou větu z připraveného seznamu. Tento pohled je jedním z dalších příkladů, které ukazují, že frontend by měl pouze data vykreslovat, ale žádná nepřenášet. Data do frontendu by měl vždy dodávat backend. V případě aplikace *SREDEMO* roli backendu zastává aplikace `api`. Tento pohled je vidět v kódu 8.

```
1  def RandomSentenceView(request):
2      sentences_file_path = finders.find('data/sentences')
3      if sentences_file_path:
4          with open(sentences_file_path, 'r', encoding='utf-8') as file:
5              sentences = file.readlines()
6          # Vyber nahodnou vetu
7          random_sentence = random.choice(sentences)
8          # Odesli nahodnou vetu jako JSON
9          return JsonResponse({'sentence': random_sentence.strip()})
10     else:
11         return JsonResponse({'error': 'Sentences not found.'}, status=404)
```

**Kód 8:** Pohled pro odesílání náhodných vět na základě požadavku

## 4.2 Frontend

V kontextu webové aplikace se považuje za frontend ta část aplikace, kterou uživatel vidí a interaguje s ní. Příkladem může být menu, kontaktní formulář a v případě této práce se jedná o rozhraní pro nahrávání a odesílání audio nahrávek na server. Pro návrh a vývoj frontend webového uživatelského rozhraní se používají různé nástroje a technologie jako je například kombinace `html`, `css` a `JavaScript` [11]. Stejně jako v případě backendu existují knihovny, které obsahují připravené nástroje a které značně urychlují proces vývoje. Příkladem takové knihovny je `React`.

### 4.2.1 React

`React` je framework, který vyvinula tehdejší společnost *Facebook* dnes známá jako *Meta*. Jedná se o velmi oblíbenou knihovnu napsanou v jazyce `JavaScript` používanou na vytváření interaktivního *UI* ve webových aplikacích. Nástrojové knihovny určené pro frontend, které jsou založené na `JavaScriptu`, poskytují rozdělení na dílčí části, které se nazývají komponenty. Dochází tak k abstrakci širšího problému na několik užších. Jednotlivé části aplikace lze tak rozdělit na jednotlivé elementy, které je možné následně opakovaně používat na jiných částech aplikace nebo dokonce v jiných aplikacích [13].

### 4.2.2 Vytvoření uživatelského rozhraní

Na vytvoření *UI* pro aplikaci *SREDEMO* jsem využil framework `React`. Samotný projekt v `Reactu` je součástí `Django` aplikace `frontend`. Tato aplikace je nastavená tak, že dodává uživateli v podstatě prázdný soubor `index.html` a skript `main.js`, který na straně uživatele vykreslí celou stránku. Tento skript nepřenáší žádná data a při vykreslování si o data žádá zpět na server. Příkladem dalších dat jsou foneticky bohaté věty, které mají řečníci předčítat. Tento přístup umožňuje snadnější správu dat a rychlejší načítání webové stránky.

Při práci na *UI* jsem využil externí knihovnu `Bootstrap` [14], která dodává zjednodušení vytváření grafického designu, a knihovnu `Wavesurfer` [15], která jednoduše řeší vykreslování a přehrávání audio signálů. Samotná práce na *UI* se skládá z vytváření komponentů v jazyce `JavaScript`. Komponenty jsou většinou tvořeny jako funkce, které mají nějaké vlastní stavy a metody, které je možné volat, případně je lze sestavit i jako třídy, ale to je méně časté. Každá tato funkce vrací kód ve formátu `JSX`, což je zápis podobný `html`. Řazením jednotlivých komponentů do sebe je tak možné jednoduše a přehledně vytvořit celou stránku.

Jednotlivé soubory komponentů se při provozu v produkci nepřenášejí. Při dokončení vývoje se z nich vygeneruje jeden výsledný `JavaScript` soubor `main.js`, který se tak stane statickým souborem. To znamená, že se po dokončení vývoje již dál nemodifikuje a webový server podle toho s ním tak může nakládat. V případě aplikace *SREDEMO* framework `Django` jako reakci na žádost o zobrazení stránky vrátí tento `main.js` soubor spolu se zmíněným `index.html` souborem a spolu s dalšími potřebnými soubory, jako jsou například `CSS` soubory, obrázky a další.

Podobně jako u frameworku `Django` i `React` má při vývoji aplikace verzi kódu typu debug. Ta se u `Reactu` nazývá Development a opět se jedná o verzi kódu,

která není určená k nasazení do produkce, přičemž její nasazení by mohlo znamenat vážné bezpečnostní ohrožení. Vygenerovaný `main.js` typu Development kromě výrazně větší velikosti i obsahuje řadu chybových hlášení, aby vývojář mohl snadno debugovat kód. Na produkci je třeba nechat `React` vygenerovat verzi Build, která tyto pomocné kódy neobsahuje a která je i zároveň výrazně menší, což urychluje načtení stránky.

### 4.2.3 Nahrávání zvukových nahrávek

Pro samotné nahrávání audio signálů jsem použil knihovnu `recorder-js` [16]. `React` sám o sobě obsahuje instanci `mediaRecorder`, která umí nahrávat zvuk. Ovšem formát zvuku může být závislý na použitém prohlížeči a většinou se jedná o formát `webm` (`codec opus`), zatímco tato knihovna umí nahrávat audio ve formátu `wav`, což je pro aplikaci *SREDEMO* nezbytné. Tímto způsobem nebylo potřeba implementovat nahrávání do formátu `wav` speciálně pro aplikaci *SREDEMO*. V kódu 9 je vidět inicializace projektu v `Reactu` a nainstalovaní zmíněných knihoven společně se základními knihovnami `Reactu`.

```
1 npm init -y
2 npm install react-router-dom
3 npm i react react-dom --save-dev
4 npm i prop-types
5 npm install bootstrap jquery --save
6 npm i recorder-js
7 npm i wavesurfer.js
```

**Kód 9:** Založení projektu `Reactu` a nainstalování knihoven

Analýza pomocí Kaldi nástrojů na straně serveru pracuje s nahrávkami, které mají vzorkovací kmitočet 16 kHz. Bylo by vhodné rovnou s touto vzorkovací frekvencí nahrávat audio stopy. Problém je, že některé prohlížeče nepodporují změnu vzorkovacího kmitočtu oproti základnímu stejným způsobem, což vede k složité realizaci, bez které by stránka některým uživatelům nahrávání neumožňovala, nebo by nahrané audio mělo neznámý vzorkovací kmitočet. Ačkoliv by nahrávání přímo se vzorkovací frekvencí 16 kHz přineslo i výhodu, že by soubory ve většině případů byly menší než s původní vzorkovací frekvencí a rychleji by se tak přenášely internetem, muselo být nahrávání v aplikaci *SREDEMO* implementováno se základní vzorkovací frekvencí hardwaru, která může být pro různé uživatele různá, a na straně serveru se po přijetí souboru provádí převzorkování na 16 kHz. Většina spotřební elektroniky má základní vzorkovací kmitočet dán jako celočíselný násobek 8 kHz. Převzorkování na 16 kHz tak nezpůsobuje značné zkreslení původního signálu.

