

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ADAPTACE A DEPLOYMENT AUTOMATICKÉHO PŘEPISU JAKO KONTEJNER DO UMBREL / START9

ADAPTATION AND DEPLOYMENT OF AN AUTOMATIC TRANSCRIPTION SERVICE AS A CONTAINER ON UMBREL / START9

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DANIEL DEDEK**
AUTHOR

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

BRNO 2026

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

166988

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Dedek Daniel**
Program:	Informační technologie
Název:	**Adaptace a deployment automatického přepisu jako kontejner do Umbrel / Start9**
Kategorie:	Zpracování řeči a přirozeného jazyka
Akademický rok:	2025/26

Zadání:

1. Nastudujte platformy jako jsou Umbrel nebo Start9.
2. Nastudujte nástroj pro automatického rozpoznávání řeči (ASR), například NeMo nebo HuggingFace.
3. Zabalte zvolený rozpoznávač jako modul pro Umbrel / Start9.
4. Vytvořte UI tak, aby šlo ASR používat uživatelsky přívětivě. Cílem je mít osobní ASR bez nutnosti sdílet data do cloudu. Prozkoumejte i možnost adaptace rozpoznávače na uživatelských datech.
5. Zhodnoťte dosažené výsledky a navrhněte směry dalšího vývoje.
6. Vytvořte A2 plakátek nebo 20-30 vteřinové video o Vašem projektu.

Literatura:

- Alec Radford et al. "Robust Speech Recognition via Large-Scale Weak Supervision", https://cdn.openai.com/papers/whisper.pdf, 2022
- Oleksii Kuchaiev, Jason Li, Huyen Nguyen, et al. "NeMo: a toolkit for building AI applications using Neural Modules", https://arxiv.org/abs/1909.09577, 2019
- Dále podle pokynů školitele

Při obhajobě semestrální části projektu je požadováno:
Body 1, 2, a část bodu 3, 4 ze zadání.

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

Vedoucí práce:	**Szőke Igor, Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2025
Termín pro odevzdání:	13.5.2026
Datum schválení:	7.11.2025

Abstrakt

Tato práce se zabývá návrhem a implementací webové aplikace pro automatický přepis řeči, určené pro běh na platformě Umbrel OS. Cílem je umožnit lokální zpracování audio dat bez nutnosti využití cloudových služeb. Aplikace je realizována jako klient-server systém s backendem ve frameworku Flask a poskytuje uživatelské rozhraní pro nahrávání, správu a přepis zvukových souborů ve formátu WAV.

Pro přepis jsou využity moderní ASR modely Whisper, Canary a Parakeet, které lze dynamicky přepínat a konfigurovat. Důraz je kladen na modularitu systému a možnost jeho dalšího rozšiřování. Výsledný přepis lze zobrazit a stáhnout ve formátu TXT, přičemž u modelu Whisper jsou dostupné i doplňující informace, jako časové značky.

Abstract

This thesis focuses on the design and implementation of a web application for automatic speech-to-text transcription running on the Umbrel OS platform. The goal is to enable local processing of audio data without relying on cloud services. The application is implemented as a client-server system using the Flask framework and provides an interface for uploading, managing, and transcribing WAV audio files.

The system supports multiple ASR models, namely Whisper, Canary, and Parakeet, which can be dynamically selected and configured. Emphasis is placed on modularity and extensibility. The resulting transcription can be displayed and downloaded as a TXT file, with additional information such as timestamps available for the Whisper model.

Klíčová slova

automatické rozpoznávání řeči, přepis řeči, webová aplikace, Flask, Python, Whisper, NVIDIA NeMo, ASR modely, zpracování audia, Umbrel

Keywords

automatic speech recognition, speech-to-text, web application, Flask, Python, Whisper, NVIDIA NeMo, ASR models, audio processing, Umbrel

Citace

DEDEK, Daniel. *Adaptace a deployment automatického přepisu jako kontejner do Umbrel / Start9*. Brno, 2026. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Igor Szőke, Ph.D.

Adaptace a deployment automatického přepisu jako kontejner do Umbrel / Start9

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doktora Igora Szőkeho. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal. Pro jazykovou revizi technické zprávy jsem použil nástroj ChatGPT. Při tvorbě odevzdaného programového kódu jsem využíval nástroj Github Copilot jako asistenční nástroj pro generování návrhů kódu. Veškerý programový kód byl následně kontrolován, upravován a integrován do výsledného řešení.

.....
Daniel Dedek
6. května 2026

Poděkování

Rád bych tímto poděkoval svému vedoucímu bakalářské práce, Ing. Igoru Szőkemu, Ph.D., za odborné vedení, cenné rady, připomínky a ochotu věnovat čas konzultacím při zpracování této práce. Jeho zkušenosti a podpora byly pro mě během řešení práce velmi přínosné.

Obsah

1 Úvod **3**

2 Umbrel/Start9 **4**
2.1 Umbrel 4
2.2 Start9 5
2.3 Porovnání a důvody volby platformy Umbrel 6
2.4 Shrnutí důvodů volby Umbrelu 7

3 Průzkum podobných aplikací **8**
3.1 Aplikace umožňující přepis řeči do textu 8
3.2 Open WebUI a Ollama 15
3.3 Shrnutí zajímavých funkcí podobných aplikací 17

4 Implementace **19**
4.1 Architektura aplikace 19
4.2 Správa souborů 20
4.3 Abstraktní vrstva modelů 21
4.4 Tovární funkce a cache 21
4.5 Uživatelské rozhraní 22
4.6 Nasazení 23
4.7 Struktura souborů a adresářů 24
4.8 Použité technologie 24
4.9 Modely 26
4.10 Nasazení aplikace na platformě Umbrel 31

5 Přehled funkcí aplikace **35**
5.1 Správa audio souborů 35
5.2 Volba ASR modelu 35
5.3 Konfigurace parametrů přepisu 36
5.4 Zobrazení výsledků přepisu 36
5.5 Export a práce s výsledky 36
5.6 Uživatelský komfort a odezva systému 37
5.7 Shrnutí funkčních možností 37

6 Testování **38**
6.1 Testovací sada 38
6.2 Výsledky testování 41
6.3 Uživatelské testování 41

7 Další rozvoj aplikace **42**
7.1 Sumarizace textu 42
7.2 Nabídka více modelů 42
7.3 Přepis řeči v reálném čase 44
7.4 Segmentace a diarizace mluvčích (speaker segmentation and diarization) . . 45
7.5 Vstupních formáty 46
7.6 Výstupní formáty 46
7.7 Překlad 47
7.8 Ukládání historie přepisů 48
7.9 Paralelní zpracování a dávkové úlohy 48
7.10 Webové API 48
7.11 Shrnutí dalšího rozvoje aplikace 48

8 Závěr **49**

Literatura **50**

Kapitola 1

Úvod

Automatické rozpoznávání řeči patří mezi rychle se rozvíjející oblasti umělé inteligence a zpracování přirozeného jazyka. Díky pokroku v neuronových sítích a dostupnosti výpočetního výkonu je dnes možné vytvářet systémy, které spolehlivě převádějí mluvenou řeč na text, doplňují časové značky, rozlišují mluvčí a poskytují další informace zvyšující využitelnost přepisu.

Cílem práce je navrhnout a implementovat webovou aplikaci v prostředí Umbrel, která umožní uživatelsky přívětivý přepis řeči ze zvukových souborů. Aplikace je realizována jako klient-server systém, kde server zajišťuje zpracování audio dat pomocí modelů strojového učení a uživatel může volit mezi různými modely a jejich konfigurací.

Systém podporuje modely Whisper, Canary a Parakeet, které se liší přístupem k rozpoznávání řeči, přesností i výpočetní náročností. Důraz je kladen na modularitu a rozšiřitelnost, aby bylo možné snadno integrovat další modely nebo rozšířit funkcionalitu.

Součástí aplikace je uživatelské rozhraní pro správu audio souborů, jejich přehrávání a zobrazení přepisu. Výstup je možné stáhnout ve formátu TXT, přičemž u vybraných modelů jsou k dispozici i doplňující informace, jako časové značky či míra jistoty přepisu.

Práce se dále věnuje návrhu architektury systému a analýze současných přístupů k automatickému rozpoznávání řeči, včetně praktických aspektů implementace, jako je správa modelů a optimalizace výkonu.

Struktura práce je následující: nejprve je provedeno srovnání platforem Umbrel a Start9 a zpracován přehled existujících řešení, následně je popsán návrh a implementace aplikace včetně použitých modelů. Závěrečná část se zabývá testováním a možnostmi dalšího rozvoje, jako je podpora dalších modelů, rozšíření formátů výstupu, sumarizace, překlad či přepis v reálném čase.

Odkaz na Umbrel obchod: `https://github.com/danieldedek/shop` a odkaz na aplikaci `https://github.com/danieldedek/transcribe_app`.

Kapitola 2

Umbrel/Start9

V rámci návrhu řešení bylo nutné rozhodnout, pro jakou cílovou platformu bude aplikace určena. Na výběr byly k dispozici platformy Umbrel a Start9, které představují moderní prostředí pro provoz self-hosted služeb s důrazem na soukromí a jednoduchou správu aplikací. Obě platformy nabízejí podobné možnosti z hlediska nasazení kontejnerizovaných aplikací, liší se však v přístupu k instalaci, správě služeb a uživatelskému rozhraní.

2.1 Umbrel

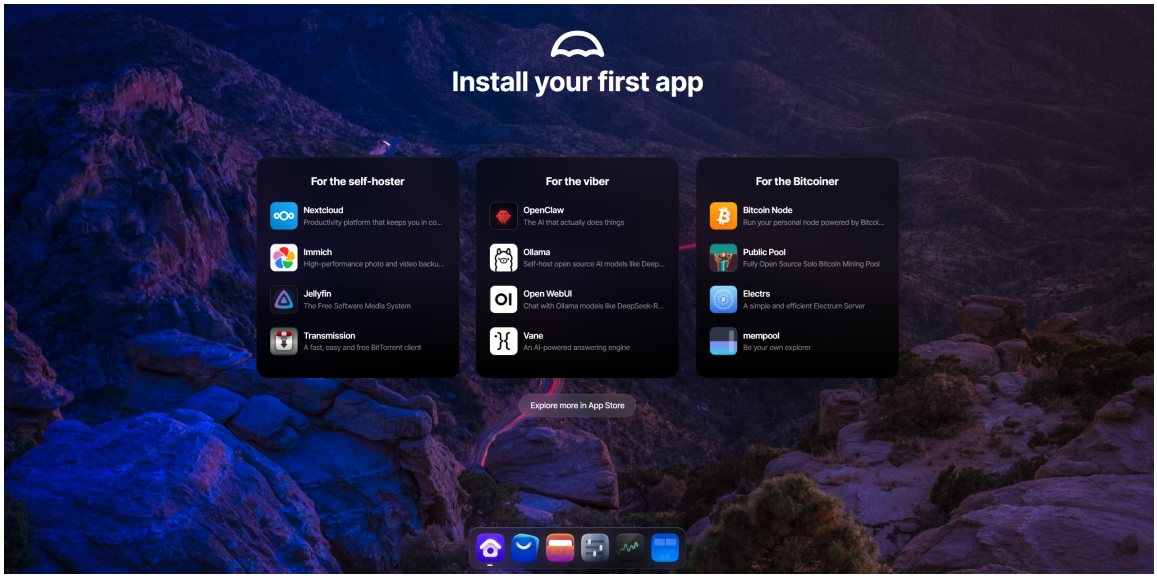
Umbrel je open-source platforma určená pro provoz osobního serveru, kterou lze nainstalovat např. na zařízení jako Raspberry Pi nebo běžný počítač. Je navržena s důrazem na jednoduchost, přehlednost a uživatelskou přívětivost. Hlavním cílem Umbrelu je umožnit i méně technicky zdatným uživatelům provozovat vlastní služby bez nutnosti pokročilých znalostí správy systému.[11]

Umbrel poskytuje integrovaný katalog aplikací, který obsahuje desítky až nižší stovky aplikací různých typů. Mezi nejčastější patří:

- plnohodnotný uzel sítě Bitcoin,
- uzel sítě Lightning Network,
- cloudové úložiště (např. Nextcloud),
- multimediální server,
- VPN služby,
- nástroje pro správu souborů a zálohování.

Jedním z klíčových prvků Umbrelu je jeho grafické webové rozhraní, které umožňuje správu systému přímo z prohlížeče. Instalace aplikací probíhá jedním kliknutím a systém automaticky řeší jejich konfiguraci a správu. Umbrel využívá kontejnerizaci (Docker), což zajišťuje izolaci jednotlivých aplikací a usnadňuje jejich aktualizaci.

Platforma klade důraz na jednoduchost použití a estetiku uživatelského prostředí, díky čemuž je vhodná zejména pro začátečníky nebo uživatele, kteří chtějí rychle zprovoznit vlastní server bez hlubších znalostí Linuxu nebo síťové administrace.



Obrázek 2.1: Ukázka platformy Umbrel

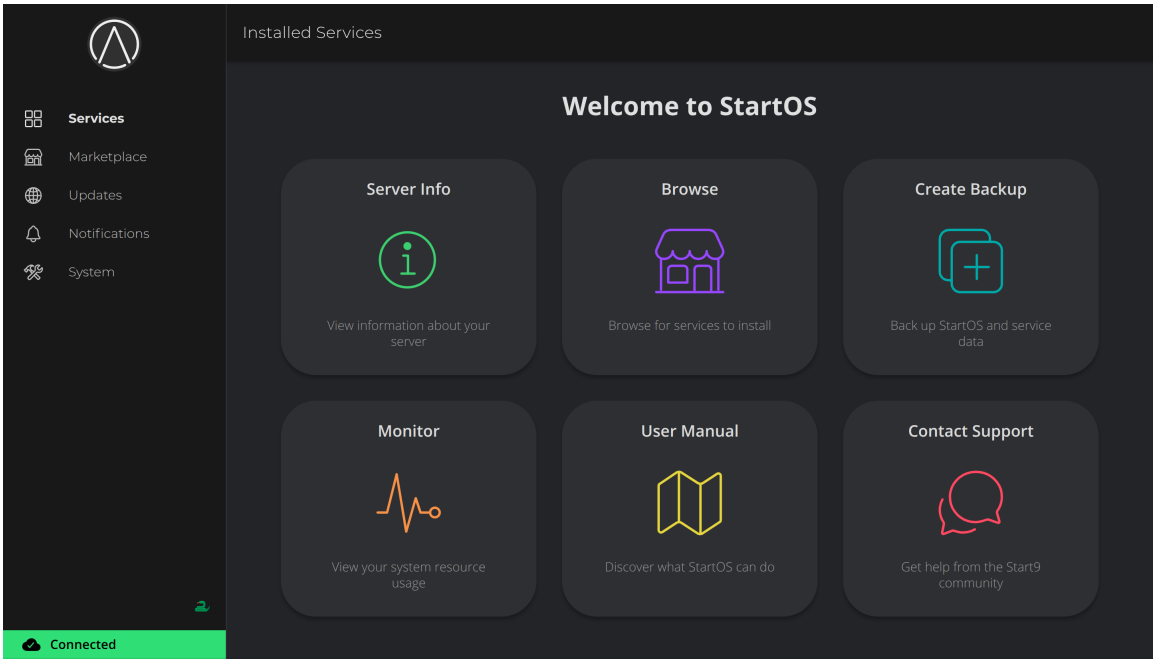
2.2 Start9

Start9 je pokročilejší platforma zaměřená na provoz suverénních serverových řešení s důrazem na maximální kontrolu, bezpečnost a decentralizaci. Operační systém Start9 (EmbassyOS) poskytuje robustní infrastrukturu pro běh různých služeb v prostředí, kde má uživatel plnou kontrolu nad svými daty.[8]

Stejně jako Umbrel nabízí Start9 vlastní katalog aplikací, který obsahuje desítky aplikací zaměřených především na bezpečnost, decentralizaci a self-hosting. Mezi typické aplikace patří:

- uzly decentralizovaných sítí (např. Bitcoin, Lightning),
- komunikační nástroje zaměřené na soukromí,
- nástroje pro správu identity a šifrování,
- cloudové a datové služby,
- vývojářské a administrátorské nástroje.

Na rozdíl od Umbrelu je Start9 více orientován na technicky zdatnější uživatele. Nabízí širší možnosti konfigurace a hlubší kontrolu nad systémem. Start9 je často provozován na dedikovaném hardwaru dodávaném výrobcem, ale lze jej nainstalovat i na vlastní zařízení. Oproti Umbrelu má méně intuitivní uživatelské rozhraní, avšak poskytuje výrazně vyšší flexibilitu a kontrolu.



Obrázek 2.2: Ukázka platformy Start9

2.3 Porovnání a důvody volby platformy Umbrel

Přestože oba systémy mají podobný cíl, kterým je umožnit uživatelům provozovat vlastní server a získat kontrolu nad svými daty, liší se v přístupu, cílové skupině i uživatelském komfortu.

2.3.1 Uživatelská přívětivost

Jedním z hlavních důvodů, proč byl zvolen Umbrel, je jeho výrazně vyšší uživatelská přívětivost. Umbrel nabízí moderní a intuitivní grafické rozhraní, které umožňuje snadnou orientaci i začátečníkům. Instalace aplikací je otázkou jednoho kliknutí a nevyžaduje hlubší technické znalosti.

Naopak Start9 je více zaměřen na pokročilé uživatele, což se projevuje složitějším rozhraním a nutností rozumět některým technickým aspektům systému.

2.3.2 Jednoduchost instalace a správy

Umbrel je navržen tak, aby jeho instalace byla co nejjednodušší. Existují předpřipravené obrazy systému a detailní návody, díky nimž lze systém zprovoznit během krátké doby.

Start9 může vyžadovat složitější konfiguraci a hlubší pochopení systému.

2.3.3 Ekosystém aplikací

Umbrel disponuje dobře organizovaným obchodem, který obsahuje široké spektrum aplikací. Tyto aplikace jsou optimalizované pro snadnou instalaci a správu.

Start9 sice také nabízí vlastní aplikace, ale jejich instalace a správa může být méně intuitivní a někdy vyžaduje více zásahů ze strany uživatele.

