VYSOKÁ ŠKOLA POLYTECHNICKÁ JIHLAVA

Aplikovaná informatika

APLIKACE PRO PŘEVOD PSANÉHO TEXTU NA DIGITÁLNĚ UPRAVITELNÝ

Bakalářská práce

Autor práce: Lukáš Velek

Vedoucí práce: Ing. Mgr. Michal Jeřábek, Ph.D.

Jihlava 2026

Vysoká škola polytechnická Jihlava

Tolstého 16, 586 01 Jihlava

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Autor práce:	**Lukáš Velek**
Studijní program:	Aplikovaná informatika
Garant studijního programu:	Ing. Lenka Kuklišová Pavelková, Ph.D.
Název práce:	**Aplikace pro převod psaného textu na digitálně upravitelný**
Vedoucí práce:	Ing. Mgr. Michal Jeřábek, Ph.D.
Cíl práce:	Cílem práce je vytvořit aplikaci pro převod psaného textu na digitálně upravitelný. Součástí práce je zmapování existujících řešení, seznámení se s nástroji AI v jazyce Python použitelnými pro převod, zaměření se na vliv kvality vstupních materiálů na výsledek a ověření funkčnosti aplikace na vybraném dokumentu většího rozsahu.

Abstrakt

Bakalářská práce se zabývá problematikou optického rozpoznávání znaků (OCR), jeho historickým vývojem, principy fungování a současnými možnostmi využití v praxi. Pozornost je věnována přehledu aktuálně dostupných OCR řešení na trhu a jejich srovnání z hlediska funkčnosti a dostupnosti. Dále se práce zaměřuje na využití programovacího jazyka Python při tvorbě OCR aplikací, přičemž představuje vybrané knihovny pro rozpoznávání textu, úpravu a předzpracování obrazu a tvorbu grafického uživatelského rozhraní. Součástí práce je návrh a implementace vlastní OCR aplikace, včetně popisu použitého kódu, uživatelského rozhraní a testování funkčnosti. V závěru jsou diskutovány problémy a omezení, které se vyskytly během vývoje aplikace.

Klíčová slova

Jazykový model

Optické rozpoznávání znaků

Python

Abstract

The bachelor's thesis deals with the topic of optical character recognition (OCR), its historical development, principles of operation, and current possibilities of practical use. Attention is given to an overview of currently available OCR solutions on the market and their comparison in terms of functionality and availability. Furthermore, the thesis focuses on the use of the Python programming language in the development of OCR applications, presenting selected libraries for text recognition, image processing and preprocessing, and graphical user interface creation. The thesis also includes the design and implementation of a custom OCR application, including a description of the used code, the user interface, and functionality testing. In conclusion, the problems and limitations encountered during the application development are discussed.

Keywords

Language model

Optical Character Recognition

Python

Prohlašuji, že předložená bakalářská práce je původní a zpracoval/a jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem v práci neporušil/a autorská práva (ve smyslu zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů, v platném znění, dále též „**AZ**“).

Byl/a jsem seznámen/a s tím, že na mou bakalářskou práci se plně vztahuje **AZ**, zejména § 60 (školní dílo).

Podle § 47b zákona o vysokých školách souhlasím se zveřejněním své práce podle Směrnice pro vedení, vypracování a zveřejňování závěrečných prací na VŠPJ, a to bez ohledu na výsledek obhajoby.

Beru na vědomí, že VŠPJ má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom/a toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem VŠPJ, která má právo ode mě požadovat přiměřený příspěvek na úhradu nákladů, vynaložených vysokou školou na vytvoření díla (až do jejich skutečné výše), z výdělku dosaženého v souvislosti s užitím díla či poskytnutím licence.

V Jihlavě dne 1. dubna 2026

…….

Podpis studenta/ky

Poděkování

Rád bych tímto poděkoval vedoucímu mé bakalářské práce Ing. Mgr. Michalu Jeřábkovi, Ph.D. za odborné vedení, cenné rady a připomínky, které mi poskytoval v průběhu zpracování této práce.