Nahrané soubory obsahující jednotlivé promluvy uživatele dohromady mohou překračovat limit objemu dat, který jsou servery schopné zpracovat jedním požadavkem. Jak již bylo zmíněno v kapitole 4.1.4, komunikace mezi frontendem a serverem probíhá tak, že v cyklu jsou odesílány požadavky obsahující vždy jednu nahrávku, přičemž její maximální délka je při nahrávání omezena na 12 sekund. Na základě unikátního ID požadavku, který předává server, je zaručena identifikace probíhající komunikace. Po úspěšném nahrání na server a zpracování všech promluv frontend zobrazí výsledek. Schéma popsané komunikace je vidět na obrázku 9.

## 4.3 Nasazení na server

Jak již bylo zmíněno, aplikace *SREDEMO* po dokončení vývoje byla nasazena na server s operačním systémem Linux Debian 12. Ačkoliv použitý framework `Django` se stará o zabezpečení a chod aplikace jako takové, neobsahuje webový server, který by přímo plnil požadavky uživatelů. Tento fakt není omezením, ale možností použít libovolný webový server, který umí svoji práci dělat kvalitně. Jako webový server jsem využil program `Apache2`, který na serveru již byl předinstalovaný.

### 4.3.1 Apache2

Propojení `Django` aplikace s webovým serverem `Apache2` je provedeno pomocí *WSGI* modulu. *WSGI* (Web Server Gateway Interface) je standardní rozhraní pro komunikaci mezi webovým serverem a aplikací napsanou v jazyce `Python`. Tímto způsobem je umožněn efektivní běh webové aplikace vytvořené ve frameworku `Django`. Čistý `Apache2` neobsahuje modul pro *WSGI*, jeho instalace je však snadná, viz kód 10.

```
sudo apt-get install libapache2-mod-wsgi-py3
sudo a2enmod wsgi
```

**Kód 10:** Instalace a spuštění *WSGI* modulu pro `Apache2`

`Apache2` bylo pro použití zmíněného modulu v kombinaci s aplikací napsanou ve frameworku `Django` potřeba nakonfigurovat. V rámci konfigurace bylo třeba uvést cesty k virtuálnímu prostředí `Pythonu` (`sreenv`) pro spouštění `Django` aplikace, spolu s cestou ke spouštěcímu *WSGI* souboru aplikace. Dále byl do konfigurace přidán i alias pro *WSGI* proces, protože aplikace *SREDEMO* bude na webové podadrese `/sredemo`. Toto nastavení způsobuje, že se server pro všechny adresy chová normálně, tedy předává statické soubory z nastavených složek, ale pro adresu `/sredemo` přesměrovává požadavky na `Django` aplikaci. To otevírá dveře do budoucnosti dalším aplikacím, které by mohly běžet na stejné doméně s jinou cestou na konci adresy. Tyto další aplikace by pak mohly být nakonfigurovány podobně, ale s jiným aliasem.

```
# cesty ke spusteni python programu - Django aplikace
WSGIDaemonProcess SREDEMO python-home=.../sreenv python-path=.../Speakly
WSGIProcessGroup SREDEMO
WSGIScriptAlias /sredemo /var/.../Speakly/wsgi.py # URL aplikace SREDEMO
# umisteni spousteciho wsgi souboru DJango aplikace
<Directory /var/.../Speakly>
    <Files wsgi.py>
        Require all granted
    </Files>
</Directory>
# nastaveni weboveho aliasu pro slozky media a static
Alias /sredemo/media/ /var/.../Speakly/media/
Alias /sredemo/static/ /var/.../Speakly/static/
# prava pro pristup do slozek
<Directory /var/.../Speakly>
    Require all granted
</Directory>
```

**Kód 11:** Přidaná konfigurace `Apache2`

Dále se webová aplikace *SREDEMO* odkazuje URL adresou na složku se statickými soubory a na složku media, kam se ukládají data. V konfiguraci bylo potřeba pro tyto dvě cesty vytvořit aliasy, aby web server správně vracel odpovědi na požadavky s těmito adresami spojené. Protože tyto cesty na serveru reálně neexistují. Další adresování v rámci webové aplikace řeší samotné `Django`. Celá tato konfigurace je vidět v kódu 11.

### 4.3.2 Docker

Jak již bylo zmíněno v kapitole 3.1, využití `Dockeru` pro spouštění nástrojů Kaldi je výrazně jednodušší, než provádět kompletní lokální instalaci na počítači. Proto jsem aplikaci *SREDEMO* navrhl tak, že když je potřeba udělat výpočet pomocí Kaldi, tak se spustí kontejner s touto knihovnou a výpočty se provedou v něm. Na konci operace se kontejner opět uspí. V kódu 12 je popsána instalace `Dockeru`, stažení obrazu Kaldi a vytvoření kontejneru.

```
1 # instalace Dockeru
2 curl -fsSL https://get.docker.com -o get-docker.sh
3 sudo sh get-docker.sh
4 # stazeni Kaldi image z Docker hubu
5 docker pull kaldiasr/kaldi
6 # spusteni kontejneru s nazvem SREDEMO... s pristupem do slozky /export/sredemo
7 docker run -it --name SREDEMO_kaldi_kontejner -v /export/sredemo:/export/sredemo
    kaldiasr/kaldi:latest
```

**Kód 12:** Instalace `Dockeru` a spuštění kontejneru

V případě, že uživatel na frondendu aplikace nahraje své promluvy a odešle je na server, `Apache2` tento požadavek předá `Django` aplikaci, která uloží potřebné informace do databáze, jak je popsáno v kapitole 4.1. Posledním krokem aplikace je spuštění BASH skriptu s výpočtem využívajícím Kaldi jako subproces v `Pythonu`. Příklad spuštění je znázorněn v kódu 13.