2.3.4 Komunita a dokumentace

Umbrel má aktivní komunitu a dobře zpracovanou dokumentaci, která usnadňuje řešení problémů a implementaci nových funkcí. To je důležité zejména při práci na projektu, kde je potřeba rychle nalézt řešení případných komplikací.[3]

2.4 Shrnutí důvodů volby Umbrelu

Na základě výše uvedeného lze shrnout hlavní důvody, proč byl pro tuto práci zvolen právě Umbrel:

- jednodušší instalace a konfigurace,
- přehledné a moderní uživatelské rozhraní,
- široká nabídka aplikací dostupných jedním kliknutím,
- vhodnost pro začátečníky i středně pokročilé uživatele,
- lepší využitelnost pro demonstrační a vzdělávací účely,
- aktivní komunita a dostupná dokumentace.

Z těchto důvodů lze Umbrel považovat za praktičtější a efektivnější řešení pro účely této bakalářské práce, zejména pokud je cílem prezentovat funkční a snadno použitelné řešení osobního serveru bez nutnosti složité konfigurace.[10]

Kapitola 3

Průzkum podobných aplikací

V současnosti existuje široká škála nástrojů a služeb zaměřených na automatický přepis řeči do textu. Tyto systémy se liší nejen použitými technologiemi a architekturami neuronových sítí, ale také způsobem nasazení, dostupností, podporovanými jazyky a doplňkovými funkcemi. Z hlediska způsobu provozu lze tato řešení rozdělit na cloudové služby, které využívají vzdálenou infrastrukturu poskytovatelů, a lokální aplikace, jež umožňují zpracování dat přímo na zařízení uživatele. Vedle hotových aplikací existují také frameworky a API rozhraní, která umožňují integraci rozpoznávání řeči do vlastních systémů. Tato kapitola se zaměřuje na analýzu vybraných zástupců jednotlivých kategorií s cílem identifikovat jejich vlastnosti a výhody, a vytvořit tak podklad pro návrh vlastního řešení.

3.1 Aplikace umožňující přepis řeči do textu

3.1.1 Cloudové aplikace

V rámci průzkumu existujících řešení zaměřených na automatický přepis řeči na text je vhodné analyzovat dostupné aplikace, které tuto funkcionalitu poskytují. Tyto nástroje využívají pokročilé metody z oblasti zpracování přirozeného jazyka a rozpoznávání řeči k převodu mluveného slova do textové podoby.

Mezi významné zástupce těchto aplikací patří např. SpokenData, Beey.io, Otter.ai, Rev, Sonix.ai a Fireflies.ai. Tyto nástroje se liší především v přesnosti přepisu, jazykové podpoře, možnostech integrace s dalšími systémy a doplňkových funkcích, jako je např. identifikace mluvčích, automatické shrnutí nebo vyhledávání v přepisech.

Důležitým aspektem při hodnocení těchto aplikací je také způsob zpracování dat a otázka ochrany soukromí uživatelů. Výše uvedená řešení fungují jako cloudové služby, což znamená, že audio data jsou odesílána na vzdálené servery k dalšímu zpracování. Tento přístup může být z hlediska výkonu a dostupnosti výhodný, avšak zároveň přináší určité riziko z pohledu bezpečnosti a kontroly nad daty.

SpokenData

SpokenData je online platforma zaměřená na převod mluvené řeči na text s důrazem na praktické využití pro různé uživatele. Podporuje nahrávání zvukových a video souborů a generuje přepis, který si lze stáhnout nebo dále zpracovat.

SpokenData umožňuje nahrát prakticky libovolné audio či video. Stačí je vložit prostřednictvím webového nahrání nebo zadáním URL. Platforma automaticky převádí řeč na

text a umožňuje stáhnout výsledný přepis ve formátech SRT, TXT nebo XML. Podporované formáty zahrnují běžné audio/video formáty i YouTube video odkazy. Po nahrání systém provede automatické rozpoznání řeči a po dokončení procesu upozorní uživatele.

SpokenData obsahuje online editor, kde lze s přepisem pracovat:

- upravovat text ručně,
- označovat důležité části,
- přidávat časové značky,
- strukturovat výstup podle mluvčích.

Beey.io

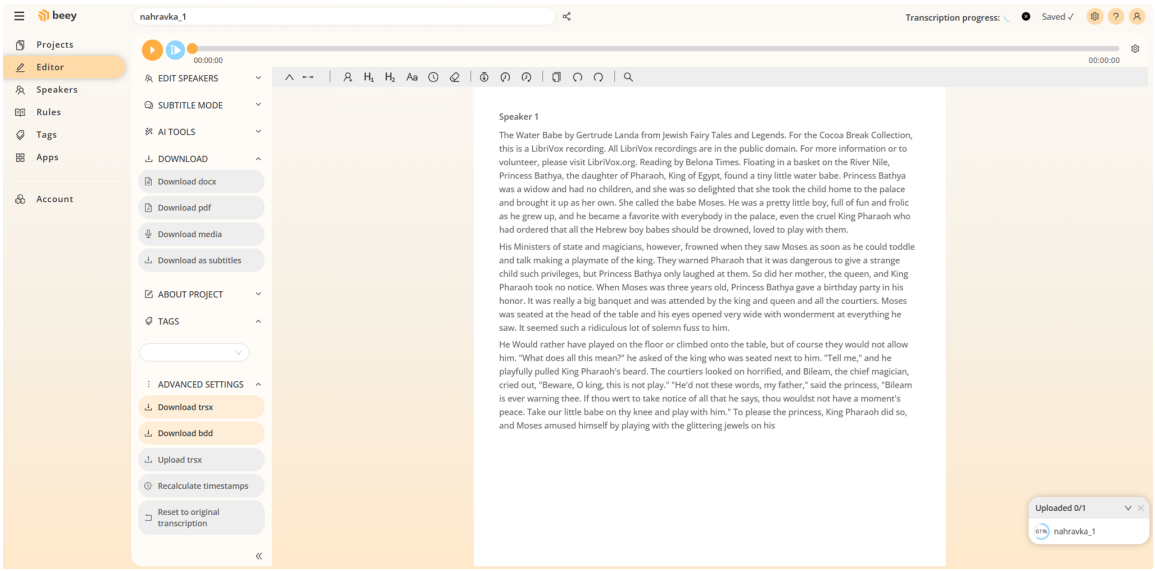
Beey.io je moderní AI nástroj, který se zaměřuje na rychlý a přesný přepis audio i video obsahu a přidává řadu doplňkových funkcí pro editaci, titulkování či živý přepis.

Beey.io umožňuje standardní nahrání audio a video souborů (např. MP3, WAV, M4A, MP4) a následný automatický převod na text. Po nahrání souboru se přepis spustí okamžitě nebo po dokončení nahrání, a uživatel může sledovat průběh přepisu v reálném čase.

V editoru přepisu se během zpracování postupně zobrazují textové části, které lze hned upravovat. Tento přístup šetří čas, protože úpravy je možno provádět ještě během přepisování. Beey.io nabízí i živý přepis řeči z mikrofonu. Stačí zvolit jazyk a začít nahrávat přímo přes webové rozhraní.

Beey.io přidává do editoru schopnosti:

- rozpoznat jednotlivé mluvčí,
- automaticky opravovat chyby,
- umožnit hromadné úpravy více projektů současně.



Obrázek 3.1: Ukázka aplikace Beey.io

Otter.ai

Otter.ai je jedním z nejpopulárnějších nástrojů pro automatický přepis schůzek, přednášek a rozhovorů, zejména v prostředí týmové spolupráce a videokonferencí.

Otter.ai podporuje import audio a video souborů, jejich přepis a dále ještě přepis v reálném čase, pokud je nástroj připojen ke schůzce přes Zoom, Google Meet nebo Microsoft Teams.

Přepis během schůzky lze vidět okamžitě v prohlížeči či v mobilní aplikaci. To je zásadní výhoda, pokud chce uživatel mít okamžitý textový záznam bez čekání na konečné zpracování.

Po převedení řeči na text jsou k dispozici následující možnosti:

- snadné úpravy přepisu,
- označování mluvčích,
- přidávání komentářů a značek,
- vyhledávání ve výsledném textu,
- export do různých formátů.

Otter.ai je tedy silný nejen čistým přepisem, ale i možnostmi organizace a spolupráce nad textem. Jednou z nejzajímavějších funkcí je automatická sumarizace schůzek a možnost vytažení důležitých témat, úkolů a klíčových frází z přepisu bez nutnosti jejich ručního vyhledávání. To je obrovská výhoda při práci s dlouhými meetingy či rozhovory. Uživatel nemusí číst celý text a umělá inteligence mu nabídne přehled nejpodstatnějších momentů.

Rev

Rev je zavedené řešení, které kombinuje rychlý automatický přepis s možností profesionálních lidských přepisů. Automatické přepisy jsou rychlé a cenově dostupné, lidské přepisy zase extrémně přesné. Ty jsou vhodné např. pro právní dokumenty nebo obsah určený pro publikaci či archivaci.

Rev poskytuje vlastní editor přepisů s možností:

- označovat časové značky,
- vyhledávat uvnitř textu,
- ručně opravovat chyby,
- přidávat značky a komentáře.

Navíc zde je možnost přidat vlastní slovník, což pomáhá zlepšit rozpoznávání specifických termínů, názvů či technických výrazů, které běžně systém automatického rozpoznávání řeči nezná. Rev má i funkce, kdy umělá inteligence z přepisu generuje stručné shrnutí.

Sonix.ai

Sonix.ai je profesionální nástroj pro přepis, překlad, tvorbu titulků a analýzu audio a video obsahu, který vyniká svou multijazykovou podporou. Sonix.ai podporuje širokou škálu jazyků, rychlý upload audio a video souborů, automatickou transkripci s detekcí mluvčích a časových kódů.

Sonix.ai nabízí robustní editor, kde lze:

- přepis opravovat přímo v textu,
- přidávat anotace a značky,
- vytvářet automatické titulky,
- překládat přepisy do jiných jazyků.

To je obrovská výhoda, pokud uživatel pracuje s mezinárodním obsahem nebo potřebuješ vysvětlovat audio materiál v jiném jazyce. Sonix.ai funguje spíše jako profesionální produktivní platforma než jen jednoduchý přepisovací nástroj.

Fireflies.ai

Fireflies.ai je nástroj zaměřený na automatické záznamy a přepisy online schůzek, který se integruje s populárními konferenčními službami jako Zoom, Google Meet či Teams.

Fireflies.ai dokáže automaticky připojit robota k probíhající schůzce, zaznamenat audio a přepisovat text v reálném čase. To je výhodné v prostředích, kde se často konají videokonference a je potřeba mít textový záznam bez manuálního přepínání nástrojů.

Po schůzce lze:

- upravovat přepis v editoru,
- označit klíčové momenty,
- získat souhrn toho nejdůležitějšího.

Ve srovnání s některými jinými nástroji je přesnost občas nižší, ale Fireflies.ai kompenzuje tuto nevýhodu právě integrací a automatizací schůzek.

3.1.2 Cloudová API

Kromě hotových aplikací určených pro koncové uživatele existují také cloudové služby poskytující rozpoznávání řeči ve formě programového rozhraní (API). Tyto služby jsou určeny především pro vývojáře, kteří potřebují integrovat přepis řeči do vlastních aplikací nebo systémů.

Mezi nejvýznamnější poskytovatele patří Microsoft Azure Speech-to-Text, Amazon AWS Transcribe a Google Speech-to-Text. Tyto platformy umožňují zpracování audio dat prostřednictvím REST API nebo SDK, přičemž podporují širokou škálu jazyků, formátů a pokročilých funkcí, jako jsou časové značky, identifikace mluvčích nebo průběžný přepis.

Hlavní výhodou těchto služeb je vysoká přesnost, škálovatelnost a minimální nároky na lokální hardware. Uživatel nemusí řešit instalaci modelů ani jejich optimalizaci, protože veškeré zpracování probíhá na infrastruktuře poskytovatele. Nevýhodou je však nutnost odesílání audio dat na vzdálené servery, což může představovat riziko z hlediska ochrany soukromí a bezpečnosti dat. Dalším omezením mohou být provozní náklady, které rostou s objemem zpracovaných dat.

Cloudová API tak představují vhodné řešení zejména pro komerční aplikace a systémy vyžadující vysokou dostupnost a jednoduchou integraci, avšak méně vhodné pro scénáře, kde je klíčová kontrola nad daty a možnost offline provozu.

Microsoft Azure Speech-to-Text

Microsoft Azure Speech-to-Text je cloudová služba pro automatické rozpoznávání řeči poskytovaná v rámci platformy Microsoft Azure. Umožňuje převod mluveného slova do textu v reálném čase i dávkovém režimu prostřednictvím REST API nebo SDK pro různé programovací jazyky.

Služba podporuje široké spektrum jazyků a nabízí pokročilé funkce, jako jsou časové značky, identifikace mluvčích, přizpůsobení modelu pomocí vlastních dat nebo filtrování nevhodného obsahu. Výhodou je také snadná integrace s dalšími službami Azure.

Amazon AWS Transcribe

Amazon AWS Transcribe je služba pro rozpoznávání řeči dostupná v rámci cloudové platformy Amazon Web Services. Umožňuje automatický přepis audio souborů i streamovaného zvuku a je navržena pro snadnou integraci do škálovatelných aplikací.

Služba nabízí funkce jako identifikace mluvčích, detekce jazyků, generování časových značek nebo automatická interpunkce. AWS Transcribe také umožňuje rozšíření o specializované slovníky, které zlepšují přesnost při práci s doménově specifickými termíny. Výhodou je vysoká škálovatelnost a integrace s dalšími AWS službami.

Google Speech-to-Text

Google Speech-to-Text je cloudová služba společnosti Google, která poskytuje rozpoznávání řeči pomocí pokročilých modelů strojového učení. Umožňuje převod řeči na text v reálném čase i z uložených audio souborů.

Služba podporuje velké množství jazyků a nabízí funkce jako automatická interpunkce, časové značky, rozpoznávání více mluvčích a adaptaci modelu pomocí specifických frází. Google Speech-to-Text je známý svou vysokou přesností a rychlou odezvou.

3.1.3 Lokální aplikace

Vedle cloudových řešení existují také lokální alternativy, které umožňují zpracování řeči přímo na zařízení uživatele bez nutnosti připojení k internetu. Mezi tyto přístupy patří např. aplikace jako Murmure, Handy nebo Vocalinux. Tyto nástroje kladou důraz především na ochranu soukromí, protože audio data neopouštějí zařízení uživatele, a zároveň umožňují jejich využití i v prostředích bez stabilního internetového připojení.

Cílem této kapitoly je analyzovat vybrané aplikace, porovnat jejich vlastnosti a identifikovat jejich silné a slabé stránky napříč cloudovými i lokálními řešeními. Získané poznatky následně poslouží jako podklad pro návrh vlastního řešení, které bude reflektovat zjištěné nedostatky a zároveň využívat osvědčené přístupy existujících nástrojů.

Murmure

Murmure je lokální aplikace zaměřená na převod řeči na text s důrazem na soukromí a offline zpracování. Celý proces přepisu probíhá přímo na zařízení uživatele, bez nutnosti odesílání dat na externí servery. Aplikace je navržena tak, aby byla jednoduchá na použití a zároveň poskytovala dostatečně kvalitní výsledky pro běžné i profesionální využití.

Mezi hlavní přednosti nástroje Murmure patří:

- plně offline zpracování (vyšší úroveň soukromí),

- žádné odesílání audio dat do cloudu,
- jednoduché uživatelské rozhraní,
- vhodné pro citlivé záznamy (např. rozhovory, interní schůzky),
- podporuje více jazyků včetně běžně používaných evropských jazyků.

Handy

Desktopová aplikace Handy je postavená na technologii Whisper, která umožňuje rychlé diktování a přepis řeči přímo do textových polí v operačním systému. Funguje jako jednoduchý nástroj pro okamžitý převod hlasu na text bez nutnosti otevírání webového rozhraní.

Nástroj Handy nabízí následující výhody:

- velmi jednoduché ovládání (aktivace klávesovou zkratkou),
- okamžité vkládání textu do libovolné aplikace,
- využívá model Whisper pro vysokou přesnost,
- plně lokální zpracování bez internetu,
- vhodné pro každodenní diktování textu.

Vocalinux

Vocalinux je open-source nástroj pro hlasové diktování v prostředí Linuxu. Umožňuje převod řeči na text v reálném čase a je navržen tak, aby byl plně offline a integrovatelný do linuxových desktopových prostředí.

Mezi výhody Vocalinux patří:

- kompletně offline provoz,
- optimalizace pro Linux prostředí,
- možnost přepisu v reálném čase,
- nízké nároky při použití lehčích modelů,
- vhodné pro vývojáře a pokročilé uživatele,
- možnost integrace do vlastních skriptů a workflow.

3.1.4 Frameworky a nástroje pro ASR

Frameworky a nástroje pro automatické rozpoznávání řeči představují důležitou kategorii řešení určenou především pro vývojáře a výzkumníky. Na rozdíl od hotových aplikací, které jsou zaměřeny na koncové uživatele, tyto nástroje poskytují nízkoúrovňový i vysokoúrovňový přístup k jednotlivým částem procesu zpracování řeči, jako je příprava dat, trénování modelů, dekódování nebo post-processing výsledků. Díky tomu umožňují vytvářet vlastní ASR systémy, přizpůsobovat je specifickým doménám a experimentovat s různými modelovými architekturami.

Tato řešení se liší především mírou abstrakce a složitosti použití. Zatímco starší nástroje kladou důraz na maximální flexibilitu a detailní kontrolu nad jednotlivými kroky zpracování, modernější frameworky se snaží celý proces zjednodušit a zpřístupnit širšímu spektru uživatelů prostřednictvím předtrénovaných modelů a intuitivnějšího rozhraní. V následující části jsou představeny vybrané nástroje, které reprezentují různé přístupy k vývoji systémů pro rozpoznávání řeči.

Kaldi

Kaldi je open-source toolkit pro rozpoznávání řeči, který patří mezi nejrozšířenější a nejdéle používané nástroje v této oblasti. Je navržen především pro výzkumné a akademické účely, ale využívá se i v průmyslových aplikacích. Kaldi poskytuje rozsáhlou podporu pro trénování akustických modelů, zpracování dat a dekódování řeči.