Obsah

Seznam obrázků

Seznam zkratek

ASCII	American Standard Code for Information Interchange
CER	Character Error Rate
GUI	Graphical user interface
HTR	Handwritten Text Recognition
ICR	Intelligent character recognition
LER	Line Error Rate
OCR	Optical character recognition
WER	Word Error Rate

Úvod

Cílem této bakalářské práce je vytvořit aplikaci schopnou převádět text z obrazového souboru do digitálně upravitelné podoby. V úvodní části práce je představena technologie OCR (Optical Character Recognition), která je pro tento převod klíčová. Stručně je popsána její historie, princip fungování i faktory, které mohou ovlivnit přesnost rozpoznávání textu, jako je kvalita vstupního dokumentu nebo použité metody zpracování obrazu.

Následující kapitola se zaměřuje na přehled současných řešení dostupných na trhu, která umožňují převod textu z obrazových dat. Samotná aplikace je realizována v jazyce Python s využitím specializovaných knihoven pro zpracování obrazu a rozpoznávání textu, jejichž popis je uveden v samostatné části práce.

Praktická část práce se věnuje návrhu a implementaci aplikace. Nejprve jsou představeny použité knihovny a jazykové modely, poté je popsána struktura aplikace a jednotlivé kroky jejího fungování. Dále je detailně rozebrán proces vývoje aplikace a trénování jazykového modelu. Výsledky po jednotlivých tréninkových cyklech jsou následně porovnány jak s původním modelem před trénováním, tak s výstupy vybraného online nástroje pro rozpoznávání textu.

V závěru práce jsou shrnuty dosažené výsledky a stručně popsány problémy, se kterými bylo nutné se během vývoje aplikace vypořádat.

1 Optical Character Recognition (OCR)

OCR představuje technologii umožňující převod textu z obrázků či naskenovaných dokumentů do strojově čitelné podoby, čímž výrazně usnadňuje práci s daty v moderních organizacích. Od svých počátků v první polovině 20. století, kdy se objevily první nástroje schopné převádět znaky na kód, až po dnešní vysoce přesné systémy využívající umělou inteligenci, prošlo OCR zásadním technologickým vývojem. Tato kapitola se proto nejprve zaměřuje na historický vývoj technologie a následně popisuje základní principy jejího fungování, včetně jednotlivých kroků zpracování obrazu a rozpoznávání textu.

Dále je v kapitole věnována pozornost různým typům OCR systémů a jejich schopnostem, stejně jako otázce přesnosti rozpoznávání, která je klíčovým faktorem pro praktické nasazení této technologie. Popsány jsou hlavní faktory ovlivňující úspěšnost OCR, metody hodnocení kvality výstupu pomocí standardizovaných metrik a příklady reálného využití OCR v různých oblastech.

1.1 Historie

První náznaky optického rozpoznávání znaků můžeme nalézt už v dobách první světové války. Fyzik Emanuel Goldberg vytvořil nástroj, který bych schopen rozpoznat znaky v daném dokumentu a následně je převézt na telegrafní kód. V roce 1974 zakládá Ray Kurzweil společnost Kurzweil Computer Products, Inc. Firma uvádí na trh produkt, který byl schopen rozpoznat písmo v jakémkoliv fontu. Ray Kurzweil následně vytváří nástroj, který byl schopný pomáhat nevidomým osobám. Vzniká takzvané “text-to speech” zařízení, které je schopno převést text na řeč. V roce 1980 je následně společnost prodána Xeroxu, který tuto technologii plánuje více komercializovat. Technologie OCR následně nabírá na popularitě v devadesátých letech, kdy se díky ní převádí do digitální podoby historické noviny. K dnešnímu dni se technologie OCR značně zlepšila a ke konverzi dochází s vysokou přesností. (Holdsworth, 2025)

1.2 Jak funguje OCR?

Proces rozpoznávání textu se skládá z několika kroků, které zajišťují přesné převedení naskenovaného dokumentu do digitální podoby. V následující části budou popsány jednotlivé fáze zpracování obrazu od jeho úpravy přes rozpoznání znaků až po finální generování výstupu.