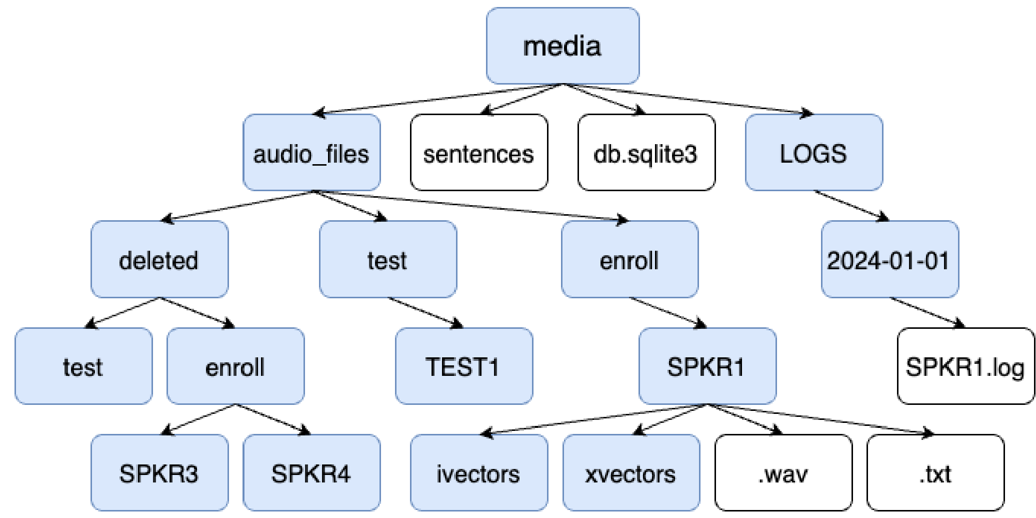
Tento skript nejprve spustí kontejner s Kaldi a v něm následně provede výpočet i-vektorů (případně x-vektorů) z nahrávek daného řečníka tak, jak bylo popsáno v sekci 3.2.7 (případně 3.3.5). Během běhu skriptu jsou jednotlivé kroky zapisovány do logu s případnými chybami. Pro každého mluvčího je vytvářený samostatný log pro zachování větší přehlednosti. Po ukončení subprocesu se kontejner uspí a `Django` aplikace vrátí informaci přes `Apache2`, zda se celá operace povedla či nikoliv a případný výsledek rozpoznání řečníka.

```
1 try:
2     subprocess.run(["../sre/v1/main_run.sh", "enroll", spkr_id], check=True)
3     logging.info("Kaldi ivecor enroll successful")
4 except subprocess.CalledProcessError as e:
5     logging.error("Kaldi process error: " + e)
```

**Kód 13:** Příklad spuštění i-vektorového systému v jazyce `Python`

### 4.3.3 Struktura databáze

Součástí práce na aplikaci *SREDEMO* je i návrh vhodné struktury databáze, do které se ukládají zpracované zvukové nahrávky. Veškeré soubory a data, ke kterým má `Django` přístup, jsou uložená v adresáři, který `Django` označuje `media` a který, jak je zvykem, je umístěný ve struktuře `Django` projektu. V této práci je ve skutečnosti adresář `media` umístěn na jiném disku a ve struktuře projektu je nahrazen odkazem. Struktura složky `media` je znázorněna na obrázku 10.




**Obrázek 10:** *SREDEMO* - Schéma struktury databáze

V adresáři media jsou umístěné soubory `sentences`, který obsahuje promluvy, které mají řečníci předčítat, a databázový soubor typu `SQLite3`, do kterého zapisuje `Django` všechny zpracované informace. Dále se tu nachází složka `LOGS`, do které se ukládají logy o požadavcích na `Djnago`, a složka `audio_files`, která uchovává celou databázi zapsaných mluvčích i jednotlivé požadavky o rozpoznání mluvčího. Tyto dvě skupiny jsou na obrázku 10 označeny `enroll` a `test`.

Dále je zde uchovávaná i třetí skupina nahrávek `deleted`. Pokud administrátor aplikace *SREDEMO* smaže nějaký zápis v databázi, doopravdy se smažou data pouze v souboru `db.sqlite3`. Veškeré soubory připadající k danému zápisu se pouze přesunou ze složky `enroll` nebo `test` do adresáře `deleted`. Při rozpoznávání mluvčího pomocí Kaldi nástrojů se jako zapsaní řečníci berou pouze zápisy, které jsou ve složce `enroll`. Data, která tedy jsou přesunutá do složky `deleted`, neovlivňují výsledek úlohy rozpoznávání. Přesunutím do separátní složky místo úplného smazání je navíc zajištěno, že omylem smazaná data nebudou ztracená.

Každý zapsaný mluvčí v databázi má ve složce `enroll` svou vlastní složku pojmenovanou podle svého unikátního ID. Tato složka obsahuje veškeré informace, které k danému řečníkovi patří. Přímo ve složce jsou ukládány jednotlivé audio soubory ve formátu `wav` společně s textovými soubory, které obsahují slovní přepis promluvy společně se všemi dostupnými informacemi o daném mluvčím. Ukázka takového přepisu je vidět v kódu 14. Dále jsou ve složce uchovávána veškerá data spojená s Kaldi zpracováním. Jedná se tedy o složky `data`, `mfcc`, `vad` a hlavně složky `ivectors` a `xvectors`.

```
1      Speaker info
2 NAM: Marek Vavrinek
3 SID: SPEN5361
4 AGE: 24
5 SEX: Male
6 LOC: Praha
7      Audio file info
8 FIL: SPEN5361S0.wav
9 SEN: Koukat se na rozbitou televizi bylo jako poslouchat ticho.
10 DAT: 2024-04-08 11:54:21
```

**Kód 14:** Výpis textového souboru, který obsahuje přepis promluvy společně s informacemi o řečníkovi.

## 4.4 Vyhodnocení funkčnosti aplikace

Po trénování extraktorů v prostředí Kaldi a jejich otestování a po sestavení a nasazení funkční aplikace *SREDEMO* do produkčního on-line prostředí přichází konečná fáze ve vývoji a tou je otestování a vyhodnocení funkčnosti celkového systému na reálných uživatelích. V této fázi se testuje, jestli webová aplikace funguje správně všem uživatelům, kteří mohou používat libovolné webové prohlížeče a libovolný hardware, a zda natrénované systémy na bázi Kaldi pracují správně i s reálnými nahrávkami, které mohou být nahrávány v rozdílných prostředích rozdílnou zvukovou technikou.

### 4.4.1 Popis testování aplikace

V rámci testování výsledného systému byla aplikace zveřejněná na dostupné URL adrese, která byla rozeslána dobrovolníkům, kteří souhlasili, že pravdivě odpoví, zda jim systém fungoval správně, či nikoliv. Uživatelé byli požádáni, aby nahráli pomocí vyvinuté aplikace promluvy v libovolném prostředí na svých zařízeních.