Výhodou Kaldi je vysoká flexibilita a možnost detailního nastavení jednotlivých částí pipeline. Umožňuje implementaci klasických i moderních přístupů k rozpoznávání řeči.

k2

k2 je moderní open-source knihovna pro práci s konečnými automaty, která je často využívána jako nástupce nebo doplněk ke Kaldi. Je navržena s důrazem na efektivitu, modularitu a integraci s frameworkem PyTorch.

k2 umožňuje implementaci pokročilých dekódovacích algoritmů a efektivní práci s jazykovými modely. V kombinaci s dalšími nástroji tvoří základ pro moderní ASR pipeline. Oproti Kaldi je více orientována na vývojáře pracující s hlubokým učením, avšak stále vyžaduje pokročilé znalosti problematiky.

NVIDIA NeMo

NVIDIA NeMo je open-source framework určený pro vývoj a nasazení modelů v oblasti zpracování řeči a přirozeného jazyka. Poskytuje předtrénované modely i nástroje pro jejich další trénování a ladění. NeMo je postaven nad knihovnou PyTorch a je optimalizován pro běh na grafických kartách.

Výhodou NeMo je relativně snadné použití ve srovnání s tradičními nástroji, jako je Kaldi, a zároveň vysoký výkon díky využití moderních architektur neuronových sítí. Umožňuje rychlé nasazení modelů pro rozpoznávání řeči, syntézu hlasu i další úlohy. Nevýhodou mohou být vyšší hardwarové nároky, zejména při využití GPU.

Hugging Face

Hugging Face představuje populární ekosystém nástrojů a modelů pro zpracování přirozeného jazyka a řeči. Knihovna Transformers poskytuje přístup k široké škále předtrénovaných modelů, včetně modelů pro rozpoznávání řeči, jako je např. Whisper.

Velkou výhodou Hugging Face je jednoduchá integrace modelů do aplikací a rozsáhlá komunita, která sdílí modely i datasetové zdroje. Platforma umožňuje jak cloudové, tak lokální nasazení modelů.

3.1.5 Shrnutí průzkumu

Na základě analýzy vybraných nástrojů pro automatický přepis řeči na text lze konstatovat, že současná řešení lze rozdělit do tří hlavních kategorií: hotové aplikace pro koncové uživatele, cloudová API a frameworky pro vývoj vlastních systémů.

Hotové aplikace, jako např. SpokenData, Beey.io, Otter.ai, Rev, Sonix.ai a Fireflies.ai, nabízejí široké spektrum funkcí a různých přístupů k řešení této problematiky. Zároveň však existují i lokální alternativy, např. Murmure, Handy a Vocalinux, které představují odlišný přístup založený na offline zpracování dat.

Společným znakem těchto aplikací je využití metod automatického rozpoznávání řeči a jejich zaměření na zjednodušení práce s audio a video obsahem. Většina nástrojů umožňuje nejen samotný přepis, ale také jeho následnou editaci, strukturování, export do různých formátů a v některých případech i pokročilé funkce, jako je identifikace mluvčích, automatická sumarizace nebo překlad do jiných jazyků.

Vedle těchto aplikací existují také cloudová API, jako jsou Microsoft Azure Speech-to-Text, Amazon AWS Transcribe a Google Speech-to-Text. Tyto služby umožňují integraci přepisu řeči přímo do vlastních aplikací prostřednictvím programového rozhraní. Jejich hlavní výhodou je vysoká přesnost, škálovatelnost a jednoduchá integrace bez nutnosti řešit infrastrukturu nebo modely. Nevýhodou je však závislost na externích službách, provozní náklady a nutnost odesílání dat na vzdálené servery, což může být problematické z hlediska soukromí.

Další významnou kategorií jsou frameworky a nástroje pro vývoj vlastních ASR systémů, jako např. Kaldi, k2, NVIDIA NeMo nebo Hugging Face. Tyto nástroje poskytují větší flexibilitu a kontrolu nad celým procesem zpracování řeči, od přípravy dat až po samotné modely a dekódování. Umožňují jak experimentování s různými přístupy, tak i plně lokální nasazení. Jejich nevýhodou je však vyšší složitost a náročnost na znalosti i výpočetní prostředky.

Jednotlivé přístupy se tedy liší především mírou kontroly, jednoduchostí použití a požadavky na infrastrukturu. Cloudová řešení se orientují na jednoduchost a dostupnost, zatímco lokální aplikace a frameworky kladou důraz na soukromí, flexibilitu a nezávislost.

Z hlediska současných trendů je patrný posun směrem k automatizaci, integraci do pracovních nástrojů a zjednodušení uživatelského rozhraní. Současně však roste význam lokálních řešení, která reagují na rostoucí požadavky na ochranu soukromí a kontrolu nad daty.

Tato analýza ukazuje, že ačkoliv jsou současné nástroje velmi pokročilé, stále existuje prostor pro zlepšení, zejména v oblasti kombinace jednoduchosti použití, přesnosti přepisu, podpory méně rozšířených jazyků a možnosti plně lokálního provozu. Tyto poznatky slouží jako důležitý podklad pro návrh vlastního systému, který se snaží propojit výhody jednotlivých přístupů a minimalizovat jejich nedostatky.

3.2 Open WebUI a Ollama

Mezi nástroje, které významně přispívají k popularizaci lokálních LLM (Large Language Model), patří Ollama, což je jednoduchá platforma pro spouštění jazykových modelů na vlastním počítači, a Open WebUI, což je webové uživatelské rozhraní, které nad Ollamou a dalšími backendy poskytuje komfortní prostředí pro interakci s modely.

Ollama je nástroj určený pro jednoduché spouštění a správu velkých jazykových modelů lokálně. Jeho hlavní výhodou je abstrakce složitosti spojené s instalací, konfigurací a

optimalizací LLM. Uživatel nemusí řešit detailní nastavení knihoven, modelových vah ani způsob jejich načítání. Vše je řízeno prostřednictvím jednoduchých příkazů.

Modely jsou optimalizovány pro běh na běžném hardwaru, často ve formě kvantizovaných variant, které snižují paměťové nároky. Díky tomu je možné provozovat LLM i na osobních počítačích nebo pracovních stanicích bez nutnosti výkonného serverového vybavení.

Open WebUI je open-source webové uživatelské rozhraní určené pro práci s jazykovými modely. Původně vzniklo jako jednoduché rozhraní pro Ollamu, postupně se však vyvinulo v komplexní platformu, která podporuje více backendů (OpenAI API, LM Studio apod.).

Hlavním cílem Open WebUI je zpřístupnit práci s LLM i netechnickým uživatelům a současně nabídnout pokročilé funkce pro expertní použití. Open WebUI využívá Ollamu jako zdroj jazykových modelů. Po spuštění Ollamy jako lokální služby je Open WebUI schopné automaticky detekovat dostupné modely, zobrazit je v seznamu modelů a přepínat mezi nimi bez nutnosti restartu aplikace. Tím je dosaženo výrazné flexibility. Uživatel může v rámci jednoho rozhraní testovat různé modely a porovnávat jejich chování.

V této webové aplikaci lze pro každý chat nebo konverzaci zvolit konkrétní model běžící v Ollamě. To je výhodné např. v situacích, kdy jeden model je optimalizovaný pro obecnou konverzaci, jiný model je specializovaný na programování, nebo další je vhodný pro shrnování nebo analýzu textů. Open WebUI předává dotazy Ollamě ve strukturované podobě a výslednou odpověď zobrazuje uživateli.

Jednou z hlavních výhod kombinace Open WebUI a Ollamy je plná kontrola nad daty. Veškeré dotazy i odpovědi zůstávají lokálně. To znamená, že data nejsou odesílána do cloudu, nehrozí únik citlivých informací a řešení je vhodné i pro firemní nebo akademické prostředí. Tento aspekt je často klíčovým důvodem, proč uživatelé volí lokální AI řešení místo cloudových služeb.[2]

3.2.1 Možnost integrace ASR do Open WebUI

V rámci návrhu řešení byla zvažována také možnost rozšíření Open WebUI o podporu automatického rozpoznávání řeči. Tato varianta by spočívala v integraci přepisu řeči přímo do existujícího rozhraní pro práci s jazykovými modely, čímž by vzniklo jednotné prostředí pro zpracování jak textových, tak hlasových vstupů.

Tento přístup však nebyl zvolen z několika důvodů. Open WebUI je primárně navrženo pro práci s velkými jazykovými modely a jeho architektura není optimalizována pro zpracování audio dat. Integrace ASR by vyžadovala zásah do backendové logiky aplikace, rozšíření o podporu nahrávání a zpracování audio souborů a řešení závislostí spojených s knihovnami pro zpracování řeči.

Dalším důvodem je odlišný charakter ASR modelů oproti LLM. Modely pro rozpoznávání řeči mají specifické požadavky na vstupní data, výpočetní prostředky a způsob inferenčního zpracování, což by vedlo ke zvýšení komplexity celého systému. Oddělení ASR aplikace od prostředí Open WebUI proto umožňuje jednodušší návrh, lepší kontrolu nad zpracováním dat a snadnější ladění jednotlivých komponent.

Zároveň toto řešení poskytuje větší flexibilitu, protože ASR aplikace může být navržena nezávisle na konkrétním rozhraní pro práci s jazykovými modely a může být v budoucnu integrována s různými systémy, včetně Open WebUI, prostřednictvím API rozhraní.

3.3 Shrnutí zajímavých funkcí podobných aplikací

Moderní aplikace pro automatický přepis řeči a práci s audio-vizuálním obsahem nabízejí širokou škálu funkcí, které výrazně usnadňují zpracování mluveného projevu. Mezi nejdůležitější a nejčastěji využívané funkce patří nahrání souboru, přepis v reálném čase, úprava přepisu a automatická sumarizace obsahu. Následující funkce mi přišli jako nejužitečnější.

3.3.1 Nahrání souboru

Základní funkcí většiny podobných aplikací je možnost nahrání audio nebo video souboru a jeho následné automatické přepsání do textové podoby. Uživatelé mohou zpracovávat nahrávky ve standardních formátech, jako jsou MP3, WAV nebo MP4. Tato funkce umožňuje zpětné zpracování již existujících záznamů, např. přednášek, rozhovorů, podcastů nebo pracovních schůzek. Významnou výhodou je automatizace procesu přepisu, která nahrazuje časově náročné ruční přepisování a snižuje riziko lidských chyb.

3.3.2 Přepis v reálném čase

Další důležitou funkcí je přepis řeči v reálném čase, kdy aplikace převádí mluvené slovo na text okamžitě během samotného záznamu. Tento přístup je využíván zejména při online schůzkách, přednáškách nebo živých prezentacích, kde je potřeba mít textový výstup k dispozici bez časového zpoždění. Přepis v reálném čase zvyšuje dostupnost obsahu, např. pro osoby se sluchovým postižením, a umožňuje uživatelům lépe sledovat strukturu a hlavní myšlenky probíraného tématu.

3.3.3 Upravení přepisu

Automaticky vytvořený přepis zpravidla vyžaduje následné úpravy, které většina aplikací umožňuje prostřednictvím integrovaného textového editoru. Uživatel má možnost opravovat chybně rozpoznaná slova, upravovat interpunkci, rozdělovat text do přehledných odstavců nebo označovat jednotlivé mluvčí. Možnost manuální editace je klíčová zejména u odborných textů, kde je kladen důraz na přesnost terminologie a srozumitelnost výsledného dokumentu. Některé nástroje navíc podporují synchronizaci textu s časovou osou nahrávky, což usnadňuje orientaci v přepisu.

3.3.4 Sumarizace

Stále častěji bývá součástí těchto aplikací také funkce automatické sumarizace, která z přepisu vytváří stručné shrnutí hlavních myšlenek a klíčových informací. Sumarizace je přínosná zejména u dlouhých nahrávek, jako jsou porady, rozhovory nebo přednášky, kde umožňuje rychlé seznámení s obsahem bez nutnosti číst celý přepis. Některé nástroje dokáží navíc identifikovat akční body nebo důležitá rozhodnutí, což zvyšuje praktickou využitelnost výsledného shrnutí.

3.3.5 Nabídka různých ASR modelů

Moderní aplikace pro automatické rozpoznávání řeči často umožňují uživatelům volbu mezi několika dostupnými ASR modely, což zvyšuje flexibilitu a přesnost přepisu v různých situacích. Různé modely se liší velikostí, rychlostí zpracování, přesností rozpoznávání a specializací na konkrétní jazyky či domény. Díky tomu si uživatel může zvolit model, který

nejlépe odpovídá jeho potřebám, např. pro rychlý přepis krátkých hovorů, detailní přepis odborných přednášek nebo práci s vícekanálovým audio vstupem. Některé platformy navíc poskytují možnost přepínat mezi modely dynamicky, což umožňuje optimalizovat výkon systému v reálném čase a přizpůsobit ho aktuálním podmínkám nahrávky.

3.3.6 Nastavení parametrů modelů

Pokročilejší aplikace pro automatické rozpoznávání řeči umožňují uživatelům nastavovat různé parametry jednotlivých modelů, čímž lze ovlivnit způsob generování výsledného přepisu. Mezi běžně nastavitelné parametry patří např. volba jazyka vstupu, délková penalizace, teplota generování, počet generovaných hypotéz nebo strategie dekódování.

Možnost konfigurace těchto parametrů umožňuje přizpůsobit chování modelu konkrétnímu scénáři použití. Např. zvýšení teploty může vést k rozmanitějším výstupům, zatímco beam search může zlepšit přesnost za cenu vyšší výpočetní náročnosti. Nastavení jazykových parametrů je pak klíčové zejména u vícejazyčných modelů.

Tato flexibilita je důležitá především pro pokročilé uživatele, kteří chtějí optimalizovat kvalitu přepisu v závislosti na typu nahrávky, akustických podmínkách nebo specifické terminologii. Zároveň však může představovat vyšší nároky na znalosti uživatele, a proto některé aplikace nabízejí přednastavené profily nebo doporučené konfigurace pro běžné scénáře použití.

Kapitola 4

Implementace

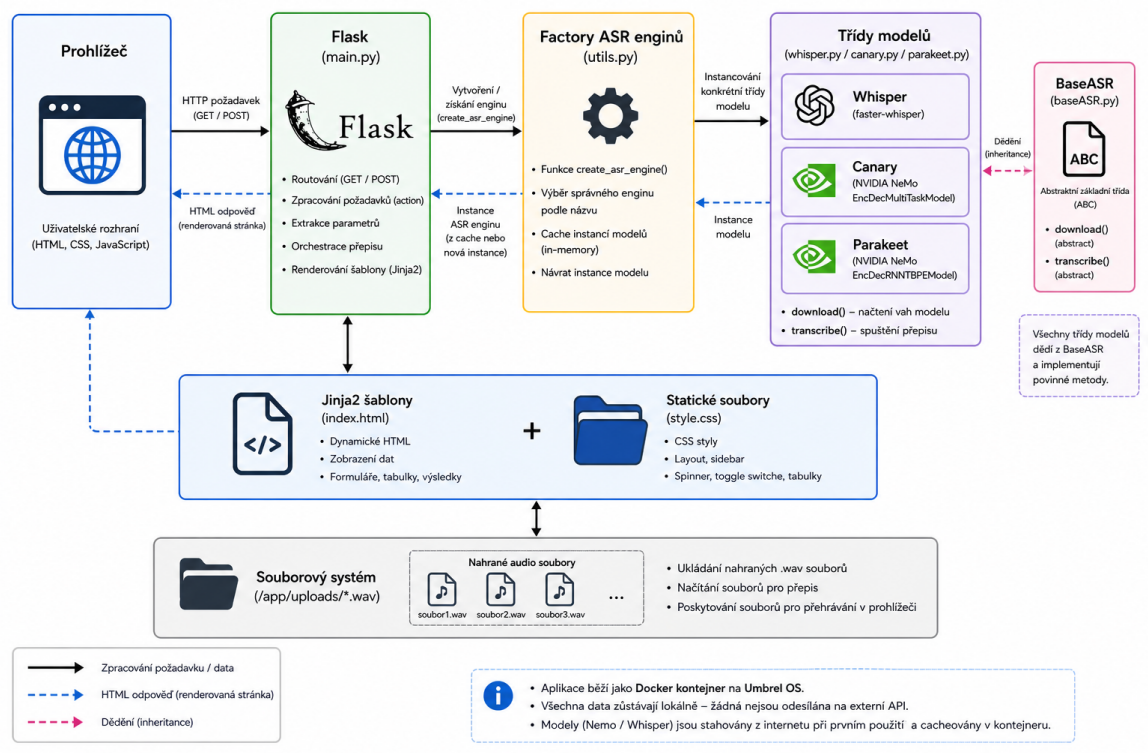
Tato práce se zabývá návrhem a implementací webové aplikace provozované lokálně pro automatický přepis řeči do textu, která umožňuje uživateli převádět zvukové soubory WAV do textové podoby pomocí několika různých modelů strojového učení. Aplikace je navržena tak, aby běžela jako Docker kontejner na platformě Umbrel. Díky tomuto přístupu je řešení plně soběstačné a soukromé, protože žádná audio data nejsou odesílána do externích API.

Navržené řešení využívá modulární architekturu postavenou na webovém frameworku Flask a integruje moderní knihovny pro rozpoznávání řeči, konkrétně NVIDIA NeMo a faster-whisper. Cílem systému je poskytnout flexibilní prostředí pro porovnávání různých ASR modelů a zároveň umožnit jejich snadné rozšiřování. Důležitým návrhovým principem bylo oddělení jednotlivých částí systému tak, aby bylo možné nezávisle upravovat uživatelské rozhraní, aplikační logiku i samotné modely.

4.1 Architektura aplikace

Z hlediska architektury je aplikace rozdělena do několika logických vrstev. Prezentační vrstvu tvoří webové rozhraní implementované pomocí HTML, CSS a šablonovacího systému Jinja2, který je součástí Flask frameworku. Šablony využívají dynamické proměnné předávané z backendu, což umožňuje generovat obsah na základě aktuálního stavu aplikace, např. seznam nahraných souborů nebo výsledky přepisu. Tato vrstva zajišťuje interakci s uživatelem, tedy výběr WAV souboru k přepisu, volbu modelu, nastavování parametrů jednotlivých modelů a zobrazení výsledků přepisu. Uživatelské rozhraní bylo navrženo s důrazem na jednoduchost a přehlednost, aby bylo použitelné i pro méně technicky zdatné uživatele.