Před samotným rozpoznáním textu musí dojít k úpravě vstupních dokumentů. Po nahrání do programu jsou dokumenty převedeny do černobílé podoby. Konverzí program pozná, co je třeba v dokumentu rozpoznat. Tmavé obrysy symboly, které jsou třeba rozpoznat. Bílé obrysy znázorňují pozadí.

Následně dochází k předzpracování obrazu, které provádí dodatečnou úpravu souborů. Dochází k odebrání nepotřebných pixelů, nežádoucích čar a jiných objektů. Zároveň dochází k zarovnání dokumentů, které mohlo být způsobeno špatným naskenováním dokumentů.

Jakmile je vstupní dokument upraven, dochází k samotnému rozpoznání textu. K tomu mohou být využity dva algoritmy. První algoritmus rozpoznává text pomocí vzorů. Program obsahuje vzory (glyfy), které následně porovnává s naskenovaným dokumentem. Pro správnou funkci musí být písmo v dokumentu ve formátu, ve kterém je program naučen. Druhý algoritmus

rozpoznává text pomocí křivek. Zde již program trénován být nemusí. Rozpoznání se provádí na základě vlastností jednotlivých znaků. Počet linek, zakřivení atd. Po identifikaci je znak převeden do formátu ASCII se kterým systém následně pracuje. (Holdsworth, 2025)

1.3 Typy OCR

OCR programy lze rozdělit do několika typů podle úrovně jejich vyspělosti a způsobu rozpoznávání textu. Následující část se zaměřuje na základní přehled těchto typů – od jednoduchého rozpoznávání jednotlivých znaků až po pokročilé metody využívající umělou inteligenci k identifikaci celých slov.

Jednoduché OCR pracuje na principu rozpoznávání jednotlivých znaků pomocí porovnávání s předem uloženými vzory, tzv. glyfy. Každý znak je analyzován samostatně a porovnáván s databází známých tvarů. Tento přístup je funkční pouze tehdy, pokud je použitý font, již systému známý, což výrazně omezuje množství dokumentů, které lze tímto způsobem zpracovat, zejména při různorodosti písem a jazyků.

Optické rozpoznávání značek se nezaměřuje na klasický text, ale na identifikaci značek a symbolů. Používá se především k detekci zaškrtnutých políček, vyplněných bublin v dotaznících, podpisů, log, symbolů nebo vodoznaků. Principem je porovnání zachyceného obrazu s uloženými vzory, podobně jako u jednoduchého OCR, avšak s jiným zaměřením.

ICR (Intelligent character recognition) představuje pokročilejší formu OCR, která využívá umělou inteligenci a strojové učení. Systém se postupně učí rozpoznávat znaky podobně jako člověk, a to na základě jejich charakteristických vlastností, jako jsou křivky, průsečíky, linie nebo smyčky. Díky tomu dokáže pracovat i s neznámými fonty nebo rukopisem a dosahuje vyšší flexibility než tradiční OCR.

Inteligentní rozpoznávání slov je dalším vývojovým krokem ICR, kdy systém nerozpoznává jednotlivé znaky, ale celé slovo jako jeden celek. Tento přístup výrazně zrychluje proces rozpoznávání a zvyšuje jeho přesnost, zejména u strukturovaných dokumentů, kde lze lépe využít kontext a celkový tvar slov. (Holdsworth, 2025)

1.4 Přesnost OCR

Přesnost OCR vyjadřuje, jak věrně dokáže systém převést text z obrázků nebo PDF dokumentů do strojově čitelné podoby. Ideálním výsledkem je text zcela totožný s originálem, avšak v reálném použití se i u kvalitních OCR nástrojů běžně objevují drobné chyby, například záměny znaků, chybějící mezery nebo nesprávné rozdělení slov. Tyto nepřesnosti mohou negativně ovlivnit další zpracování dat, vyhledávání informací nebo automatizované analýzy.