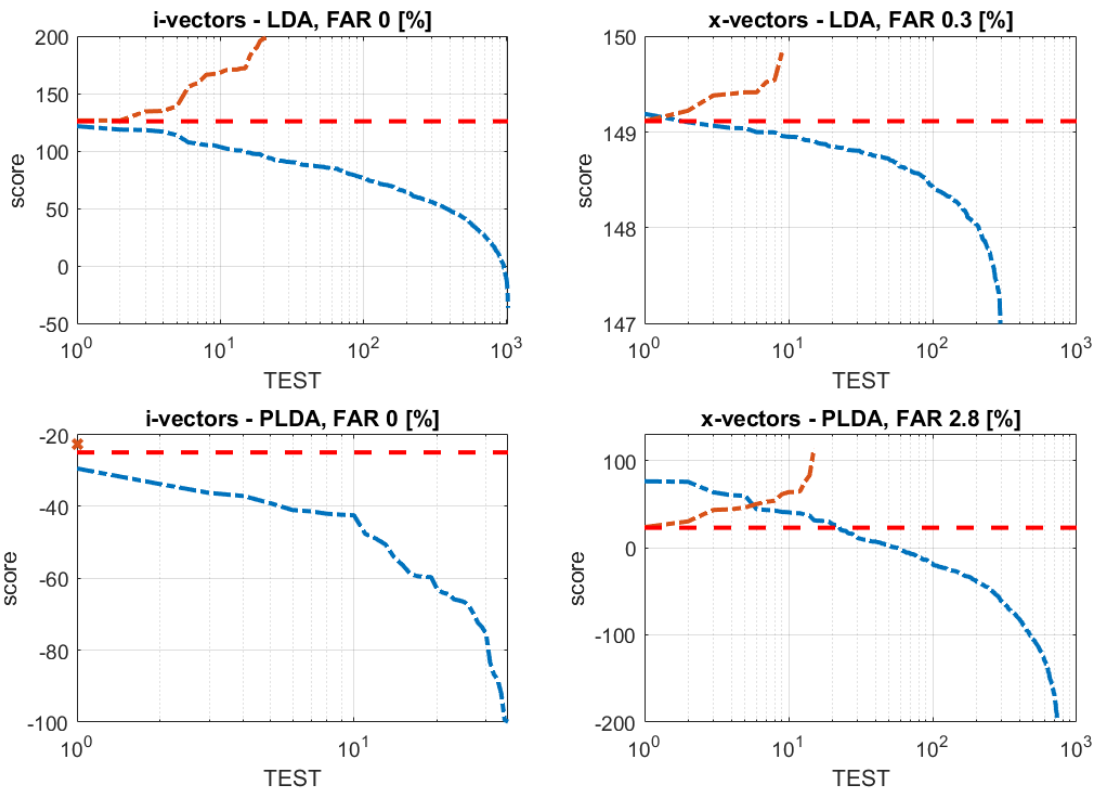
Během první fáze testování, tj. zápisu, uživatelé měli předčítat jednotlivé náhodné věty zobrazované aplikací z dostupného souboru vět. Celkový počet promluv použitých pro zápis si uživatelé měli zvolit sami v rozsahu 3 až 10 nahrávek, přičemž průměrná doba 1 nahrávky byla 5,77 sekund.

Po zapsání dostatečného počtu uživatelů, byli titíž uživatelé požádáni, aby v aplikaci provedli své rozpoznávání. V této fázi uživatelé stejným způsobem jako v předchozí fázi měli nahrávat promluvy a odesílat je na rozpoznání. Řečníci měli za úkol nahrát samostatně věty pro rozpoznávání na bázi i-vektorového systému a samostatně pro systém na bázi x-vektorů. Po dokončení rozpoznávání a zobrazení výsledku uživatelé dávali zpětnou vazbu přímo v aplikaci, zda je daný systém určil správně.

Z počátku nebyl nastavený práh pro identifikaci na otevřené množině. Později byl práh nastaven podle dostupných zpětných vazeb a mohlo tak probíhat i otestování verifikačního prahu.

### 4.4.2 Nastavení prahu

Při testování aplikace nebyl úmyslně z počátku nastaven práh pro identifikaci na otevřené množině. Systém se tedy choval jako rozpoznávání na uzavřené množině a každý uživatel byl rozpoznán pod nějakým jménem. Tento proces testování byl zvolen, aby byly získány množiny výsledků, které obsahují napočítaná skóre pro testy, při kterých se jednalo o dvojici reprezentující shodnou osobu a případy, při kterých se nejednalo o shodnou osobu. Tyto množiny skóre byly následně použity pro správné nastavení prahu pro identifikaci na otevřené množině.




**Obrázek 11:** Nastavení verifikačního prahu (červeně) dle vyhodnocení rozpoznávacích metod na reálných datech. Testy k přijetí (oranžově), testy k zamítnutí (modře)

Po krátkém otestování webového systému bylo nastaveno, že uživateli se zobrazí výsledky metody *LDA* pro systém na bázi i-vektorů a *PLDA* pro systém na bázi x-vektorů. Následně po sbírání dat se provedla analýza systémů v kombinaci s oběma metodami vyhodnocení rozpoznávání *LDA* i *PLDA*. U každé metody byly sestaveny dvě množiny a uspořádány do grafu, viz obrázek 11. Oranžová křivka představuje testy, které mají být přijaty, a modrá křivka představuje testy k zamítnutí. Verifikační prahy byly zvoleny jako minimální skóre z množiny k přijetí, aby bylo docíleno, že co nejvíce osob systém rozpozná. Toto nastavení je vhodné pouze pro demonstrační účely. Následně byla spočítána hodnota *FAR* pro daný práh. Tímto nastavením je zaručeno, že hodnota *FRR* je nulová.

Ačkoliv v experimentální části v kapitole 3.4 bylo zmíněno, že nejlepších výsledků bylo dosaženo pomocí metody *PLDA*, v testování webové aplikace se ukázalo, že lepší

výsledky dává metoda *LDA*. U systému na bázi i-vektorů metoda *PLDA* dává sice hodnotu *FAR* rovnou nule, avšak dle zpětné vazby proběhlo pouze jedno správné rozpoznání (označené v grafu křížkem), proto lze říct, že metoda *PLDA* v kombinaci se systémem na bázi i-vektorů je zcela nevyhovující, zatímco metoda *LDA* v kombinaci se systémem na bázi i-vektorů funguje perfektně a dává hodnotu *FAR* nulovou a lze tedy nastavit práh takový, při kterém bude vždy správně rozhodnuto, jaké testy zamítnout a jaké přijmout.

Systém na bázi x-vektorů v kombinaci s *PLDA* funguje výrazně lépe než u předchozího systému. Avšak jeho hodnota *FAR* není nulová a to znamená, že můžou nastat případy, při kterých systém daného řečníka neodmítne a následně ho rozpozná chybně. Metoda *LDA* v kombinaci se systémem na bázi x-vektorů sice dává nižší hodnotu *FAR* a systém tak pracuje spolehlivěji, ale na stejných datech správně rozpoznal menší počet testů než *PLDA*.

### 4.4.3 Orientační výsledky vyhodnocení

Prvním výsledkem je hodnocení funkčnosti aplikace *SREDEMO*. Dle zpětných vazeb od uživatelů, aplikace funguje spolehlivě a rychle a je možné ji otevřít na libovolné moderní počítačové platformě. Grafické rozhraní sice není optimalizované pro vertikální formáty obrazu, ale na klasických zobrazovačích s horizontálním formátem (např. 16:9) funguje přizpůsobivě, a je přehledné.