Tok dat v aplikaci je znázorněn na následujícím schématu. Uživatel nejprve nahraje WAV soubor, zvolí model a parametry. Aplikace následně zpracuje požadavek, provede přepis pomocí vybraného ASR modelu a vygeneruje výsledek. Ten je zobrazen uživateli přímo v aplikaci, přičemž jej může u modelu Whisper volitelně doplnit o časové značky, segmentaci a skóre spolehlivosti. Výsledný přepis si uživatel může stáhnout ve formátu TXT souboru.



Obrázek 4.1: Schéma aplikace

Aplikační logika je implementována v hlavním souboru `main.py`, kde jsou definovány jednotlivé routy. Hlavní routa `/` zpracovává všechny akce na základě parametru `action` předaného z HTML formuláře. To umožňuje využívat jednu endpoint funkci pro více operací – nahrání souboru, výběr modelu, spuštění transkripce, mazání souborů nebo stažení přepisu. V rámci této funkce dochází k větvení logiky podle typu požadavku.

4.2 Správa souborů

Součástí uživatelského rozhraní je také správa nahraných souborů, jejich přehrávání přímo v prohlížeči a možnost exportu výsledného přepsaného textu v podobě TXT souboru. Rozhraní umožňuje uživateli přehledně pracovat s více nahrávkami současně, přičemž jednotlivé soubory lze vyhledávat, řadit podle názvu nebo data poslední modifikace a opakovaně zpracovávat s různými modely či parametry.

Samotné nahrávání souborů je realizováno pomocí HTML formuláře s metodou POST, který odesílá data na definovanou Flask routu. Na backendu je využita podpora pro práci s `multipart/form-data`, která umožňuje přijmout binární obsah souboru. Po přijetí požadavku je soubor validován – kontroluje se přípona souboru a skutečný obsah WAV souboru prostřednictvím standardní knihovny `wave`. Až po úspěšné validaci je soubor uložen pomocí metody `file.save()` do adresáře `/app/uploads`.

Uložené soubory jsou načítány pomocí standardních operací nad souborovým systémem (`os.listdir()`). Čas poslední modifikace souboru je určen pomocí `os.path.getmtime()`. Tato informace je využita při řazení a filtrování v postranním panelu. Přehrávání souborů je realizováno pomocí HTML5 audio prvku, který přímo odkazuje na soubor uložený na

serveru prostřednictvím routy `/uploads/<filename>`. Stránkování je implementováno na straně serveru s pevným počtem 5 souborů na stránku.

4.3 Abstraktní vrstva modelů

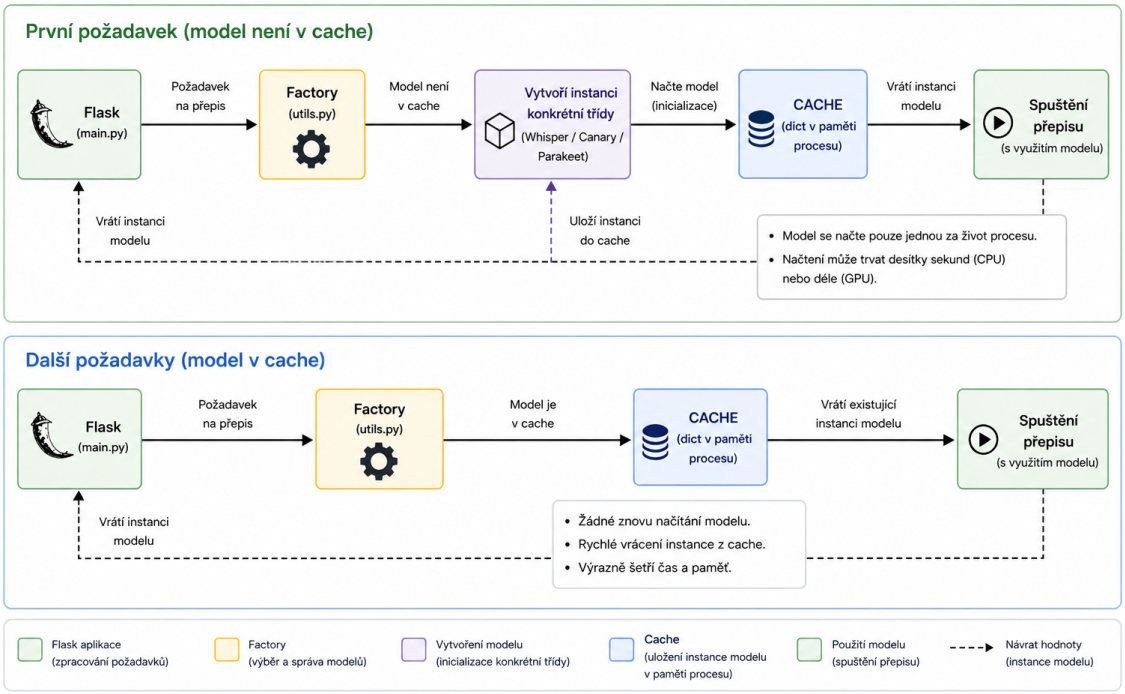
Klíčovou část systému tvoří abstraktní vrstva modelů implementovaná pomocí třídy `BaseASR` v souboru `baseASR.py`. Tato třída definuje pomocí modulu `abc` (Abstract Base Class) dvě abstraktní metody (`download()` a `transcribe()`), které musí být implementovány v každé konkrétní podtřídě. Díky tomuto sjednocení může aplikační logika pracovat s libovolným modelem bez znalosti jeho interní implementace.

Aplikace v současné době integruje tři ASR modely reprezentující různé architektury. Whisper využívá transformerovou architekturu a je implementován prostřednictvím knihovny faster-whisper, která je výrazně rychlejší než původní implementace OpenAI díky využití CTranslate2. Canary je encoder-decoder model od NVIDIA dostupný přes framework NeMo, konkrétně checkpoint `nvidia/canary-180m-flash`. Parakeet využívá architekturu RNN-T (Recurrent Neural Network Transducer), rovněž dostupnou přes NeMo jako checkpoint `nvidia/parakeet-tdt-0.6b-v3`.

Každá třída implementuje metodu `download()`, která zajistí stažení a inicializaci vah modelu, a metodu `transcribe()`, která přijme cestu k WAV souboru a vrátí textový přepis. Whisper navíc podporuje parametr `return_segments=True`, který vrátí seznam segmentů obsahujících časové značky a skóre spolehlivosti pro každý segment.

4.4 Tovární funkce a cache

Inicializace modelů probíhá pomocí tovární funkce `create_asr_engine()` v souboru `utils.py`. Tato funkce na základě vstupního parametru s názvem modelu a dalších konfiguračních parametrů vytvoří odpovídající instanci třídy. Zároveň implementuje mechanismus cache pomocí globálního slovníku `_model_cache` uloženého v paměti procesu.



Obrázek 4.2: Cache diagram

Jak znázorňuje diagram výše, při prvním požadavku na přepis Flask předá požadavek tovární funkci. Továrna zjistí, že model není v cache, a proto vytvoří instanci konkrétní třídy (Whisper, Canary nebo Parakeet), zavolá metodu `download()`, která načte váhy modelu, a instanci uloží do cache. Teprve poté je spuštěn samotný přepis. Při každém dalším požadavku se stejnými parametry továrna nalezne existující instanci přímo v cache a předá ji bez jakéhokoli opětovného načítání. Klíč cache je tvořen kombinací názvu modelu, zařízení (CPU/GPU) a všech konfiguračních parametrů, takže změna libovolného parametru změní záznam v cache. Tento mechanismus je zásadní z hlediska výkonu, protože načtení ASR modelu může na CPU trvat desítky sekund až minuty. Díky cache dochází k načtení modelu pouze jednou za dobu běhu procesu, což výrazně zrychluje odezvu aplikace při opakovaných přepisech.

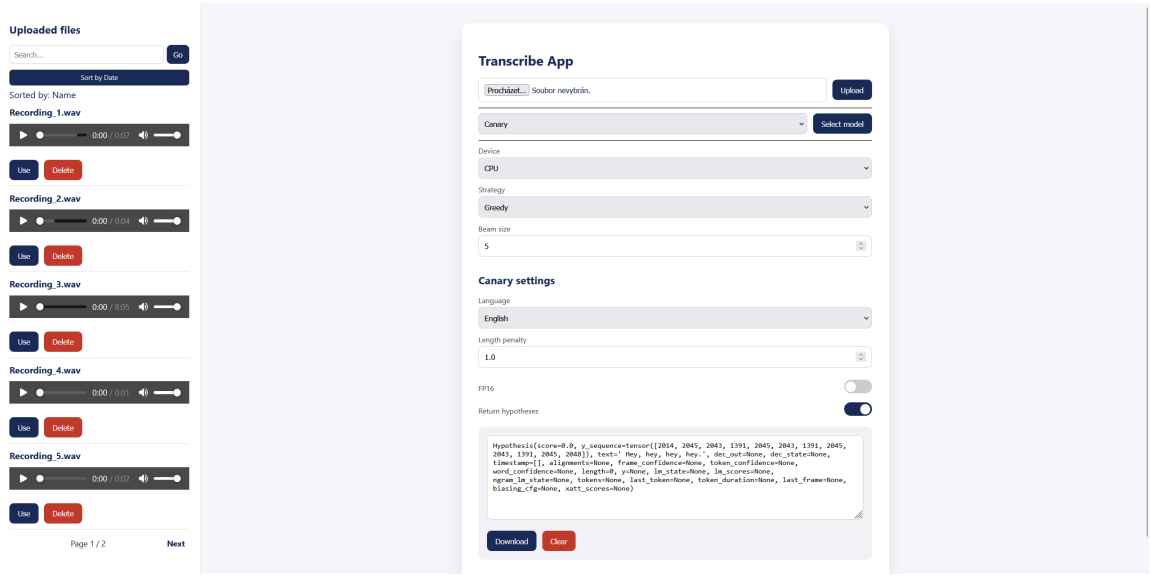
4.5 Uživatelské rozhraní

Uživatelské rozhraní je implementováno jako jednostránková aplikace s renderováním na straně serveru. Layout využívá CSS flexbox se dvěma sloupci – postranní panel vlevo obsahuje správu souborů a hlavní panel vpravo obsahuje formulář s parametry a výsledky přepisu.

Booleovské parametry modelů jsou zobrazeny jako přepínače implementované v čistém CSS bez JavaScriptu. Parametry specifické pro každý model jsou zobrazeny podmíněně pomocí Jinja2 bloků, takže uživatel vidí vždy pouze relevantní nastavení pro zvolený model.

Při spuštění přepisu je zobrazen překryvný panel s animovaným indikátorem načítání, který je aktivován JavaScriptem při odeslání formuláře. Tento panel informuje uživatele o probíhajícím přepisu, který může na CPU trvat několik minut. Panel zmizí automaticky po načtení stránky s výsledkem.

Pokud je u modelu Whisper aktivována volba zobrazení časových značek nebo míry spolehlivosti, je pod textovým přepisem zobrazena tabulka se segmenty. Každý řádek tabulky obsahuje volitelně počáteční a koncový čas segmentu a míru spolehlivosti. Sloupce tabulky jsou dynamicky zobrazeny nebo skryty podle toho, které volby uživatel aktivoval.



Obrázek 4.3: Ukázka aplikace

4.6 Nasazení

Celý systém je zabalen do Docker kontejneru postaveného na obrazu `python:3.12-slim`. Dockerfile instaluje systémové závislosti včetně `ffmpeg` pro zpracování audio souborů a `gosu` pro bezpečné snížení oprávnění procesu. Entrypoint skript vytvoří adresář `/app/uploads`, nastaví jeho vlastnictví a spustí aplikaci jako neprivilegovaný uživatel s UID 1000.

Sestavení a publikace Docker obrazu jsou automatizovány pomocí GitHub Actions pipeline, která se spouští při každém commitu do hlavní větve. Pipeline sestaví obraz a nahraje jej do Docker Hub, odkud jej Umbrel stáhne při aktualizaci aplikace.

Celý proces zpracování požadavku je plně synchronní: uživatel nahraje soubor, zvolí model a parametry, backend vytvoří nebo načte model z cache, provede transkripci a výsledek vrátí zpět do šablony. Tento přístup zjednodušuje implementaci a zajišťuje deterministické chování aplikace, přičemž vizuální zpětná vazba v podobě načítacího překryvu kompenzuje absenci asynchronního zpracování.

Celý systém je navržen jako modulární a rozšiřitelný. Přidání nového ASR modelu znamená vytvoření nové třídy implementující rozhraní `BaseASR` a její registraci v tovární funkci `create_asr_engine()`. Díky tomu není potřeba zasahovat do ostatních částí systému, ať už jde o uživatelské rozhraní nebo aplikační logiku.

4.7 Struktura souborů a adresářů

Soubor / adresář	**Účel**
app/main.py	Vstupní bod Flask aplikace. Obsahuje všechny routy, zpracování požadavků, extrakci parametrů a renderování šablon.
app/utils.py	Továrna pro ASR enginy s in-memory cache modelů. Centrální místo pro vytváření instancí modelů.
app/baseASR.py	Abstraktní základní třída definující rozhraní, které musí implementovat všechny ASR enginy.
app/whisper.py	Wrapper pro Whisper ASR engine využívající knihovnu faster-whisper.
app/canary.py	Wrapper pro Canary ASR engine využívající NVIDIA NeMo EncDecMultiTaskModel.
app/parakeet.py	Wrapper pro Parakeet ASR engine využívající NVIDIA NeMo EncDecRNNTBPEModel.
app/templates/index.html	Jediná Jinja2 HTML šablona pro celé UI. Obsahuje sidebar, hlavní panel, formuláře a výsledky.
app/static/style.css	Veškeré CSS aplikace – layout, sidebar, formuláře, tabulky, spinner, přepínače.
app/requirements.txt	Python závislosti pro instalaci pomocí pip.
Dockerfile	Definice Docker image. Vytváří Python 3.12 slim image s ffmpeg, gosu a všemi pip závislostmi.
entrypoint.sh	Docker entrypoint skript. Vytváří adresář /app/uploads a snižuje oprávnění na uživatele s ID 1000.
.github/workflows/docker-image.yml	GitHub Actions CI/CD pipeline. Sestavuje a odesílá Docker image na Docker Hub při pushi do větve main.

Tabulka 4.1: Struktura projektu a popis jednotlivých souborů

4.8 Použité technologie

V rámci implementace webové aplikace pro automatický přepis řeči bylo využito několik moderních technologií a knihoven, které společně tvoří základ celého systému. Jejich volba byla motivována zejména požadavky na modularitu, jednoduchost nasazení, možnost lokálního provozu bez závislosti na cloudových službách a zároveň podporu pro integraci moderních modelů strojového učení.

Celá architektura aplikace je postavena jako klient-server řešení s renderováním na straně serveru, přičemž jednotlivé technologie pokrývají prezentační vrstvu, aplikační logiku i výpočetně náročné operace spojené se zpracováním audia.

4.8.1 Programovací jazyk Python

Základním stavebním kamenem serverové části je programovací jazyk Python, který byl zvolen především díky své vysoké čitelnosti, jednoduché syntaxi a rozsáhlému ekosystému knihoven zaměřených na umělou inteligenci a zpracování signálů.

Python je v současné době standardem pro vývoj aplikací využívajících neuronové sítě, což umožňuje snadnou integraci předtrénovaných modelů pro automatické rozpoznávání řeči. Významnou výhodou je také jeho multiplatformnost. Díky Pythonu je možné efektivně propojit jednotlivé komponenty systému, od webové logiky až po výpočetně náročné modely běžící na GPU.

4.8.2 Webový framework Flask

Pro implementaci webové části aplikace byl zvolen framework Flask, který představuje minimalistické, ale velmi flexibilní řešení pro tvorbu webových aplikací v Pythonu.

Volba Flasku byla motivována jeho jednoduchostí a absencí striktně dané architektury, což umožňuje přizpůsobit strukturu aplikace konkrétním požadavkům. Aplikace využívá jedinou hlavní routu `/`, která na základě parametru `action` odeslaného z HTML formuláře rozlišuje různé operace – nahrání souboru, výběr modelu, spuštění transkripce, mazání souborů nebo stažení přepisu. Tento přístup minimalizuje počet endpoint funkcí a zjednodušuje správu stavu aplikace.

Flask také poskytuje jednoduché a přehledné routy pro obsluhu statických souborů (`/uploads/<filename>`) a stahování výsledků (`/download`), kde je výsledný TXT soubor generován přímo v paměti pomocí `BytesIO` bez nutnosti zapisovat dočasné soubory na disk.

Součástí frameworku je šablonovací systém Jinja2, který umožňuje generování dynamického HTML obsahu. Šablony využívají podmíněné bloky pro zobrazení pouze těch parametrů, které jsou relevantní pro aktuálně zvolený model, což zjednodušuje uživatelské rozhraní a zabraňuje zobrazování nepodporovaných voleb.

4.8.3 Prezentační vrstva

Prezentační vrstva aplikace je implementována pomocí standardních webových technologií HTML a CSS bez využití JavaScriptových frameworků. HTML zajišťuje strukturu uživatelského rozhraní, zatímco CSS definuje jeho vizuální podobu včetně layoutu, přepínačů, tabulek a animovaného indikátoru načítání.

Layout využívá CSS flexbox se dvěma sloupci. Postranní panel vlevo obsahuje správu nahraných souborů – vyhledávání, řazení podle názvu nebo data nahrání, stránkování a přehrávání souborů pomocí HTML5 audio prvku. Hlavní panel vpravo obsahuje formulář s výběrem modelu, nastavením parametrů a zobrazením výsledků přepisu.

4.8.4 PyTorch

Základním frameworkem pro práci s neuronovými sítěmi je PyTorch, který slouží jako výpočetní základ pro všechny tři integrované modely. PyTorch poskytuje nízkoúrovňové operace pro výpočty na tenzorech a umožňuje efektivní využití GPU akcelerace prostřednictvím CUDA. V aplikaci je PyTorch využíván také pro explicitní přesun modelu na zvolené zařízení pomocí `model.to(device)` a volitelně pro převod na poloviční přesnost `model.half()` při práci na GPU, což snižuje paměťové nároky.