Pro objektivní hodnocení kvality OCR se využívá porovnání výstupu systému s tzv. „ground truth“, tedy referenčním bezchybným přepisem původního dokumentu. Na tomto základě se počítají standardizované metriky, mezi které patří především míra chyb znaků (Character Error Rate), míra chyb slov (Word Error Rate) a případně míra chyb řádků (Line Error Rate). Tyto ukazatele umožňují srovnávání různých OCR řešení a slouží také k ověřování zlepšení rozpoznávacích algoritmů. (Martinez, 2025)

1.4.1 Faktory ovlivňující přesnost rozpoznání

Přesnost OCR je ovlivněna řadou faktorů souvisejících jak s kvalitou vstupního dokumentu, tak s použitou technologií rozpoznávání. Mezi klíčové vlivy patří stav dokumentu, kdy skvrny, vyblednutí, poškození nebo artefakty výrazně snižují úspěšnost rozpoznání, a také kvalita obrazu, zejména rozlišení, zkreslení, šum, nerovnoměrné osvětlení či natočení při skenování. Důležitou roli hraje čitelnost textu – malé písmo, ozdobné fonty, rukopis nebo nízký kontrast mezi textem a pozadím zvyšují chybovost, zatímco jednoduché fonty a vysoký kontrast přesnost zlepšují.

Významným faktorem je rovněž struktura dokumentu, protože složité rozvržení s tabulkami, sloupci nebo kombinací textu a obrázků klade vyšší nároky na analýzu rozložení. Přesnost ovlivňuje také samotný OCR software, konkrétně kvalita použitých algoritmů, jazykových modelů, trénovacích dat a schopnost práce s různými jazyky či znakovými sadami. Dalšími aspekty jsou typ použitého skenovacího zařízení, podmínky při skenování, zvolený formát souboru a využití předzpracovacích technik, jako je binarizace, odstranění šumu nebo úprava kontrastu. Systematickým sledováním příčin chyb lze tyto faktory cíleně optimalizovat a dosáhnout vyšší přesnosti OCR převodu. (Martinez, 2025)

1.4.2 Character Error Rate (CER)

Míra chybovosti znaků (CER) patří mezi základní metriky používané pro hodnocení přesnosti výstupu systémů OCR a HTR (Handwritten text recognition). CER vyjadřuje poměr chybně rozpoznaných znaků vůči celkovému počtu znaků ve správném (referenčním) textu, a to v rámci celého testovacího souboru dokumentů. Do chyb jsou započítávány záměny znaků, jejich vynechání i vložení nadbytečných znaků.

Hodnota CER se vypočítá jako podíl počtu chyb na celkovém počtu znaků v referenčním textu. Čím nižší je hodnota CER, tím vyšší je kvalita rozpoznání, přičemž ideální hodnota je 0 %, která odpovídá bezchybné konverzi. U kvalitních a čistých skenů v anglickém jazyce se běžné hodnoty CER pohybují přibližně v rozmezí 2–10 %, přičemž u historických dokumentů nebo ručně psaného textu bývají tyto hodnoty zpravidla vyšší. (Martinez, 2025)

1.4.3 Word Error Rate (WER)

Míra chybovosti slov (WER) je další důležitou metrikou pro hodnocení přesnosti systémů OCR a HTR. WER vyjadřuje podíl slov, která obsahují alespoň jednu chybu, ve srovnání s referenčním (správným) textem. Za chybu je považováno každé slovo, ve kterém došlo k nesprávnému rozpoznání jednoho nebo více znaků.

Hodnota WER se vypočítá jako poměr počtu chybně rozpoznaných slov k celkovému počtu slov v referenčním textu. Stejně jako u CER platí, že nižší hodnota WER znamená vyšší přesnost rozpoznávání, přičemž cílové hodnoty se obvykle pohybují pod hranicí 5 %. Společné sledování metrik CER a WER poskytuje komplexní pohled na kvalitu výstupu OCR, protože umožňuje hodnotit přesnost jak na úrovni jednotlivých znaků, tak i celých slov. (Martinez, 2025)