Druhé hodnocení se týká samotných systémů *SRE*, přičemž následující výsledky byly dosaženy pro systém na bázi i-vektorů v kombinaci s metodou *LDA* a pro systém na bázi x-vektorů v kombinaci s metodou *PLDA*. Tyto kombinace metod byly zvoleny, protože podle předchozího vyhodnocení se jevily jako nejlépe fungující. Během celého testování se zapsalo do databáze aplikace *SREDEMO* 45 mluvčích ve věku od 23 let do 61 let. Celkem proběhlo 96 testů a z toho bylo 9 bez zpětné vazby, ty byly vyřazeny z následujícího vyhodnocení. Výsledná úspěšnost x-vektorového systému činí 71,4 %, zatímco systém na bázi i-vektorů dosáhl 73 %, počty úspěšných pokusů z celkového počtu pokusů jsou uvedeny v tabulce 4. Celkový počet neúspěšných pokusů o rozpoznání činí 24, z čehož 6 pokusů provedl jeden uživatel, který chtěl otestovat různé mikrofony. Tímto pokusem bylo zjištěno, že výsledný systém má problém identifikovat jedince zapsaného pomocí nějakého kvalitnějšího mikrofonu a následného rozpoznání pomocí nekvalitního mikrofonu. Ostatní uživatelé neprováděli takové množství testů, a pokud by se zmíněné testy vynechaly, úspěšnost systému na bázi i-vektorů by byla 77 % a u x-vektorů 78 %.

| | **úspěšné pokusy** | **celkový počet pokusů** | **úspěšnost [%]** |
|---|---|---|---|
| i-vektor | 38 | 52 | 73 |
| x-vektor | 25 | 35 | 71,4 |

**Tabulka 4:** *SREDEMO* - Úspěšnost rozeznávání řečníků pro zapsaných 45 mluvčích

Celkový počet pokusů testů x-vektorového systému je výrazně menší než u systému na bázi i-vektorů. Důvodem pravděpodobně je, že mnoho uživatelů provedlo pouze test jednoho systému a pak opustili webovou aplikaci. Avšak dle výsledků v předchozí kapitole by se dalo čekat, že úspěšnost x-vektorového systému bude

daleko horší než i-vektorového systému, místo toho oba systémy pracují s velmi podobnou úspěšností nad 71 %, kterou by jistě bylo možné výrazně zlepšit. Protože sbírání vzorků z reálného on-line provozu trvalo dlouho, po provedení tohoto finálního vyhodnocení bohužel již nezbýval čas na úpravy systému za účelem zlepšení úspěšnosti.

## 4.5 Jak změnit systém SRE v aplikaci

Aby bylo možné v budoucnu na této aplikaci dále pracovat a přidávat nové funkce, nebo nové systémy na rozpoznávání řečníka, nebo případně aktualizovat současné systémy na rozpoznávání řečníka, je třeba v této kapitole uvést, jaké změny v aplikaci je pro takové případy potřeba provést.

Nejjednodušší variantou je pouze aktualizace natrénovaných extraktorů i-vektorů (nebo x-vektorů) jednoho, případně obou systémů provádějících úlohu *SRE* a s tím spojená aktualizace i transformačních matic *LDA* a modelů *PLDA*. V takovém případě je možné provést aktualizaci v příslušné složce daného systému nahrazením složky `exp`, která dle Kaldi konvence uchovává zmíněné soubory. Není třeba provádět žádné změny ve skriptech Kaldi za předpokladu, že jsou zachovány shodné názvy souborů.

Druhou jednoduchou variantou je kompletní záměna například i-vektorového systému za jiný i-vektorový systém. V takovém případě by také nebylo potřeba provádět žádné změny v uživatelském rozhraní ani v jádru aplikace. Stačilo by pouze nahradit obsah celého adresáře uchovávající zmíněný systém kromě spouštěcího souboru `main_run.sh`. Tento spouštěcí soubor je totiž volaný v jádru aplikace (viz kód 13) a spouští hlavní Kaldi skript v `Dockeru`. Nově nahrazený systém by však musel mít své výstupy nastaveny shodně se současným systémem, aby bylo zaručeno fungování celé aplikace. Tedy bylo by třeba brát v potaz adresářovou strukturu, do které je členěna databáze pojímající jednotlivé řečníky, viz 4.3.3.

Poslední složitější varianta je přidání nového systému k aktuálním systémům rozpoznávání. Současná aplikace na něco takového není připravena a vyžadovalo by to úpravu všech částí aplikace, tedy uživatelského rozhraní i samotného jádra aplikace na bázi `Django`. Počínaje uživatelským rozhraním, bylo by potřeba přidat tlačítka, která by tuto novou metodu obsluhovala, a s tím spojené funkce, které by potřebné nahrávky odesílaly na server. Nejjednodušším způsobem je pouze přidání nové možnosti vybírající tuto metodu vedle možností `i-vektors` a `x-vectors` ve formuláři na odeslání na server. V takovém případě by se dala využít funkce, která odesílá požadavek na server, a jen by se její parametr `method` nastavoval na požadovanou hodnotu. Tato úprava by však musela být provedena přímo ve zdrojových komponentách frameworku `React` a následně by musela být přeložena do produkční verze. V posledním kroku by musel framework `Django` provést sběr nových statických souborů pomocí příkazu: `python manage.py collectstatic`.

Popsaná změna by se však projevila pouze na frontendu aplikace, další změna by musela být v samotném jádru aplikace, a to v `Django` aplikaci `api`. Změna by však mohla být provedena přímo na souborech na serveru, protože `Django` nevyžaduje překlad souborů psaných v jazyce `Python`. Jednalo by se o změnu v pohledu obsluhu-

jící spuštění rozpoznávání, aby v případě parametru `method` nastaveného na novou metodu spustil správný skript obdobným způsobem jako v kódu 13. Ve zmíněném pohledu pro rozpoznávání je implementovaná funkce, která na základě parametru `method` provede danou úlohu v Kaldi. Změna by se tedy mohla provést pouze zde, viz kód 15.

```
def KaldiAndResponse(record, spkr_id, p_type, method):
    try:
        # creating i-vectors, p_type = enroll or test
        if method == "ivector":
            subprocess.run(["../v1/main_run.sh",p_type,spkr_id], check=True)
        # creating x-vectors, p_type = enroll or test
        if method == "xvector":
            subprocess.run(["../v2/main_run.sh",p_type,spkr_id], check=True)
        # NEW METHOD, p_type = enroll or test
        if method == "NEW_METHOD":
            subprocess.run(["../XX/main_run.sh",p_type,spkr_id], check=True)
        if process_type == "enroll":
            return JsonResponse(status=status.HTTP_201_CREATED)
        if process_type == "test":

            ...
```

**Kód 15:** Zjednodušený příklad úpravy funkce pro spuštění nové metody

Když by se vykonaly výše uvedené změny, samotný systém vykonávající novou metodu by pak mohl být umístěný v libovolné složce. Bylo by však vhodné, aby nový systém zachovával adresářovou konvenci Kaldi a bral v potaz databázovou strukturu aplikace *SREDEMO*. Možných změn by však mohlo být více jak ve frontendu aplikace, tak i v backendu, například by mohly být přidané nové modely pro ukládání dat do `SQL`. Na závěr, aby se veškeré změny projevily, je však vždy třeba restartovat webový server `Apache2`. Pokud tak není provedeno, server provádí vždy poslední verzi aplikace, která byla v době spuštění serveru v adresářích nehledě na současný obsah.

Zdrojové kódy uživatelského rozhraní i serverové části výsledné aplikace, která byla popsaná v předchozích sekcích a která neobsahuje výše popsané změny, jsou dostupné v příloze této práce.