4.8.5 Podpůrné knihovny

Pro bezpečnou manipulaci s nahrávanými soubory je využívána funkce `secure_filename()` z knihovny Werkzeug, která je součástí ekosystému Flask. Tato funkce čistí název souboru a zabraňuje útokům typu path traversal, při kterých by útočník mohl prostřednictvím speciálně sestaveného názvu souboru přistupovat k souborům mimo určený adresář. Validace nahrávaných souborů je dále rozšířena o kontrolu přípony a skutečného obsahu WAV souboru prostřednictvím standardní knihovny `wave`.

Pro zpracování audio souborů je v Docker obrazu nainstalován nástroj `ffmpeg`, který je interně využíván knihovnami faster-whisper a NeMo pro dekódování a převzorkování audio souborů před samotnou inferencí.

4.8.6 Nasazení a Docker

Nasazení celé aplikace je realizováno pomocí technologie Docker. Sestavení a publikace Docker obrazu jsou automatizovány pomocí GitHub Actions pipeline definované v souboru `.github/workflows/docker-image.yml`. Pipeline se spouští při každém commitu do hlavní větve, sestaví obraz a nahraje jej do Docker Hub. Na platformě Umbrel je aplikace provozována jako samostatná služba, která stáhne aktuální obraz z Docker Hub a spustí kontejner s namapovaným adresářem `/app/uploads` jako Docker volume, čímž zajistí persistenci nahraných souborů napříč restarty kontejneru.

4.9 Modely

ASR modely jsou schopné analyzovat akustický signál, identifikovat jednotlivá slova a syntakticky je spojit do koherentního textového výstupu. Nacházejí uplatnění v široké škále aplikací, např. v hlasových asistentech, automatickém titulkování videí, přepisování schůzek nebo záznamů přednášek, stejně jako v překladu řeči v reálném čase.

Vývoj ASR modelů se v posledních letech výrazně posunul směrem k multijazyčnosti, robustnosti a efektivitě inferencí. Mnoho moderních systémů podporuje více jazyků a dokáže automaticky detekovat jazyk vstupu, čímž odstraňuje potřebu předchozího nastavení. Další důležitou vlastností je možnost tzv. fine-tuning modelů na specifickou doménu nebo firemní terminologii, což umožňuje zvýšit přesnost přepisu odborných textů, technických termínů či jmen vlastníků dat. Současné ASR modely často kombinují různé metody zpracování signálu, jazykového modelování a optimalizace rychlosti zpracování, aby bylo možné poskytovat spolehlivé a rychlé přepisy, které jsou využitelné jak v reálném čase, tak i při zpracování archivních audio dat.

V oblasti automatického rozpoznávání řeči se v posledních letech výrazně prosadily platformy a frameworky, které umožňují rychlý vývoj a nasazení ASR modelů. Mezi nejznámější patří NVIDIA NeMo a OpenAI Whisper. NeMo je framework zaměřený na stavbu a trénink multimodálních modelů pro rozpoznávání řeči, překlad a syntézu řeči. Díky otevřenému přístupu, rozsáhlým předtrénovaným modelům a snadné integraci s GPU se stal velmi populárním ve výzkumné komunitě i v průmyslových aplikacích. Podobně modely Whisper od OpenAI získaly široké uznání pro svou schopnost robustně rozpoznávat řeč v mnoha jazycích a náročných akustických podmínkách. Popularita těchto řešení spočívá zejména v kombinaci vysoké přesnosti, dostupnosti otevřeného kódu a možnosti rychlého nasazení pro různé praktické scénáře, od akademického výzkumu až po komerční aplikace, jako jsou hlasoví asistenti, přepisovací služby či automatické titulkování videí.

V rámci systému jsou implementovány tři různé modely pro rozpoznávání řeči.

4.9.1 Whisper

Prvním z nich je model Whisper, který je založen na architektuře transformerů a je známý svou vysokou přesností a robustností v různých jazykových podmínkách. V této aplikaci je využita optimalizovaná verze faster-whisper, která je efektivnější z hlediska výpočetního výkonu a umožňuje rychlejší inferenci. Whisper podporuje pokročilé funkce, jako je segmentace řeči, časové značky jednotlivých úseků a odhad důvěryhodnosti predikce.

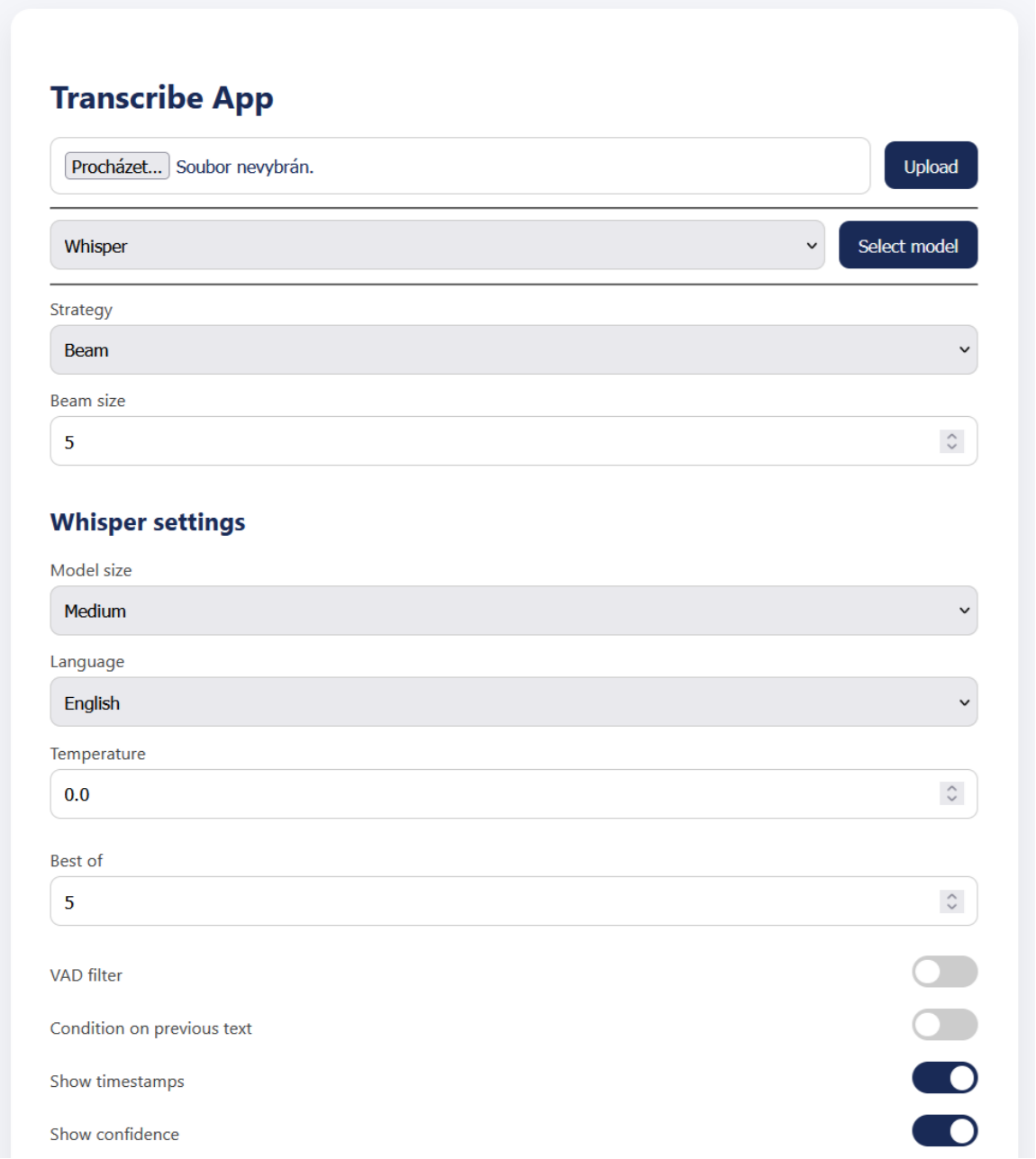
Faster-whisper je reimplementace modelu Whisper od OpenAI využívající CTranslate2 – vysoce optimalizovaný inference engine. Je výrazně rychlejší než původní Whisper na CPU a používá méně paměti, což z něj dělá nejpraktičtější volbu pro nasazení pouze na CPU. Podporované jazyky zahrnují prakticky všechny hlavní světové jazyky, včetně češtiny, angličtiny, němčiny, francouzštiny, španělštiny a mnoha dalších.[9]

Parametry Whisperu

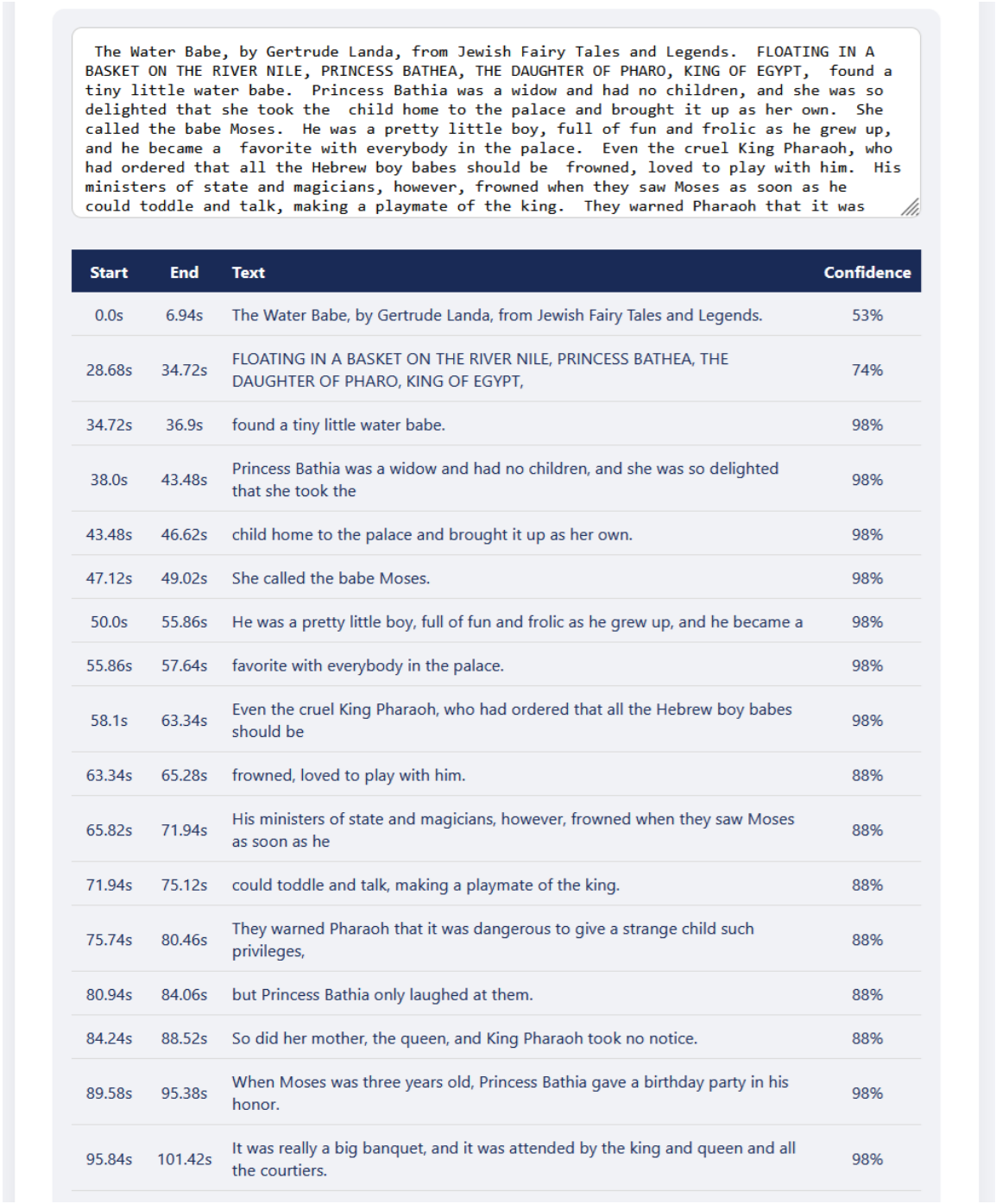
Parametr	**Popis**
model_size	Určuje velikost vah modelu: tiny, base, small, medium, large-v3. Větší modely jsou přesnější, ale pomalejší a náročnější na RAM.
language	Vynutí přepis v konkrétním jazyce. Pokud je nastaven správně, zlepšuje přesnost i rychlost tím, že přeskočí detekci jazyka.
beam_size	Počet paprsků (beams) při beam search dekódování. Vyšší hodnoty zkoumají více kandidátů a zvyšují přesnost za cenu nižší rychlosti.
temperature	Teplota vzorkování. Při hodnotě 0.0 (výchozí) se používá deterministický beam search. Vyšší hodnoty přidávají náhodnost.
best_of	Počet vzorkovaných kandidátů, pokud temperature > 0. Má význam pouze při nenulové teplotě.
vad_filter	Filtr detekce hlasové aktivity (Voice Activity Detection). Přeskakuje tiché části před přepisem, čímž zrychluje zpracování a omezuje „halucinace“ u tichého audia.
condition_on_previous_text	Určuje, zda je každý 30sekundový úsek podmíněn přepisem předchozího úseku. Pomáhá konzistenci u dlouhých nahrávek, ale může způsobit smyčky halucinací.
show_timestamps	Zapíná výstup časových značek na úrovni segmentů (začátek/konec v sekundách), zobrazovaných v tabulce pod přepisem.
show_confidence	Zapíná zobrazení míry spolehlivosti pro každý segment. Užitečné pro identifikaci nejistých částí.

Tabulka 4.2: Konfigurační parametry modelu Whisper

Pokud je zapnuta volba `show_timestamps` nebo `show_confidence`, je metoda `transcribe()` volána s parametry `return_segments=True` a `word_timestamps=True`. Tím je zajištěno, že model vrací detailnější strukturovaný výstup zahrnující jednotlivé segmenty přepisu. Surová data získaná z knihovny faster-whisper jsou následně převedena do jednotné struktury slovníků, které obsahují informace o začátku a konci jednotlivých segmentů, samotný přepsaný text, hodnotu confidence (míru spolehlivosti) a také seznam jednotlivých slov s jejich časováním a pravděpodobností výskytu. Tato strukturovaná data jsou následně předána do šablonovací vrstvy aplikace, kde jsou vizualizována jako HTML tabulka pro přehledné zobrazení uživateli.



Obrázek 4.4: Ukázka přepisu pomocí modelu Whisper s časovými značkami (1. část)



Obrázek 4.5: Ukázka přepisu pomocí modelu Whisper s časovými značkami (2. část)

4.9.2 Canary

Druhým použitým modelem je Canary (canary-180m-flash), který je součástí knihovny NVIDIA NeMo a představuje multitask model schopný nejen rozpoznávání řeči, ale v některých konfiguracích také provádět překlad mezi jazyky. Tento model využívá architekturu EncDecMultiTaskModel a je navržen pro práci s více jazyky současně. V rámci aplikace je Canary využíván především pro standardní přepis řeči, přičemž umožňuje konfiguraci pa-

rametrů, jako je jazyk vstupu, délková penalizace nebo režim generování hypotéz, které ovlivňují výslednou podobu a kvalitu přepisu.[5]

Model Canary-180m-flash podporuje pouze čtyři následující jazyky:

- angličtina (en)
- němčina (de)
- španělština (es)
- francouzština (fr)

Češtinu tento model nepodporuje.

Parametry Canary

Parametr	**Popis**
language	Zdrojový a cílový jazyk pro přepis.
strategy	Strategie dekódování: greedy (nejrychlejší, nižší kvalita) nebo beam (pomalejší, vyšší kvalita).
beam_size	Počet paprsků (beams) při beam search dekódování.
device	Spuštění na CPU nebo GPU (CUDA).
use_fp16	Poloviční přesnost. Funguje pouze na GPU.
return_hypotheses	Vrací celý objekt hypotézy z NeMo místo pouhého textu.
len_pen	Penalizace délky při beam search. Hodnoty nad 1.0 upřednostňují delší výstupy, pod 1.0 kratší. Pro běžné ASR ji obvykle není potřeba měnit.

Tabulka 4.3: Konfigurační parametry modelu Canary

4.9.3 Parakeet

Třetím použitým modelem je Parakeet (parakeet-tdt-0.6b-v3), který je založen na architektuře Recurrent Neural Network Transducer (RNNT) a využívá tokenizaci pomocí Byte Pair Encoding (BPE). Tento model je optimalizován především pro rychlou inferenci a nízkou latenci, což jej činí vhodným zejména pro scénáře, kde je kladen důraz na rychlost zpracování, např. při delších audio nahrávkách nebo interaktivním přepisu v reálném čase.

Z hlediska implementace je Parakeet integrován prostřednictvím knihovny NVIDIA NeMo, konkrétně pomocí třídy EncDecRNNTBPEModel. To, že se jedná o model typu RNNT, představuje odlišný přístup oproti modelům založeným na attention mechanismech, jako je např. Canary. RNNT architektura obecně dosahuje nižší latence při inferenci, protože umožňuje průběžné zpracování vstupního signálu bez nutnosti čekat na kompletní sekvenci.

Model Parakeet podporuje různé dekódovací strategie, zejména greedy dekódování a beam search, přičemž dostupnost a efekt těchto strategií může být v některých implementacích omezen v závislosti na konkrétní konfiguraci modelu. Díky své optimalizaci je Parakeet vhodný především pro situace, kde je prioritou rychlost zpracování nad maximální přesností, zatímco stále poskytuje kvalitní výsledky přepisu.[6]

Parametry Parakeet

Parametr	Popis
strategy	Strategie dekódování: greedy (nejrychlejší, nižší kvalita) nebo beam (pomalejší, vyšší kvalita).
beam_size	Počet paprsků (beams) při beam search dekódování.
device	Spuštění na CPU nebo GPU (CUDA).
use_fp16	Poloviční přesnost. Funguje pouze na GPU.
return_hypotheses	Vrací celý objekt hypotézy z NeMo místo pouhého textu.

Tabulka 4.4: Konfigurační parametry modelu Parakeet

4.10 Nasazení aplikace na platformě Umbrel

Platforma Umbrel je určena primárně pro aplikace hostované na vlastním serveru, přičemž klíčovou součástí je automatizace instalace a správy služeb pomocí Docker kontejnerů. Umbrel poskytuje uživateli přístup k aplikacím a rovněž možnost instalovat komunitní či oficiální aplikace prostřednictvím obchodu Umbrel App Store.