1.5 Příklady použití

OCR technologie pomáhá podnikům napříč odvětvími automatizovat zpracování dokumentů, čímž snižuje ruční práci, chyby a náklady, a zároveň zvyšuje produktivitu i bezpečnost díky digitalizaci a centralizaci dat. Ve vzdělávání podporuje studenty převodem textu na řeč, formátováním či zvýrazňováním textu a je užitečná i pro dyslektiky. Ve finančním sektoru zrychluje a zpřesňuje ověřování dokumentů, snižuje podvody a umožňuje automatickou extrakci dat z bankovních výpisů či smluv. Ve zdravotnictví zefektivňuje správu pacientské dokumentace a minimalizuje chyby při zadávání údajů. V logistice automatizuje zpracování přepravních dokumentů, zrychluje tok zboží, zlepšuje přesnost a poskytuje aktuální přehledy. (Rossum, 2023)

2 Současná řešení na trhu

Před samotným návrhem aplikace je vhodné se seznámit s aktuálními nástroji a technologiemi dostupnými na trhu, které umožňují převod obrazových dokumentů do digitální textové podoby. Technologie optického rozpoznávání znaků (OCR) je dnes široce využívána v různých oblastech, například při digitalizaci archivních materiálů, zpracování administrativních dokumentů nebo automatizaci práce s daty. Moderní OCR řešení často využívají metody umělé inteligence a strojového učení, díky nimž dokážou rozpoznávat nejen tištěný text, ale také rukopis či dokumenty se složitým rozvržením.

Následující kapitola představuje přehled vybraných nástrojů a služeb, které jsou v současnosti využívány pro rozpoznávání textu z dokumentů a obrazových souborů. Text se zaměřuje jak na specializované platformy určené pro práci s historickými dokumenty, tak na cloudové služby nebo aplikace využívané v podnikové praxi. U jednotlivých nástrojů jsou stručně popsány jejich hlavní funkce, možnosti využití a základní principy fungování.

2.1 Transkribus

Transkribus je platforma využívající umělou inteligenci, která slouží k přepisu, rozpoznávání a digitalizaci ručně psaných, strojopisných i tištěných historických dokumentů. Původně byla vyvinuta Univerzitou v Innsbrucku a v současnosti je spravována organizací READ-COOP. Platforma umožňuje převádět obrazové archivy do digitální, prohledatelné textové podoby a je zaměřena především na práci s historickými materiály různého stáří.

Mezi hlavní funkce Transkribusu patří rozpoznávání rukopisu (HTR) schopné pracovat s různými styly písma, včetně historických skriptů, jako jsou Kurrent nebo Sütterlin. Kromě samotného přepisu textu nabízí nástroje pro analýzu rozložení dokumentu, které umožňují detekci textových oblastí, sloupců či tabulek. Uživatelé mohou využívat širokou škálu předtrénovaných jazykových modelů nebo si vytvořit vlastní modely pro specifické dokumenty. Platforma je dostupná prostřednictvím webového rozhraní i rozhraní API a je využívána zejména ve výzkumu, archivnictví, knihovnách a genealogii, přičemž zpracování dokumentů probíhá na základě kreditního systému. (Read-Coop, 2026)

2.2 Amazon Textract

Amazon Textract je cloudová služba využívající umělou inteligenci, která umožňuje automatické rozpoznávání a analýzu textu v dokumentech. Dokáže detekovat tištěný i ručně psaný text v různých typech dokumentů, například ve finančních výkazech, lékařských záznamech nebo daňových formulářích. Kromě samotného rozpoznání textu nabízí pokročilé funkce pro extrakci strukturovaných dat, jako jsou formuláře, tabulky, faktury, účtenky či identifikační doklady. Umožňuje také cílené vyhledávání konkrétních informací pomocí funkce Queries a automatizované zpracování specializovaných dokumentů, například úvěrových smluv. Služba je dostupná prostřednictvím jednoduchého API a nevyžaduje znalosti strojového učení.