# 5 Uživatelské rozhraní webové aplikace

V předchozích částech této práce, konkrétně v kapitolách 3 a 4, bylo detailně rozebráno technické řešení aplikace *SREDEMO*. Nyní se text zaměřuje na uživatelské rozhraní a jednotlivé funkce této aplikace. *UI* bylo navrženo s důrazem na minimalistický a intuitivní design světlých barev, který je optimalizován pro širokoúhlé formáty obrazovek s poměrem stran 16:9. V této kapitole je detailně popsán vzhled a chování *UI* a jeho jednotlivých prvků. Současně je zde uveden odkaz na aktuální verzi aplikace, která v době psaní této práce je dostupná on-line na adrese: `https://aspeech.fel.cvut.cz/sredemo`.

## 5.1 Popis uživatelského rozhraní

Když uživatel přijde na titulní stránku aplikace *SREDEMO*, má na výběr ze dvou možností. První možností je otevření rozhraní pro nový zápis řečníka do databáze, ve kterém je možné i nahrát dodatečné promluvy a přidat je k existujícímu zápisu v databázi. Druhou možností je otevření rozhraní pro nahrávání promluv určených k rozpoznání neznámého řečníka.

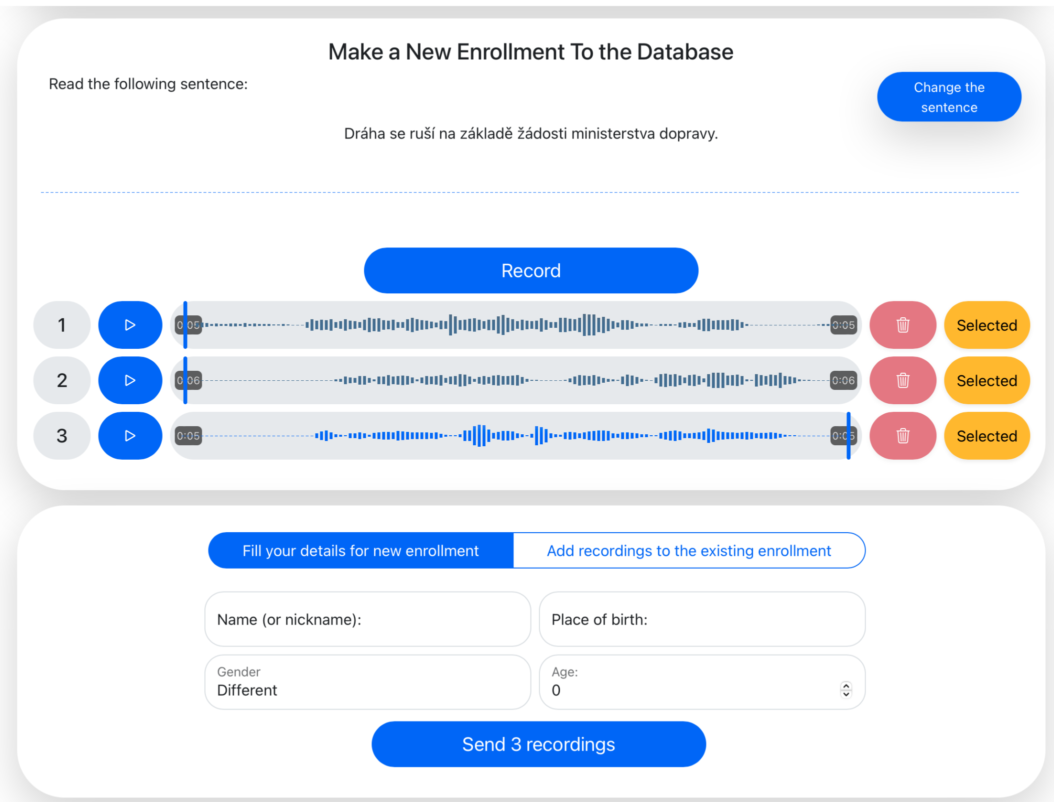
V obou případech zvolení dané operace se otevře z počátku podobné rozhraní, které je uvedeno v horní části obrázku 12. Uživateli se zobrazuje věta, kterou by měl přečíst pro nahrání svého hlasu, případně uživatel může do promluvy nahrát libovolnou větu. Věta, která se zobrazuje, je později uložena společně s nahrávkou do databáze. Případ, že řečník nahraje obsah věty rozdílný s textem, není možné zjistit, a proto je třeba brát obsah uložených vět v databázi jen orientačně. Předčítanou větu je možné změnit tlačítkem na pravé straně.

Pod řádkem s vypsanou větou se nachází indikátor vybuzení mikrofonu. Na obrázku 12 je znázorněn modrou přerušovanou čárou. Během nahrávání indikátor vykresluje krátkodobý časový průběh zvukové vlny, čímž je mluvčímu dána vizuální zpětná vazba o tom, jak nahrávání probíhá.

Nejdůležitější je tlačítko `Record`, kterým se zahajuje nahrávání dané promluvy. Po prvním aktivování nahrávání se uživateli zobrazí žádost prohlížeče o přístup k mikrofonu, poté tlačítko změní barvu na červenou a začne nahrávání. Při druhém stlačení, nyní červeného tlačítka, se kromě ukončení nahrávání automaticky změní věta k předčítání a v dolním řádku se zobrazí ovládání nahrané promluvy. Maximální délka nahrávky je nastavena na 12 sekund, což odpovídá dvojnásobku průměrné délky jedné promluvy. Při překročení této doby je automaticky nahrávání zastaveno.

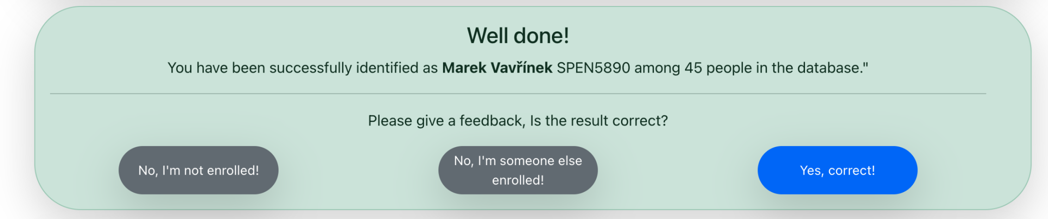
Jednotlivé nahrané promluvy se objevují pod sebou od nejstarší po nejnovější. Nahrávku si je možné poslechnout a případně smazat, přičemž rozhraní umožňuje uchovávat v tomto seznamu maximálně deset nahrávek. Po vybrání minimálního

množství nahrávek, které je pro zápis do databáze nastaveno na hodnotu tři, se zobrazí formulář pro zadání údajů nového zápisu řečníka do databáze, ten je vidět na obrázku 12 v dolní části. Pokud uživatel zadá údaje osoby, která je již v databázi zapsaná, je o tom informován a k zápisu do databáze nedojde. Zmíněný formulář je možné přepnout do režimu přidání promluv k existujícímu zápisu v databázi. Pak jediným údajem, který uživatel musí zadat, je unikátní ID, pod kterým byl vytvořen daný zápis v databázi a který se zobrazí po úspěšném zápisu do databáze.