Umbrel nabízí dva způsoby, jak zpřístupnit aplikace uživatelům: prostřednictvím oficiálního Umbrel App Store a prostřednictvím komunitního obchodu dostupného pomocí odkazu na vlastní GitHub repozitář. Oba přístupy mají odlišné nároky na kvalitu, strukturu a testování aplikací.

Pro zařazení aplikace do oficiálního Umbrel App Store musí být splněny přesně definované podmínky. Každá aplikace musí být zabalená jako Docker kontejner a poskytovat webové rozhraní, které bude kompatibilní s Umbrel UI. Součástí balení je manifestový soubor `umbrel-app.yml`, obsahující povinná metadata, jako jsou identifikátor aplikace, název, popis, kategorie a verze, a případně odkazy na podporu. Dále je vyžadován soubor `docker-compose.yml`, který správně definuje služby aplikace, porty a proměnné prostředí. Docker image by měl podporovat více architektur, např. ARM64 a x86, aby bylo možné aplikaci spustit na různých typech zařízení. Součástí požadavků jsou také ikonka aplikace ve formátu SVG a galerie screenshotů, které zajišťují vizuální prezentaci aplikace uživatelům.

Před samotným odesláním do obchodu je aplikace testována lokálně v prostředí Umbrel development, a pokud je to možné, také na fyzickém zařízení s Umbrel OS. Testuje se funkčnost webového rozhraní, správná persistence dat a kompatibilita s ostatními službami. Poté autor vytvoří kopii repozitáře `getumbrel/umbrel-apps`, vloží svou aplikaci do nové větve a pošle ji ke kontrole komunitě. Během kontroly se ověřuje správnost nastavení aplikace, její bezpečný a spolehlivý běh a případně se provádějí úpravy. Po schválení kontroly je aplikace oficiálně dostupná všem uživatelům Umbrel prostřednictvím oficiálního obchodu s aplikacemi.

Naopak, pokud je aplikace zpřístupněna pouze prostřednictvím komunitního obchodu nebo odkazu na vlastní GitHub, požadavky jsou výrazně volnější. Struktura souborů, testování a kompatibilita nejsou oficiálně kontrolovány, a odpovědnost za funkčnost a bezpečnost aplikace nese její autor. Uživatelé si aplikaci instalují manuálně, což znamená, že musí ručně stáhnout Docker image, případně upravit konfiguraci a ověřit funkčnost na vlastním zařízení. Tento přístup je vhodný pro experimentální, neveřejné nebo beta verze aplikací, které zatím neprošly formálním schvalovacím procesem.[4]

4.10.1 Komunitní obchod

V rámci řešení této práce jsem se rozhodl publikovat vlastní aplikaci prostřednictvím komunitního obchodu, tedy distribuovat ji samostatně, bez zařazení do oficiálního Umbrel App Store. Komunitní obchod představuje způsob, jak zpřístupnit aplikace uživatelům platformy Umbrel bez nutnosti projít přísným schvalovacím procesem, který vyžaduje oficiální Umbrel repozitář. Tento přístup je vhodný zejména v raných fázích vývoje, pro testování nebo pro sdílení neveřejných či experimentálních aplikací. Pro potřeby této práce byly využity dva veřejné repozitáře na GitHubu.

Repozitář se strukturou komunitního obchodu

První repozitář slouží jako šablona komunitního obchodu s aplikacemi. Neobsahuje samotnou funkční aplikaci, ale poskytuje infrastrukturu a vzorové soubory, které představují minimální strukturu potřebnou pro založení vlastního komunitního obchodu s Umbrel aplikacemi mimo oficiální Umbrel App Store. Když uživatel přidá URL takového repozitáře do své Umbrel instance, Umbrel ho stáhne, zpracuje konfigurační soubory YAML a zobrazí aplikace v uživatelském rozhraní – stejně jako aplikace z oficiálního obchodu. Uživatel je pak může nainstalovat jedním kliknutím.

Hlavním prvkem tohoto repozitáře je konfigurační soubor, který definuje identifikátor a název komunitního obchodu. Tento soubor je důležitý, protože určuje, jakým způsobem budou aplikace uvnitř tohoto obchodu identifikovány v systému Umbrel. Dále repozitář obsahuje základní dokumentaci, která popisuje proces vytvoření takového obchodního kanálu, včetně pokynů, jak aplikace organizovat a sdílet. Tato struktura vychází z oficiálního Open Source template repozitáře určeného právě pro komunitní obchody, což usnadňuje autorům založit vlastní distribuční kanál bez přímého zapojení oficiálního vývojového týmu Umbrel.

Soubor `umbrel-app-store.yml` je konfigurační soubor v kořenové složce celého app storu. Obsahuje id a název obchodu. Pole `id` je nejdůležitější část celé struktury projektu. Funguje jako povinný prefix pro každou aplikaci v obchodě. Tím se předchází kolizím názvů mezi různými community obchody i oficiálním obchodem. Je to jednoduchý, ale efektivní způsob, jak zabránit kolizím.

Pole name slouží pouze k zobrazení. Je to lidsky čitelný název, který se zobrazí v rozhraní Umbrel jako „<name> App Store“.

Každá aplikace v obchodě má svou vlastní samostatnou složku. Název složky musí přesně odpovídat id aplikace. Uvnitř každé složky aplikace jsou přesně dva povinné soubory: `umbrel-app.yml` a `docker-compose.yml`. Tyto dva soubory dohromady říkají Umbrelu všechno, co potřebuje vědět – jak aplikace vypadá v obchodě a jak ji ve skutečnosti spustit.

Soubor `umbrel-app.yml` definuje stránku aplikace v obchodě. Všechno, co uživatel vidí v rozhraní Umbrelu před instalací aplikace, pochází právě odsud.

Soubor `umbrel-app.yml` obsahuje:

- Unikátní ID aplikace. Musí přesně odpovídat názvu složky a musí obsahovat prefix ID obchodu.
- Zobrazovaný název aplikace a krátký jednořádkový popis, který se ukazuje v přehledu aplikací.
- URL adresu ikonky aplikace. Umbrel si obrázky neukládá – pouze je načítá z externích zdrojů, což udržuje repozitář na GitHub malý a rychlý.

- Kategorii aplikace v obchodě. Umbrel má předdefinované kategorie, podle kterých se aplikace třídí.
- Verzi aplikace.
- Port, na kterém poběží webové rozhraní aplikace.
- Delší popis aplikace, který se zobrazuje na detailní stránce.
- Informace o autorech a zdrojích.
- Seznam URL snímků aplikace. Opět jsou hostované externě, takže repozitář zůstává bez binárních souborů.
- Seznam dalších Umbrel aplikací, které musí být nainstalované, aby tato aplikace fungovala.

V souboru `docker-compose.yml` se definuje, jak se aplikace skutečně spustí. Umbrel používá Docker pro běh aplikací v izolovaných kontejnerech a Docker Compose pro jejich orchestraci. Mapování složek je klíčové pro uchování dat. Kontejnery jsou dočasné – při aktualizaci by se data ztratila.

Tento návrh odráží několik záměrných principů:

Princip	**Popis**
Oddělení odpovědností	Soubor s metadaty (`umbrel-app.yml`) je zcela oddělený od definice nasazení (`docker-compose.yml`). Uživatelské rozhraní obchodu a samotné spuštění aplikace jsou definovány nezávisle na sobě.
Docker jako univerzální runtime	Díky standardizaci na Dockeru Umbrel nemusí řešit implementační jazyk aplikace. Aplikace může být napsána v Pythonu, Node.js, Go nebo PHP. Pokud běží v Docker kontejneru, je kompatibilní s Umbrel platformou.
Namespacovaná ID	Systém prefixů podle ID obchodu představuje jednoduchý a efektivní způsob, jak zabránit kolizím názvů aplikací v decentralizovaném prostředí, kde může existovat více nezávislých repozitářů.
Externí hosting assetů	Ukládání obrázků na externí služby místo přímo v repozitáři udržuje projekt malý, rychlý na stažení a zaměřený především na konfigurační soubory.
Pattern `app_proxy`	Centralizace autentizace a směrování provozu přes sdílený proxy kontejner eliminuje potřebu implementace vlastního přihlašování v jednotlivých aplikacích. Bezpečnostní vrstva Umbrelu tak automaticky „obaluje“ každou aplikaci.

Tabulka 4.5: Základní principy návrhu komunitního obchodu

Repozitář obsahující aplikaci

Druhý repozitář obsahuje konkrétní implementaci aplikace, která může být součástí komunitního obchodu. Na rozdíl od prvního repozitáře se zde nachází `Dockerfile` a složka s kódem aplikace, což znamená, že tento repozitář slouží k vytvoření Docker image, který Umbrel dokáže spustit jako samostatnou službu.

`Dockerfile` je konfigurační soubor, který definuje prostředí, ve kterém aplikace poběží, včetně použití základního systému, instalace potřebných závislostí a parametrů spuštění. Složka s aplikací obsahuje zdrojový kód a automatizované procesy, jako je např. automatická tvorba Docker image při každém změně v repozitáři. Tento repozitář tedy představuje funkční jádro vlastní Umbrel aplikace, kterou lze díky komunitnímu obchodu distribuovat uživatelům.

Komunitní obchod tedy představuje pružnou a méně formální alternativu k oficiálnímu App Store, která dává autorům možnost prezentovat své aplikace bez potřeby formální recenze a dodatečných schvalovacích kroků. V kontextu této práce umožnil použitý přístup rychle ověřit funkčnost vyvíjené aplikace a zpřístupnit ji uživatelům platformy Umbrel bez složitého schvalovacího procesu.

Kapitola 5

Přehled funkcí aplikace

Navržená aplikace pro automatický přepis řeči poskytuje uživateli komplexní sadu funkcí zaměřených na efektivní práci se zvukovými daty. Důraz je kladen především na jednoduchost použití, flexibilitu konfigurace a možnost využití různých modelů pro rozpoznávání řeči. Aplikace nepředstavuje pouze nástroj pro převod řeči na text, ale univerzální prostředí umožňující experimentování s různými přístupy k automatickému přepisu.

5.1 Správa audio souborů

Základní funkcionalitou aplikace je práce se zvukovými soubory ve formátu WAV. Uživatel má možnost nahrát vlastní audio soubor prostřednictvím webového rozhraní pomocí HTML formuláře, přičemž soubory jsou po úspěšné validaci ukládány na server do vyhrazeného adresáře.

Součástí správy souborů je integrovaný audio přehrávač postavený na HTML5 audio prvku, který umožňuje přehrání nahrávky přímo v aplikaci bez nutnosti stahování souboru. Tato funkcionalita slouží k rychlé kontrole obsahu souboru před jeho zpracováním. Uživatel může rovněž soubory mazat, čímž je zajištěna možnost dlouhodobé správy uložených dat.

Pro zvýšení přehlednosti aplikace podporuje filtrování souborů podle názvu a jejich řazení podle data poslední modifikace nebo abecedně podle názvu. Při větším množství souborů je seznam rozdělen do stránek po pěti souborech, což zajišťuje komfortní práci i při rozsáhlejším využití aplikace. Soubory jsou přehledně zobrazeny v postranním panelu, přičemž každý soubor je doplněn o audio přehrávač a tlačítka pro jeho použití k přepisu nebo smazání.

5.2 Volba ASR modelu

Klíčovou funkcí aplikace je možnost výběru konkrétního modelu pro automatické rozpoznávání řeči. Aplikace podporuje tři modely (Canary, Parakeet a Whisper), z nichž každý využívá odlišnou architekturu neuronové sítě a liší se jak přesností, tak výpočetní náročností či podporou jazyků.

Tato variabilita umožňuje uživateli přizpůsobit chování aplikace konkrétním požadavkům. Whisper je vhodný pro vícejazyčné přepisy, zatímco Parakeet poskytuje rychlé výsledky v angličtině a Canary nabízí kompromis mezi rychlostí a kvalitou pro podporované evropské jazyky. Možnost volby modelu tak výrazně zvyšuje flexibilitu celé aplikace.

Po výběru modelu aplikace dynamicky upraví zobrazené parametry tak, aby uživatel viděl pouze nastavení relevantní pro zvolený model. Parametry nepodporované daným modelem jsou automaticky skryty.

5.3 Konfigurace parametrů přepisu

Aplikace umožňuje detailní nastavení parametrů jednotlivých modelů, čímž poskytuje uživateli kontrolu nad procesem přepisu. Parametry jsou rozděleny na společné a modelově specifické.

Společné parametry dostupné u modelů Canary a Parakeet zahrnují volbu výpočetního zařízení (CPU nebo GPU), způsob dekódování (greedy search nebo beam search) a počet paprsků při beam search dekódování. Greedy search je rychlejší, ale méně přesný, zatímco beam search prohledává více hypotéz současně a dosahuje zpravidla lepších výsledků za cenu vyšší výpočetní náročnosti. U obou modelů NeMo je dále možné aktivovat výpočet v poloviční přesnosti FP16, které je relevantní při využití GPU a snižuje paměťové nároky modelu.

Model Canary nabízí navíc nastavení jazyka přepisu a penalizaci délky výstupu (`len_pen`), která ovlivňuje preferenci kratších nebo delších přepisů při beam search dekódování.

Model Whisper poskytuje nejrozsáhlejší sadu konfigurovatelných parametrů. Uživatel může zvolit velikost modelu, přičemž větší modely dosahují vyšší přesnosti za cenu pomalejšího přepisu a vyšších paměťových nároků. Dále lze nastavit jazyk vstupu, teplotu vzorkování, počet alternativních návrhů (`best_of`), filtraci ticha pomocí detekce hlasové aktivity (VAD) a podmínění na předchozí text při zpracování dlouhých nahrávek rozdělených do segmentů.

5.4 Zobrazení výsledků přepisu

Výsledný přepis je uživateli zobrazen v textové podobě v přehledném textovém poli. Tento formát umožňuje snadné kopírování nebo další zpracování textu.

U modelu Whisper je možné volitelně aktivovat rozšířený výstup ve formě tabulky segmentů. Uživatel může nezávisle zapnout zobrazení časových značek a zobrazení míry spolehlivosti, přičemž tabulka dynamicky zobrazuje pouze aktivované sloupce. Časové značky určují počáteční a koncový čas každého segmentu v sekundách, což umožňuje přesnou lokalizaci řeči v nahrávce. Míra spolehlivosti je vyjádřena v procentech a udává, s jakou jistotou model rozpoznal řeč v daném segmentu. Tato informace je užitečná zejména pro identifikaci pasáží, kde mohlo dojít k chybě přepisu. Whisper navíc interně sbírá i časové značky a pravděpodobnosti na úrovni jednotlivých slov, které jsou dostupné pro případné další zpracování.

5.5 Export a práce s výsledky

Aplikace umožňuje export výsledného přepisu do textového souboru ve formátu TXT. Soubor je generován přímo v paměti serveru pomocí `BytesIO` bez nutnosti zapisovat dočasné soubory na disk a je uživateli nabídnut ke stažení prostřednictvím HTTP hlavičky `Content-Disposition: attachment`. Tento formát byl zvolen z důvodu jednoduchosti a široké kompatibility s dalšími nástroji.

Kromě exportu nabízí aplikace také možnost vymazání aktuálního výsledku a opětovného spuštění přepisu s jinými parametry nebo modelem. Tento cyklus umožňuje uživateli snadno experimentovat a porovnávat různé přístupy ke zpracování stejné nahrávky.

5.6 Uživatelský komfort a odezva systému

Z hlediska uživatelského komfortu aplikace obsahuje vizuální indikaci probíhajícího zpracování. Při spuštění přepisu je zobrazen překryvný panel s animovaným indikátorem načítání, který informuje uživatele o tom, že operace stále probíhá. Indikátor je aktivován JavaScriptem při odeslání formuláře a automaticky zmizí po načtení stránky s výsledkem. Tato funkce je důležitá zejména u výpočetně náročnějších modelů nebo delších nahrávek, kde může přepis na CPU trvat několik minut.

Aplikace dále využívá mechanismus cache pro opakované použití již načteného modelu. Načtení ASR modelu může trvat desítky sekund až minuty, proto je inicializovaná instance naposledy použitého modelu uchovávána v paměti procesu. Při opakovaném přepisu se stejným modelem a parametry je instance vrácena přímo z cache bez opětovného načítání, což výrazně zkracuje dobu odezvy.

Celé uživatelské rozhraní je navrženo s důrazem na jednoduchost a přehlednost. Parametry jsou logicky seskupeny podle modelu a zobrazeny formou přehledných přepínačů a vstupních polí. Uživatel vidí vždy pouze parametry relevantní pro aktuálně zvolený model, čímž je minimalizována složitost rozhraní.

5.7 Shrnutí funkčních možností

Celkově aplikace pokrývá celý proces práce se zvukovými daty – od jejich nahrání a validace, přes správu, přehrávání a opakované zpracování, až po prezentaci výsledků s volitelnými časovými značkami a skóre spolehlivosti a jejich export. Díky podpoře tří modelů různých architektur, rozsáhlým konfigurovatelným parametrům, mechanismu cache a modulární architektuře představuje flexibilní nástroj, který lze dále rozšiřovat o nové modely a funkce bez zásahu do stávající aplikační logiky.

Kapitola 6

Testování

Testování představuje klíčovou fázi vývoje softwarové aplikace, jejímž cílem je ověřit správnou funkčnost, stabilitu a spolehlivost navrženého řešení. V případě aplikací využívajících modely strojového učení, jako je automatické rozpoznávání řeči, nabývá testování ještě většího významu. Kromě ověření samotné implementace je totiž nutné zajistit i korektní chování modelů při různých typech vstupních dat a konfigurací.

Navržená aplikace kombinuje více vrstev – od uživatelského rozhraní přes aplikační logiku až po integraci s externími knihovnami pro zpracování řeči. Z tohoto důvodu je nutné přistupovat k testování systematicky a pokrýt jak jednotlivé komponenty, tak jejich vzájemnou interakci. Důraz je kladen zejména na odhalení chyb v raných fázích vývoje, prevenci regresních problémů a ověření správného chování systému v reálných scénářích použití.

V této kapitole je popsán návrh a implementace testovací sady, použité přístupy k testování a vyhodnocení dosažených výsledků.