**Obrázek 12:** *SREDEMO* - Rozhraní nahrávání promluv pro nový zápis do databáze řečníků

V případě úlohy rozpoznávání je rozdíl v *UI* pouze ve formuláři pro odeslání. Uživatel nezadává žádné své údaje, pouze může zvolit, zda chce rozpoznání provést na základě metody využívající i-vektory, nebo využívající x-vektorů. Zde není nastaven minimální počet promluv, uživatel tak může zkoušet systém, jak dobře rozpoznává, už od jedné promluvy, přičemž délka jedné promluvy bývá okolo 5 sekund.




**Obrázek 13:** *SREDEMO* - Výsledek rozpoznání se zpětnou vazbou

Výsledkem rozpoznání je jméno zapsaného řečníka v databázi společně s jeho unikátním ID a informací o tom, z kolika zápisů v databázi proběhlo rozpoznávání. Dále zde uživatel může podat zpětnou vazbu, jestli se rozpoznání povedlo, či nikoliv. Příklad výsledku rozpoznání je vidět na obrázku 13.

## 5.2 Popis rozhraní administrátora

Z bezpečnostních důvodů k rozhraní pro administrátora nevede z titulní stránky, ani z jiné části aplikace přímý odkaz. Administrátor tak musí zadat URL adresu z paměti, aby se tak dostal na požadovanou stránku, na které je nejprve vyzván o zadání přihlašovacího jména a hesla. V dnešní době se přihlašování pomocí formuláře se zadáním jména a hesla provádí pomocí složitých postupů, které jsou navrženy tak, aby pokud možno nešly prolomit. Vývoj takto zabezpečeného systému je však nad rámec této práce, a proto v aplikaci *SREDEMO* je implementován nejjednodušší způsob pro přenos přihlašovacích údajů na server. Po ověření totožnosti administrátora se zobrazí požadované rozhraní. Při jednotlivých požadavcích z tohoto rozhraní na server již další zabezpečení není implementováno. Zde je prostor pro budoucí zlepšení systému.



**Obrázek 14:** *SREDEMO* - Zobrazení zapsaných řečníků

Rozhraní pro administrátora je rozděleno do tří sekcí, které lze přepínat ovládáním v horní části obrázku 14. V první části určené pro zobrazení jednotlivých řečníků zapsaných v databázi administrátor vidí jednotlivé zápisy jako řádky pod sebou spolu s nejdůležitějšími informacemi, jako je ID mluvčího, jméno, místo narození, pohlaví, věk a datum vytvoření zápisu. Po stisknutí tlačítka s nápisem ID daného mluvčího se zobrazí další informace, které se týkají jednotlivých promluv patřících řečníkovi. Společně s tím se zobrazí i akce pro možnost smazání zápisu v databázi (resp. přesun do složky `deleted`). Po stisknutí tlačítka ke smazání se ještě zobrazí dialogové okno, kde administrátor musí potvrdit, že opravdu chce zápis smazat. Popsané rozhraní je vidět na obrázku 14.

Druhou částí rozhraní pro administrátora je zobrazení jednotlivých pokusů o rozpoznání řečníka. Zde jednotlivé řádky představují jednotlivé pokusy rozpoznání a obsahují informace o ID daného rozpoznání, výsledku rozpoznání, použité metody k rozpoznání, zpětnou vazbu a datum provedení rozpoznání. Po stisknutí ID

daného testu se stejně jako v předchozím případě zobrazí informace o jednotlivých promluvách společně s akcí smazání. Pole zpětné vazby může nabývat čtyř hodnot. Šedé pole značí, že uživatel nezadal zpětnou vazbu. Zelené pole značí, že rozpoznání bylo úspěšné, žluté a červené pole říká, že rozpoznání bylo neúspěšné, žluté navíc říká, že systém špatně neodhalil, že daná osoba není zapsaná v databázi, červená označuje chybu rozpoznání osoby, která je zapsaná v databázi. Na obrázku 15 je vidět popsané rozhraní.



**Obrázek 15:** *SREDEMO* - Zobrazení provedených rozpoznávání

Poslední třetí část rozhraní pro administrátora vypisuje jednotlivé věty, které mají řečníci předčítat. Rozhraní dále umožňuje vybranou větu smazat. Po stisknutí příslušného tlačítka akce se opět zobrazí dialogové okno s potvrzením smazání. Na rozdíl od předchozích dvou případů se daná věta smaže definitivně, tedy není zálohována do jiné složky. Dále rozhraní umožňuje přidání nové věty do souborů všech vět. Toto rozhraní je vidět na obrázku 16 a v době psaní této práce obsahuje 800 různých vět.



**Obrázek 16:** *SREDEMO* - Zobrazení a možnost úpravy vzorových vět

# 6 Závěr

Cílem předložené práce bylo seznámit se s problematikou rozpoznání řečníka na bázi i-vektorů a x-vektorů, realizovat zmíněné systémy pro rozpoznávání řečníka pomocí nástrojového balíku Kaldi a zmíněné systémy otestovat na evaluačních datech. Druhým cílem bylo navrhnout webovou aplikaci pro demonstrační účely, která má využívat realizované systémy na rozpoznání řečníka.

Zadání práce bylo splněno ve všech bodech. Systémy provádějící úlohu rozpoznávání řečníka byly navrženy a implementovány jako dva rozdílné systémy, jeden využívající *GMM* pro generování i-vektorů a druhý založený na *DNN*, který produkuje x-vektory. Oba systémy byly otestovány na datech získaných z databáze *SPEECON* za účelem porovnání jejich schopnosti identifikace na otevřené i uzavřené množině řečníků. Testování na evaluačních datech ukázalo, že x-vektorový systém poskytuje vyšší přesnost rozpoznávání v porovnání s i-vektorovým systémem, což lze připsat schopnosti *DNN* lépe modelovat komplexní vzory a variabilitu v řečových datech. Dalším důležitým výsledkem je, že databáze *SPEECON* obsahuje dostatečné množství dat pro natrénování *DNN* pro úlohu *SRE*, neb byly pro trénování použity promluvy s celkovým trváním okolo 9 hodin.

Další částí práce byl návrh webové aplikace pro demonstraci funkcionality systémů rozpoznávání řečníka. Uživatelské rozhraní aplikace bylo navrženo pomocí frameworku `React` a jádro aplikace na straně serveru bylo implementováno pomocí knihovny `Django`. Samotné rozpoznávání řečníka aplikace se provádí pomocí výše zmíněných systémů na základě Kaldi, které jsou na aplikaci napojené pomocí kontejnerového systému `Docker`. Aplikace umožňuje registraci nových řečníků, přidání záznamových dat k již existujícím řečníkům a identifikaci neznámých řečníků na základě nahrávek, které uživatelé nahrávají prostřednictvím webového rozhraní. Aplikace po zobrazení výsledku rozpoznávání umožňuje i zaznamenat zpětnou vazbu uživatele pro možnost hodnocení celkového systému aplikace. Všechny důležité informace ohledně databáze zapsaných uživatelů a o výsledcích jednotlivých rozpoznávacích testů jsou přehledně členěny v administrátorské části webové aplikace. Tento interaktivní nástroj poskytuje užitečnou platformu pro praktické ověření a demonstraci technologií rozpoznání řečníka, který může být použit ve výuce a zároveň může sloužit jako nástroj pro sbírání zvukových nahrávek s cílem vytvořit zvukovou databázi pro další výzkum.