6.1 Testovací sada

Testovací sada aplikace představuje komplexní systém automatizovaných testů, jejichž cílem je ověřit správnou funkčnost celé aplikace napříč všemi vrstvami architektury. Testy jsou implementovány pomocí frameworku pytest, který umožňuje jak jednotkové (unit) testování jednotlivých komponent, tak i integrační testování celé aplikace včetně HTTP rozhraní postaveného nad Flaskem.

Hlavní motivací pro návrh této testovací sady bylo zajistit vysokou míru spolehlivosti systému, který pracuje s výpočetně náročnými modely automatického rozpoznávání řeči. Tyto modely mají odlišné chování, různé požadavky na hardware a mohou vykazovat specifické chyby v závislosti na délce vstupu, dekódovacích parametrech nebo typu audio signálu. Z tohoto důvodu byla testovací sada navržena tak, aby kombinovala rychlé validační testy s náročnějšími integračními testy využívajícími reálnou inferenci modelů.

6.1.1 Struktura testovací sady

Testy jsou rozděleny do tří hlavních kategorií. První kategorií jsou jednotkové testy, které neprovádějí načítání modelů ani výpočetně náročnou inferenci. Tyto testy slouží primárně k ověření správnosti logiky, struktury tříd, práce s parametry a jednotlivých komponent aplikace v izolaci. Jejich hlavní výhodou je velmi krátká doba běhu, což umožňuje jejich spouštění při každé změně kódu v rámci vývojového cyklu.

Druhou kategorií jsou integrační testy, které ověřují spolupráci více částí systému a zahrnují i interakci s modely automatického rozpoznávání řeči. Tyto testy již pracují s reálnou aplikační logikou a HTTP rozhraním a slouží k ověření správné funkčnosti klíčových částí systému včetně jejich vzájemné komunikace.

Třetí kategorií jsou end-to-end testy, které provádějí reálnou inferenci jednotlivých modelů nad syntetickými i reálnými audio daty. Tyto testy jsou zásadní pro ověření správné funkčnosti celé pipeline, tedy od načtení vstupního audio souboru, přes jeho zpracování modelem, až po generování finálního transkriptu. Nevýhodou těchto testů je jejich časová náročnost, která se pohybuje v řádu desítek minut v závislosti na výkonu CPU a velikosti modelů.

Spouštění testů je možné selektivně pomocí pytest markerů, což umožňuje oddělit rychlý vývojový cyklus od plné validační kontroly systému.

6.1.2 Sdílené fixture a generování testovacích dat

Základním stavebním prvkem testovací sady je soubor `conftest.py`, který obsahuje sdílené fixture využívané napříč všemi testy. Tyto fixture zajišťují konzistentní testovací prostředí a eliminují duplicitní kód.

Významnou roli zde hraje generování syntetických audio signálů, které je implementováno výhradně pomocí standardní knihovny Pythonu, konkrétně modulů `wave`, `struct` a `math`. Tento přístup byl zvolen záměrně, protože eliminuje závislost na externích knihovnách a tím zajišťuje plnou kompatibilitu s Docker prostředím.

Syntetická data jsou navržena tak, aby pokrývala různé typy vstupů. Krátké audio slouží pro základní testování funkčnosti, středně dlouhé audio simuluje situace přesahující jeden výpočetní blok modelu a dlouhé audio slouží jako zátěžový test více po sobě jdoucích segmentů. Speciální význam má také tiché audio, které slouží k ověření chování modelu v situacích bez řeči, což je důležitý edge-case z pohledu robustnosti systému.

Kromě syntetických dat testovací sada pracuje i s reálnými audio soubory, pokud jsou dostupné v adresáři uploadů. Tyto soubory umožňují ověřit chování systému v podmínkách blízkých produkčnímu nasazení.

6.1.3 Konfigurační řízení testů

Chování testovací sady je řízeno konfiguračním souborem `test_config.py`, který umožňuje dynamicky upravovat parametry testování bez zásahu do samotného testovacího kódu. Tento přístup podporuje flexibilitu a umožňuje např. vypnutí jednotlivých modelů, změnu velikosti Whisper modelu nebo omezení délky reálných testovacích souborů.

Důležitým aspektem této konfigurace je možnost přizpůsobení testovací sady dostupným výpočetním zdrojům. Např. změna velikosti Whisper modelu z varianty medium na tiny výrazně zrychluje testy, zatímco použití větších modelů umožňuje přesnější validaci kvality transkripce.

6.1.4 Testování továrny a cache mechanismu

Jednou z klíčových částí systému je tovární funkce `create_asr_engine()`, která zajišťuje vytváření instancí jednotlivých modelů. Tato funkce je doplněna o cache mechanismus, jehož cílem je minimalizovat počet opakovaných inicializací modelů, které jsou výpočetně i paměťově náročné.

Testy v této oblasti se zaměřují na několik důležitých aspektů. Především ověřují, že správný vstup vede k vytvoření odpovídající instance modelu, zatímco neplatné vstupy jsou správně odmítnuty formou výjimky. Dále je testováno, zda cache skutečně zajišťuje opakované použití již vytvořených instancí, což je ověřováno porovnáním identity objektů.

Zvláštní důraz je kladen na kontrolu, že inicializační metoda pro načítání modelu je volána pouze jednou pro danou kombinaci parametrů. Tím se ověřuje, že cache plní svůj účel a výrazně snižuje výpočetní náročnost systému.

6.1.5 Testování modelu Whisper

Model Whisper představuje jeden z hlavních komponent systému a jeho testování je proto velmi rozsáhlé. Testy pokrývají jak inicializační fázi, tak samotnou inferenci.

V rámci jednotkových testů je ověřována správná dědičnost z abstraktní třídy, konzistence výchozích parametrů a schopnost třídy správně uchovávat uživatelsky definované hodnoty. Důležitým aspektem je také ověření lazy loading přístupu, tedy skutečnosti, že model není načítán při inicializaci objektu, ale až při prvním použití.

Integrační část testů se zaměřuje na chování modelu při zpracování audio souborů různé délky. Zvláštní pozornost je věnována segmentaci audio signálu, protože Whisper pracuje s omezenou délkou vstupního kontextu a dlouhá audia jsou rozdělena do více částí. Testy ověřují správné spojení těchto segmentů, jejich časovou konzistenci a validitu výstupních metadat, jako jsou časové značky nebo odhady spolehlivosti.

6.1.6 Testování modelu Canary

Model Canary, založený na architektuře NVIDIA NeMo, má odlišné vlastnosti než Whisper, zejména v oblasti jazykové podpory a dekódovacích strategií. Testy jsou proto zaměřeny na validaci specifických parametrů tohoto modelu.

Zásadní část testů ověřuje omezenou jazykovou podporu, kde jsou povoleny pouze čtyři jazyky. Dále je testována správnost implementace dekódovacích strategií, jako je greedy a beam search, a jejich vliv na výstupní transkripci. Další důležitou oblastí je testování parametrů ovlivňujících délku a kvalitu výstupu, např. length penalty nebo velikost beam search.

6.1.7 Testování modelu Parakeet

Model Parakeet využívá architekturu RNNT, která se výrazně liší od předchozích modelů. Testy se proto zaměřují především na stabilitu inference a správnou implementaci dekódovacích strategií.

Zvláštní pozornost je věnována beam search režimu, který byl v předchozích verzích aplikace problematický. Testy ověřují, že tento režim nyní funguje správně a nevede k chybám při různých velikostech beam okna. Dále je testována stabilita modelu při různých typech vstupního audia včetně ticha a dlouhých nahrávek.

6.1.8 Integrační testy HTTP rozhraní

Integrační testy představují nejvyšší úroveň testování v systému. Jejich cílem je ověřit správnou funkčnost celé aplikace z pohledu uživatele, tedy prostřednictvím HTTP požadavků.

Testy pokrývají všechny klíčové operace, jako je nahrávání souborů, výběr modelu, spouštění transkripce, mazání souborů nebo stahování výsledků. Většina těchto testů vyu-

žívá mockování modelů, aby se eliminovala výpočetní náročnost. Některé testy však spouštějí reálnou inferenci, čímž ověřují kompletní end-to-end chování systému.

Důležitým aspektem těchto testů je ověření správného chování HTTP vrstvy, včetně redirectů, práce s query parametry a správného renderování HTML šablon.

6.2 Výsledky testování

Na základě provedeného testování lze konstatovat, že všechny implementované testy byly úspěšně dokončeny a neodhalily žádné kritické chyby, které by bránily správné funkčnosti aplikace. Jednotkové testy potvrdily korektní chování jednotlivých komponent, správnou práci s parametry a očekávané chování izolovaných částí systému. Integrační testy následně ověřily správné propojení mezi jednotlivými vrstvami aplikace, zejména mezi HTTP rozhraním, aplikační logikou a modelovou vrstvou.

End-to-end testy potvrdily správnou funkčnost celé aplikační pipeline, tedy od nahrání vstupního audio souboru přes jeho zpracování zvoleným modelem až po generování a stažení výsledné transkripce. Testování rovněž prokázalo správnou implementaci cache mechanismu modelu a konzistentní chování aplikace při opakovaném zpracování požadavků.

Celkově výsledky testování potvrzují, že navržená aplikace je funkční a stabilní.

6.3 Uživatelské testování

Pro ověření funkčnosti a použitelnosti aplikace bylo provedeno základní uživatelské testování. Testování se zaměřilo především na intuitivnost ovládání, srozumitelnost rozhraní a celkovou uživatelskou zkušenost.

Testování probíhalo formou scénářů, kdy uživatel plnil základní úkoly, jako je nahrání audio souboru, výběr modelu, spuštění přepisu a stažení výsledku. Byla sledována doba potřebná k dokončení úkolu a případné problémy při používání aplikace.

Z výsledků testování vyplynulo, že uživatelské rozhraní je obecně vnímáno jako přehledné a snadno použitelné. Pozitivně byla hodnocena zejména jednoduchost nahrávání souborů a přehledné zobrazení výsledků.

Naopak jako potenciální nedostatek byla identifikována absence průběžné indikace postupu přepisu, což může být problematické u delších audio souborů. Tento problém byl částečně řešen implementací vizuálního indikátoru načítání.

Výsledky testování byly využity k drobným úpravám rozhraní a potvrzují, že aplikace je vhodná pro běžné použití.

Kapitola 7

Další rozvoj aplikace

V současné době je k dispozici funkční verze aplikace, která splňuje stanovené minimální požadavky definované v zadání práce. Tato verze umožňuje základní provoz aplikace a ověřuje správnost zvoleného návrhu, architektury i použitých technologií.

Do budoucna je plánováno rozšíření aplikace o další funkce, které by zvýšily její uživatelskou přívětivost, funkcionalitu a praktické využití. Mezi tyto plánované úpravy patří zejména přepis v reálném čase, nabídka více různých modelů, možnost sumarizace a překladu textu.

7.1 Sumarizace textu

Sumarizace textu představuje jednu z klíčových úloh zpracování přirozeného jazyka, jejímž cílem je automaticky vytvořit zkrácenou reprezentaci vstupního textu, která zachovává jeho nejdůležitější informace a význam. V kontextu rostoucího množství digitálního obsahu nabývá automatická sumarizace na významu zejména v oblastech, jako jsou zpravodajství, akademické publikace, právní dokumenty, přepisy mluveného slova nebo zákaznická komunikace. Uživatelům umožňuje rychle získat přehled o obsahu rozsáhlých textů bez nutnosti jejich kompletního čtení.

V kontextu této aplikace by sumarizace představovala logické rozšíření funkce přepisu. Výstupem by nebyl pouze samotný přepis, ale i jeho strukturované shrnutí, případně rozdělené do tematických bloků nebo doplněné o klíčové body.

7.2 Nabídka více modelů

Jedním z možných směrů dalšího rozvoje aplikace je rozšíření podpory o širší nabídku ASR modelů, které by si uživatel mohl zvolit podle svých potřeb. Konkrétně se nabízí integrace modelů dostupných v rámci platforem Hugging Face, nebo NVIDIA NeMo, které v současnosti patří mezi nejpoužívanější a nejaktivněji rozvíjené platformy v oblasti zpracování přirozeného jazyka a řeči. Takový přístup by umožnil zvýšit flexibilitu aplikace a lépe ji přizpůsobit různým scénářům použití.

Platforma Hugging Face poskytuje rozsáhlý katalog předtrénovaných modelů pro široké spektrum úloh, včetně automatického rozpoznávání řeči, sumarizace textu, strojového překladu nebo klasifikace dokumentů. Tyto modely jsou obvykle distribuovány jako open-source a jsou doplněny o detailní dokumentaci, což usnadňuje jejich integraci do vlastních aplikací. Uživatelé by tak mohli mít možnost vybrat si mezi různými architekturami, ve-

likostmi modelů a jazykovou podporou v závislosti na dostupném výpočetním výkonu a požadované kvalitě výstupu.

Podobně platforma NVIDIA NeMo nabízí vysoce optimalizované modely zaměřené zejména na zpracování řeči a přirozeného jazyka, které jsou navrženy s ohledem na výkon a efektivní využití hardwarových prostředků. Modely z tohoto prostředí jsou často používány v profesionálních a výzkumných aplikacích, kde je kladen důraz na přesnost, stabilitu a možnost dalšího doladďování. Integrace těchto modelů by mohla být přínosná zejména pro uživatele, kteří disponují výkonnějším hardwarem a požadují vyšší kvalitu výstupů.

7.2.1 Přínosy výběru modelu uživatelem

Možnost volby konkrétního modelu přináší uživateli významnou míru kontroly nad chováním aplikace. Různé modely se mohou lišit nejen přesností, ale také rychlostí zpracování, nároky na paměť nebo podporovanými jazyky. Uživatel by si tak mohl zvolit např. menší a rychlejší model pro běžné použití, nebo naopak větší a přesnější model pro zpracování náročnějších vstupů, kde je klíčová kvalita výsledku.

Dalším přínosem je možnost experimentování a srovnávání jednotlivých modelů. Uživatelé by mohli přímo v aplikaci porovnávat výstupy různých modelů na stejných datech a vyhodnocovat jejich vhodnost pro konkrétní typ úloh. Tento přístup je výhodný nejen z praktického hlediska, ale také z pohledu vzdělávacího a výzkumného využití aplikace. Rozšíření o možnost výběru modelů lze také chápat jako přirozený krok dalšího vývoje, který by významně zvýšil praktičnost a dlouhodobou udržitelnost aplikace. Uživatel by nebyl vázán na konkrétní modely, které by v průběhu času mohly zastarat a ztratit svou aktuálnost či vhodnost pro dané úlohy.

7.2.2 Technický a architektonický kontext

Z technického hlediska by nabídka více modelů vyžadovala modulární architekturu aplikace, která umožní snadné přidávání a správu nových modelů bez nutnosti zásadních změn stávajícího řešení. Modely z Hugging Face i NVIDIA NeMo jsou zpravidla distribuovány ve formě balíčků nebo kontejnerů, což umožňuje jejich relativně snadné nasazení pomocí nástrojů pro správu kontejnerů, jako je Docker.

Důležitým aspektem je také standardizace rozhraní mezi aplikací a jednotlivými modely. Pokud by všechny modely poskytovaly jednotné vstupní a výstupní formáty, bylo by možné přepínat mezi nimi bez zásahu do uživatelského rozhraní. Takové řešení by zvýšilo rozšiřitelnost aplikace a umožnilo její další vývoj i po dokončení této práce.

7.2.3 Integrace modelu Whisper do prostředí NVIDIA NeMo

Jedním z uvažovaných směrů dalšího rozvoje aplikace je využití modelu OpenAI Whisper v kombinaci s nástroji platformy NVIDIA NeMo Toolkit. V případě, že by nabídka modelů v aplikaci byla z platformy NVIDIA NeMo. Model Whisper patří mezi nejrozšířenější open-source modely pro automatické rozpoznávání řeči a je ceněn zejména pro svou robustnost vůči šumu, podporu velkého množství jazyků a schopnost pracovat s reálnými i nekvalitními nahrávkami. Z těchto důvodů představuje vhodného kandidáta pro nasazení v aplikacích zaměřených na přepis mluveného slova.

Je však důležité zdůraznit, že Whisper není nativní součástí NVIDIA NeMo Toolkitu. NeMo je primárně určen pro vývoj, trénování a nasazování vlastních modelů v oblasti zpracování řeči, přirozeného jazyka a syntézy řeči, zatímco Whisper je samostatný model

vyvinutý společností OpenAI. Přímá integrace Whisperu do NeMo ve smyslu jeho trénování nebo správy pomocí interních NeMo API tedy v současné době není standardně podporována.

Přesto lze uvažovat o tom, že Whisper může být provozován a spravován v prostředí, které využívá principy a nástroje platformy NeMo. V praxi by se nejednalo o přímé trénování modelu Whisper pomocí NeMo, ale spíše o jeho nasazení jako samostatné inferenční služby, která je integrována do širší architektury aplikace. NeMo zde může sloužit jako referenční rámec pro správu modelů, práci s GPU, škálování nebo sjednocení rozhraní pro různé ASR systémy.

V budoucím rozšíření aplikace by tak bylo možné nabídnout uživatelům možnost volby mezi modely NeMo a modelem Whisper podle jejich konkrétních požadavků. Tento přístup by zvýšil flexibilitu systému a umožnil využít silné stránky obou řešení.

7.3 Přepis řeči v reálném čase

Přepis řeči v reálném čase představuje pokročilou funkcionalitu systémů automatického rozpoznávání řeči, jejímž cílem je převádět mluvený projev na text průběžně, tedy současně s tím, jak je řeč zaznamenávána. Na rozdíl od dávkového zpracování nahraných audio souborů, kdy je celý záznam nejprve uložen a následně analyzován, umožňuje přepis v reálném čase uživateli sledovat vznikající text s minimálním zpožděním. Tento přístup je zvláště přínosný v situacích, kde je důležitá okamžitá dostupnost informací, např. během online schůzek, přednášek nebo živých přenosů.

Z hlediska uživatelského komfortu představuje přepis v reálném čase významné zlepšení oproti tradičním metodám. Uživatel získává okamžitou zpětnou vazbu a může reagovat na obsah řeči bez nutnosti čekat na dokončení celého záznamu. Tato vlastnost zvyšuje interaktivitu systému a rozšiřuje spektrum jeho možného využití.

7.3.1 Technické principy realizace

Základem přepisu řeči v reálném čase je streamové zpracování audio signálu. Namísto práce s celým zvukovým souborem je audio vstup rozdělen na menší časové úseky, které jsou průběžně odesílány do systému automatického rozpoznávání řeči. Tyto úseky jsou následně zpracovávány ASR modelem, který generuje částečné nebo průběžné přepisy.

Důležitým aspektem je volba vhodného ASR modelu, který podporuje nízkou latenci a je schopen pracovat s neúplným kontextem. Některé modely jsou navrženy primárně pro dávkové zpracování a dosahují vysoké přesnosti až po zpracování celé nahrávky, zatímco jiné jsou optimalizovány právě pro streamové scénáře. Pro přepis v reálném čase je klíčová schopnost modelu postupně zpřesňovat výstup a případně opravovat dříve vygenerované části textu.

7.3.2 Architektura systému

Z architektonického hlediska je přepis v reálném čase obvykle realizován jako klient-server řešení. Klientská část zajišťuje snímání audio signálu z mikrofonu, jeho základní předzpracování a rozdělení na krátké úseky. Tyto úseky jsou následně odesílány na server, kde probíhá samotné rozpoznávání řeči pomocí ASR modelu. Výsledný text je pak v reálném čase vracen zpět klientovi a zobrazován uživateli.

Pro přenos audio dat se často využívají perzistentní komunikační kanály, např. WebSockety, které umožňují obousměrnou a nízkolatenční komunikaci. Tento přístup minimalizuje zpoždění a umožňuje plynulý přenos dat mezi klientem a serverem. Důležitým faktorem je rovněž efektivní správa výpočetních prostředků, zejména v případě, že systém obsluhuje více uživatelů současně.

7.3.3 Výhody přepisu v reálném čase

Jednou z hlavních výhod přepisu řeči v reálném čase je okamžitá dostupnost informací. Uživatel nemusí čekat na dokončení záznamu, ale může sledovat obsah řeči průběžně. Tato vlastnost je zásadní např. pro osoby se sluchovým postižením, pro které může živý přepis výrazně zlepšit dostupnost mluveného obsahu.

Další výhodou je možnost okamžitého zásahu do průběhu komunikace. V případě, že je přepis využíván během schůzky nebo prezentace, může uživatel reagovat na vznikající text, upřesnit nejasnosti nebo se vrátit k předchozím částem diskuse. Přepis v reálném čase rovněž umožňuje snadnější tvorbu poznámek a záznamů, které lze dále zpracovávat, např. sumarizovat nebo analyzovat.[1]

7.3.4 Omezení a technické výzvy

Přestože přepis v reálném čase přináší řadu výhod, je spojen i s určitými technickými omezeními. Jedním z hlavních problémů je kompromis mezi rychlostí a přesností. Modely, které generují výstup s velmi nízkou latencí, mohou dosahovat nižší přesnosti, zejména v případě složitějších vět nebo rychlé řeči. Další výzvou je práce s neúplným kontextem, kdy model nemá k dispozici celý výrok a musí předvídat jeho další vývoj.

Významnou roli hraje také kvalita vstupního audio signálu. Šum, ozvěna nebo překrývání hlasů mohou negativně ovlivnit přesnost přepisu, což je v reálném čase obtížnější korigovat. Z těchto důvodů je často nutné kombinovat ASR modely s dalšími technikami, jako je detekce řeči, filtrace šumu nebo následná korekce přepisu.

7.3.5 Perspektivy dalšího využití

Přepis řeči v reálném čase lze chápat jako základní stavební prvek pro další pokročilé funkce. Na vznikající text je možné navázat např. automatickou sumarizaci nebo detekci klíčových témat. V kombinaci s dalšími jazykovými modely může systém poskytovat uživateli strukturované informace již během probíhající komunikace.

V kontextu této práce představuje implementace přepisu v reálném čase perspektivní směr dalšího rozvoje aplikace. Ačkoli současná verze systému splňuje základní požadavky, rozšíření o tuto funkcionalitu by výrazně zvýšilo její praktickou hodnotu.

7.4 Segmentace a diarizace mluvčích (speaker segmentation and diarization)

Segmentace mluvčích umožňuje rozdělit audio záznam na části podle jednotlivých mluvčích. Pokud je kombinována s identifikací, mluvíme o tzv. diarizaci mluvčích, která poskytuje informace o tom, kdo a kdy hovořil.

Tato funkce má zásadní význam v aplikacích, které pracují s rozhovory nebo skupinovými diskusemi, např. při přepisu schůzek, interview nebo konferenčních hovorů. Segmentace mluvčích umožňuje přehlednější strukturování přepisu a usnadňuje jeho další analýzu.

Technicky proces segmentace a diarizace zahrnuje několik kroků. Nejdříve probíhá detekce řeči, která identifikuje části nahrávky obsahující řeč oproti tichu nebo šumu. Následně se z jednotlivých segmentů extrahují akustické vlastnosti. Segmenty jsou pak seskupovány podle podobnosti hlasů, přičemž moderní metody využívají embeddingy mluvčích a algoritmy typu k-means, spectral clustering či neuronové sítě, které dokážou automaticky rozlišit různé mluvčí v nahrávce. Nakonec probíhá post-processing, při kterém se výsledné segmenty upravují, aby bylo minimalizováno přepínání mezi mluvčími v krátkých intervalech a aby byl výstup konzistentní po celé délce záznamu.

Společnost NVIDIA nabízí v rámci svého NeMo Toolkitu robustní řešení pro diarizaci mluvčích, které zahrnuje předtrénované modely. Tyto nástroje umožňují automatickou detekci hranic mezi mluvčími, identifikaci jednotlivých mluvčích bez nutnosti předchozího tréninku na konkrétní osoby, práci s vícekanálovým audio vstupem, což výrazně zvyšuje přesnost při překrývání hlasů, a snadnou integraci do aplikací využívajících ASR modely, např. pro přepis v reálném čase.

Využití předtrénovaného nástroje, jako je NeMo, přináší významné výhody. Zkracuje vývojový cyklus, protože není potřeba modely trénovat od nuly, čímž šetří čas a výpočetní zdroje. Předtrénované modely navíc zajišťují konzistenci a robustnost napříč různými typy nahrávek, včetně vícekanálového audio vstupu či šumu.[7]

7.5 Vstupních formáty

V aktuální verzi aplikace je práce se zvukovými daty omezena primárně na formát WAV, který je z hlediska zpracování řeči výhodný díky své nekomprimované podobě a jednoduché interpretaci. Pro širší praktické využití aplikace by však bylo vhodné rozšířit podporu o další běžně používané audio formáty, jako jsou např. MP3, FLAC, OGG nebo M4A.

Rozšíření o více vstupních formátů by výrazně zvýšilo uživatelský komfort, protože uživatelé by nemuseli před nahráním souborů provádět jejich manuální konverzi do podporovaného formátu. V praxi se totiž audio data často vyskytují v komprimovaných formátech, které jsou optimalizovány pro ukládání nebo přenos, nikoli pro přímé zpracování ASR systémy.

7.6 Výstupní formáty

V současné verzi aplikace je výstup přepisu dostupný ve formátu TXT, který představuje nejjednodušší a nejuniverzálnější způsob práce s textovými daty. Tento formát umožňuje snadné zobrazení, kopírování i další úpravy přepisu bez nutnosti použití specializovaných nástrojů. Zároveň je kompatibilní s širokým spektrem aplikací a operačních systémů.

Pro širší praktické využití aplikace by však bylo vhodné rozšířit podporu o další výstupní formáty, které by umožnily efektivnější práci s přepisem v různých scénářích. Jedná se zejména o strukturované formáty, jako jsou JSON nebo XML, které kromě samotného textu umožňují ukládat i doplňující informace, např. časové značky, segmentaci řeči, identifikaci mluvčích nebo míru jistoty rozpoznání. Tyto formáty jsou vhodné především pro další automatizované zpracování nebo integraci s jinými systémy.

Pro koncové uživatele by mohl být přínosný také export do formátu DOCX, který umožňuje přehledné formátování textu a jeho další úpravu v kancelářských nástrojích. Tento formát je vhodný zejména pro oficiální dokumenty, zápisy z jednání nebo archivaci přepisů ve strukturované podobě.

Rozšíření o více výstupních formátů by tak významně zvýšilo flexibilitu aplikace a umožnilo její využití v širším spektru praktických situací, od jednoduchého čtení přepisu až po jeho integraci do komplexních informačních systémů.

7.7 Překlad

V oblasti automatického rozpoznávání řeči představuje samotný přepis mluvené řeči pouze jeden z možných výstupů. Pro zvýšení praktičnosti a univerzálnosti ASR systémů by bylo velmi přínosné, kdyby součástí těchto systémů byly také integrované nástroje pro automatický překlad. Taková funkce by umožnila nejen získat přepis v původním jazyce, ale rovnou jej převést do jiného cílového jazyka, což výrazně rozšiřuje možnosti využití přepisu např. ve vícejazyčném prostředí, při mezinárodních konferencích, online kurzech či mediálních produkcích.

Integrace překladových modulů přímo do ASR systému nabízí několik významných výhod. Především eliminuje potřebu využívat samostatné překladové služby, čímž se zjednodušuje pracovní postup a zkracuje čas potřebný k získání plně zpracovaného výstupu. Navíc je možné zajistit lepší konzistenci terminologie a stylistiky, protože překlad může být proveden s vědomím kontextu původního přepisu a případně i s využitím metadat, jako jsou informace o mluvčích nebo segmentace řeči. To je zvláště důležité v odborných či technických nahrávkách, kde standardní překladové nástroje mohou generovat nesprávné výrazy nebo neúplné překlady.

Některé moderní modely a platformy již podobnou funkcionalitu částečně implementují. Např. modely typu Canary nebo OpenAI Whisper mají zabudovanou schopnost přepisu a překladů. Takové propojení ASR a automatického překladu poskytuje uživatelům vysokou flexibilitu. Nejen že mohou získat přepis v originálním jazyce, ale zároveň jej mohou okamžitě využít pro mezinárodní komunikaci či lokalizaci obsahu.

Dlouhodobě by bylo možné tyto překladové nástroje kombinovat s dalšími funkcemi ASR, např. s diarizací mluvčích, časovými značkami či strukturou formátů výstupu (JSON, XML, SRT, DOCX), čímž by vznikl komplexní systém schopný produkovat přehledný, synchronizovaný, vícejazyčný přepis. Takový systém by byl výrazně univerzálnější než tradiční ASR, protože by pokrýval celou řadu praktických scénářů, od překladů konferencí a rozhovorů až po titulkování videí či mediální obsah v několika jazycích.

V ideálním případě by uživatelé mohli zvolit cílový jazyk při zadání nahrávky a získat okamžitě přepis v původním i přeloženém jazyce. Tato funkcionalita by zároveň umožnila snadnou integraci do dalších aplikací, např. do systémů pro automatickou sumarizaci, vyhledávání informací, tvorbu poznámek či vícejazyčné titulkování. Kombinace ASR a integrovaného překladu tak představuje významný krok směrem k plně automatizovaným řešením pro zpracování mluvené řeči a její adaptaci pro globální uživatele.

7.8 Ukládání historie přepisů

Další významnou funkcionalitou by mohlo být ukládání historie přepisů. V současné verzi aplikace uživatel pracuje se soubory a jejich aktuálními výstupy, avšak není k dispozici historie předchozích přepisů.

Rozšíření o historii by umožnilo uchovávat více verzí přepisů pro stejný soubor, porovnávat výsledky různých modelů a sledovat změny při úpravě parametrů. Technicky by bylo možné tuto funkcionalitu realizovat pomocí jednoduché databáze, která by ukládala metadata o přepisech včetně použitých parametrů.

7.9 Paralelní zpracování a dávkové úlohy

V současné implementaci je přepis prováděn sekvenčně, což může být limitující při práci s větším množstvím souborů. Do budoucna by bylo možné implementovat podporu pro dávkové zpracování (batch processing), kdy by uživatel mohl nahrát více souborů najednou a nechat je zpracovat automaticky.

Dalším krokem by bylo zavedení paralelního zpracování, které by využívalo více vláken nebo GPU jednotek. Tento přístup by umožnil výrazně zkrátit dobu zpracování, efektivněji využít dostupný hardware a škálovat aplikaci pro náročnější použití.

7.10 Webové API

Dalším významným rozšířením by bylo vytvoření veřejného nebo lokálního API rozhraní, které by umožnilo integraci aplikace s dalšími systémy. Namísto použití pouze přes webové rozhraní by bylo možné aplikaci využívat jako službu.

API by mohlo umožňovat odesílání audio souborů pomocí HTTP požadavků, spouštění přepisu s definovanými parametry a získání výsledků ve strukturovaném formátu. Tato funkcionalita by otevřela cestu k integraci do dalších aplikací, např. systémů pro správu dokumentů nebo analytických nástrojů.

7.11 Shrnutí dalšího rozvoje aplikace

Možnosti dalšího rozvoje aplikace jsou velmi široké a zahrnují jak rozšíření funkcionality, tak zlepšení výkonu a uživatelského komfortu. Navržená architektura aplikace umožňuje tyto změny implementovat postupně bez zásadních zásahů do stávajícího řešení.

Díky modularitě systému lze očekávat, že aplikace může být v budoucnu rozšířena o pokročilé funkce, jako je real-time přepis, automatická sumarizace, překlad nebo integrace s externími systémy. Tím by se mohla stát komplexním nástrojem pro zpracování mluvené řeči v různých aplikačních scénářích.

Kapitola 8

Závěr

Cílem této práce bylo navrhnout a implementovat webovou aplikaci pro automatický přepis řeči, která běží na platformě Umbrel OS a umožňuje uživatelsky přívětivým způsobem převádět zvukové nahrávky do textové podoby. Důraz byl kladen především na modularitu systému, možnost konfigurace a podporu více modelů pro rozpoznávání řeči.

V rámci práce byla vytvořena aplikace umožňující nahrávání a správu zvukových souborů ve formátu WAV, jejich přehrávání a následný přepis pomocí moderních ASR modelů. Implementace zahrnuje podporu několika modelů, konkrétně Whisper, Canary a Parakeet, které lze mezi sebou dynamicky přepínat a konfigurovat pomocí dostupných parametrů. Tento přístup umožňuje uživateli porovnávat jednotlivé modely z hlediska kvality přepisu, rychlosti i výpočetní náročnosti.

Výsledná aplikace poskytuje přehledné uživatelské rozhraní, ve kterém lze zobrazit výsledný přepis a stáhnout jej ve formátu TXT. V případě modelu Whisper je navíc možné získat detailnější informace o přepisu, jako jsou časové značky jednotlivých segmentů a odhad míry jistoty.

Během implementace byly řešeny i praktické problémy spojené s integrací knihoven pro strojové učení, zejména kompatibilita verzí závislostí, správa výpočetních prostředků a stabilita běhu jednotlivých modelů. Tyto aspekty ukazují, že nasazení moderních ASR systémů v reálné aplikaci vyžaduje nejen teoretické znalosti, ale i praktické zkušenosti s jejich integrací a optimalizací.

Navržené řešení představuje funkční aplikaci, která demonstruje možnosti současných technologií pro automatický přepis řeči a zároveň poskytuje základ pro další rozvoj. Do budoucna lze aplikaci rozšířit např. o podporu dalších modelů, nové formáty vstupních dat nebo pokročilejší způsoby zpracování výstupu.

Celkově lze konstatovat, že vytvořený systém splňuje stanovené cíle a představuje flexibilní nástroj pro experimentování s různými přístupy k automatickému rozpoznávání řeči, přičemž jeho architektura umožňuje další rozšiřování a přizpůsobení konkrétním potřebám uživatele.

Odkaz na Umbrel obchod: `https://github.com/danieldedek/shop` a odkaz na aplikaci `https://github.com/danieldedek/transcribe_app`.

Literatura

[1] AssemblyAI. *What streaming speech to text model is best for voice agents and why?* online. 2026. Dostupné z: https://www.assemblyai.com/blog/real-time-speech-to-text-best-for-voice-agents. Odborný blog popisující výhody real-time přepisu řeči.

[2] Baek, J.; Hussain, A.; Liu, D.; Vincent, N. a Kim, L. H. *Open WebUI: An open, extensible, and usable interface for AI interaction* online. 2025. Dostupné z: https://arxiv.org/pdf/2510.02546. [cit. 2026-05-06].

[3] BuoyBitcoin. *Start9 vs Umbrel* online. 2026. Dostupné z: https://buoybitcoin.com/compare/start9-vs-umbrel.html. [cit. 2026-05-06]. Informativní srovnávací článek.

[4] getumbrel. *Umbrel App Framework* online. 2025. Dostupné z: https://github.com/getumbrel/umbrel-apps. [cit. 2026-01-02]. GitHub repotitář.

[5] NVIDIA. *Canary 180M Flash Model card* online. Dostupné z: https://huggingface.co/nvidia/canary-180m-flash. [cit. 2026-05-06]. Hugging Face modelová karta.

[6] NVIDIA. *Parakeet TDT 0.6B v2 Model card* online. Dostupné z: https://build.nvidia.com/nvidia/parakeet-tdt-0_6b-v2/modelcard. [cit. 2026-05-06]. NVIDIA modelová karta.

[7] NVIDIA. *Speaker Diarization – NeMo Framework User Guide* online. Dostupné z: https://docs.nvidia.com/nemo-framework/user-guide/latest/nemotoolkit/asr/speaker_diarization/intro.html. [cit. 2026-01-02]. Oficiální dokumentace NVIDIA NeMo.

[8] Start9 Labs. *About Start9* online. Dostupné z: https://start9.com/about/. Oficiální stránka popisující platformu Start9.

[9] SYSTRAN. *Faster-whisper: Faster Whisper transcription with CTranslate2* online. Dostupné z: https://github.com/SYSTRAN/faster-whisper. [cit. 2026-05-06]. GitHub repotitář.

[10] The Sovereign Builder. *Umbrel Review: The Decentralized Personal Computer for Real Digital Sovereignty* online. 2025. Dostupné z: https://www.onnetwork.io/umbrel-review-the-decentralized-personal-computer-for-real-digital-sovereignty/. [cit. 2026-05-06]. Technická recenze Umbrel OS.

[11] Umbrel. *What is Umbrel?* online. 2026. Dostupné z: https://umbrel.com/support/getting-started/what-is-umbrel. [cit. 2026-05-06]. Oficiální Umbrel dokumentace.