Závěrečné testování funkcionality v reálném on-line provozu bylo provedeno na základě zpětné vazby od uživatelů. Testovala se funkčnost samotné aplikace a orientačně funkčnost systémů *SRE*. V první fázi testování byla sbírána data pro nastavení verifikačního prahu pro úlohu identifikace na otevřené množině. Během tohoto testování se ukázalo, že navržený systém na bázi i-vektorů v kombinaci s *PLDA* dává horší výsledky, zatímco i-vektorový systém v kombinaci s *LDA* funguje výborně, dokonce lépe než systém na bázi x-vektorů, neb se dal najít takový práh, pro který hodnota *FAR* je nulová. U realizovaného x-vektorového systému v kombinaci s *PLDA* je hodnota *FAR* rovna 2,8 % a u systému kombinovaného s *LDA* 0,3 %. Důvodem roz-

dílných výsledků testování oproti testování na evaluačních datech je pravděpodobně malý počet testů a nízká kvalita zvukových nahrávek způsobená různou technikou, kterou uživatelé použili během nahrávání, a také prostředím s větším rušením na pozadí, například integrované mikrofony v noteboocích umístěných hned vedle chladících ventilátorů.

V druhé fázi testování byla orientačně testována úspěšnost systémů *SRE*. Dle předchozího kroku bylo vybráno, že aplikace bude jako výsledky rozpoznávání zobrazovat výsledky od systému na bázi i-vektorů v kombinaci s *LDA* a od x-vektorového systému v kombinaci s *PLDA*. Dle předchozího vyhodnocení se tyto kombinace jevily jako nejlépe fungující. Proběhlo celkově 96 jednotlivých testů, avšak celkový počet pokusů x-vektorového systému byl výrazně menší než u i-vektorového, kvůli tomu, že mnoho uživatelů vyzkoušelo pouze jeden systém a opustilo aplikaci. Překvapivě však oba systémy dosahují podobné úspěšnosti nad 71 %, kterou by bylo možné zlepšit, opětovným trénováním a testováním extraktorů na evaluačních datech obohacených o nasbíraná data. Avšak kvůli nedostatku času po sběru dat z reálného provozu zmíněné úpravy systému nebyly provedeny.

Dalším výsledkem výše popsaného testování je uživatelské otestování výsledné aplikace. Dle zpětných vazeb uživatelů aplikace *SREDEMO* je webové rozhraní dobře navržené a samotná aplikace funguje správně a rychle na všech moderních počítačových platformách. Přílohou této práce jsou zdrojové kódy serverové části aplikace, uživatelského rozhraní i obou metod rozpoznávání řečníků.

V budoucí práci by bylo vhodné zaměřit se na zvýšení robustnosti obou systémů vůči šumu a hluku a na rozšiřování funkcionalit webové aplikace o další metody z oblasti zpracování řeči. Možným vylepšením by mohlo být přidání řečového detektoru přímo do webové aplikace, aby uživatel nemusel spouštět a ukončovat nahrávání, ale aby mohl pouze mluvit a systém by automaticky rozřezával jednotlivé věty do samostatných souborů a na pozadí je nahrával na server. Dalším vhodným vylepšením by mohlo být vylepšení zabezpečení administrátorské části uživatelského rozhraní. Současná aplikace zde využívá pouze jednoduché zabezpečení pomocí přístupového hesla. V budoucnu by se toto mohlo nahradit modernějším a bezpečnějším způsobem. Dále by se daly do aplikace přidat další metody z oblasti zpracování řeči, nebo pokročilejší nástroje na nahrávání a úpravu promluv. Takto by postupně mohla vzniknout moderní užitečná aplikace pracující v on-line prostředí na nahrávání mluveného slova.

# Literatura a použité zdroje


[1] W. Isaacson, *The Innovators.* Simon and Schuster, 2015. ISBN-13 978-1476708706.

[2] J. Psutka *et al.*, *Mluvíme s počítačem česky.* Praha: Academia, 2006. ISBN 8020013091.

[3] J. Silovský, *Generativní a diskriminativní klasifikátory v úlohách textově nezávislého rozpoznávání a diarizace mluvčích.* Disertační práce, Technická Univerzita v Liberci, 2011.

[4] M. Záruba, *Moderní metody rozpoznávání mluvčího na bázi GMM a DNN.* Diplomová práce, České vysoké učení technické v Praze, 2016.

[5] L. Machlica, *Vysokodimenzionální prostory a modelování v úloze rozpoznávání řečníka.* Disertační práce, Západočeská univerzita v Plzni, 2012.

[6] D. Yu and L. Deng, *Automatic Speech Recognition A Deep Learning Approach.* Springer-Verlag, 2015. ISBN 978-1-4471-5778-6.

[7] D. Snyder *et al.*, "X-Vectors: Robust DNN Embeddings for Speaker Recognition," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5329–5333, 2018. DOI: 10.1109/ICASSP.2018.8461375.

[8] D. Povey *et al.*, "The Kaldi Speech Recognition Toolkit," in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, IEEE Signal Processing Society, pros. 2011. IEEE Catalog No.: CFP11SRW-USB.

[9] brijmohan, "Fix for Kaldi allocate_egs.py." Dostupné z: https://github.com/brijmohan/kaldi/commit/62b8ed90e261a6bb5088dfe506b7972dd052743f?fbclid=IwZXh0bgNhZW0CMTAAAR0k0pSZNn4kMuKr_UXCS4EWVzS-IXDUIiMCjO6bM6TIZ5qRXlGFG4Jg1Fo_aem_AdWlEXlIU_XCYYMM95hF_iiQRQPwl3BRFsIJSws4DhR88HU-i_CEE4LrWBVU6Kvmb6Oa6aKLIXo7pUNRzwB9gMfb, 2021. [cit. 14. 5. 2024].

[10] P. Pollák and J. Černocký, *Czech SPEECON Adult Database.* Dokumentace databáze, ČVUT Praha a VUT Brno, 2004.

[11] H. M. Abdullah and A. M. Zeki, "Frontend and backend web technologies in social networking sites: Facebook as an example," in *2014 3rd International Conference on Advanced Computer Science Applications and Technologies*, pp. 85–89, 2014.

[12] D. Rubio, *Beginning Django: Web Application Development and Deployment with Python.* Berkeley, CA: Apress L. P, 2017. ISBN 1484227867.

[13] F. Ferreira and M. T. Valente, "Detecting code smells in react-based web apps," *Information and Software Technology*, vol. 155, p. 107111, 2023.

[14] M. Otto *et al.*, "Bootstrap." Dostupné z: https://getbootstrap.com/. [cit. 13. 2. 2024].

[15] Katspaugh *et al.*, "Wavesurfer." Dostupné z: https://wavesurfer.xyz/. [cit. 13. 2. 2024].

[16] M. Diamond, "recorder-js." Dostupné z: https://www.npmjs.com/package/recorder-js. [cit. 17. 4. 2024].