Amazon Textract je postaven na škálovatelných technologiích hlubokého učení a je navržen pro integraci do webových, mobilních i podnikových aplikací. Mezi hlavní přínosy patří možnost rychlé analýzy velkého množství dokumentů, automatizace zpracování dat z formulářů a podpora následného využití dat například v systémech zpracování přirozeného jazyka (NLP). Služba funguje na principu platby za zpracované dokumenty bez vstupních nákladů a podporuje jak synchronní zpracování jednorázových dokumentů, tak asynchronní analýzu vícestránkových souborů. (Amazon, 2026)

2.3 Microsoft Azure Document Intelligence

Azure Document Intelligence (dříve Form Recognizer) je cloudová služba platformy Microsoft Azure využívající umělou inteligenci pro automatické získávání textu, tabulek, struktury dokumentu a dvojic klíč–hodnota z digitálních i naskenovaných dokumentů (např. PDF nebo obrázků). Nabízí předtrénované modely pro běžné typy dokumentů, jako jsou faktury, účtenky, identifikační doklady či daňové formuláře, a zároveň umožňuje vytvářet vlastní modely přizpůsobené specifickým potřebám organizace. Součástí řešení je také analýza rozložení dokumentu a podpora rozpoznávání tištěného i ručně psaného textu.

Služba je navržena především pro automatizaci firemních procesů, například při zpracování formulářů, bankovní dokumentace nebo faktur, kde nahrazuje ruční přepisování dat a snižuje chybovost. Poskytuje nástroje pro klasifikaci dokumentů, bezpečné řízení přístupu prostřednictvím Azure Active Directory a snadnou integraci do aplikací pomocí SDK nebo grafického rozhraní bez nutnosti programování. Azure Document Intelligence funguje na škálovatelném cenovém modelu podle objemu zpracovaných dokumentů a je součástí širšího ekosystému nástrojů Azure AI pro práci s daty řízenými dokumenty. (Microsoft, 2026)

2.4 Adobe Scan

Adobe Scan je bezplatná mobilní aplikace využívající umělou inteligenci, která umožňuje proměnit chytrý telefon v přenosný skener dokumentů. Pomocí aplikace lze snadno digitalizovat dokumenty, účtenky, vizitky nebo například poznámky z tabule. Aplikace automaticky detekuje okraje dokumentu, odstraní odlesky a stíny a pomocí technologie OCR (optické rozpoznávání znaků) převede obraz na upravitelný a prohledávatelný text.

Mezi hlavní funkce aplikace patří automatické vyrovnání a doostření obrazu, převod naskenovaných dokumentů do editovatelných formátů PDF nebo JPEG a ukládání souborů do cloudového úložiště Adobe Document Cloud, odkud jsou dostupné jak na mobilních zařízeních, tak na počítači. Aplikace umožňuje také skenování více stránek do jednoho PDF souboru a nabízí pokročilé nástroje, například přidávání podpisů, vyplňování formulářů nebo komentování dokumentů prostřednictvím Adobe Acrobat Reader. Adobe Scan lze využít například pro digitalizaci účtenek, vizitek, poznámek z výuky, právních dokumentů, smluv nebo tištěných fotografií. (Adobe, 2026)

2.5 ABBYY FineReader

ABBYY FineReader PDF je pokročilý software využívající umělou inteligenci pro optické rozpoznávání znaků (OCR) a práci s dokumenty ve formátu PDF. Slouží především k digitalizaci, úpravě, ochraně a sdílení dokumentů. Program dokáže převádět naskenované dokumenty, obrázky nebo neprohledávatelné PDF soubory do editovatelných formátů, například Word nebo Excel, přičemž dosahuje přesnosti rozpoznání přibližně 95–99 %. Podporuje také více než 100 jazyků a umožňuje uživatelům dokumenty porovnávat, anotovat nebo zpracovávat ve větším množství.

Mezi hlavní funkce programu patří pokročilé OCR založené na neuronových sítích, které dokáže přesně rozpoznat text i v dokumentech se složitým rozvržením. Software umožňuje také úpravu PDF dokumentů, například změnu textu, úpravu obrázků nebo správu jednotlivých stránek. Další užitečnou funkcí je porovnávání dvou verzí dokumentů, které zvýrazní rozdíly mezi nimi. Program rovněž podporuje automatizované zpracování většího množství dokumentů a je často využíván například pro digitalizaci papírových archivů, extrakci dat z tabulek a formulářů nebo kontrolu změn mezi různými verzemi dokumentů. (Abbyy, 2026).

2.6 Další nástroje

Kromě již popsaných řešení existuje také řada dalších nástrojů využívajících technologii optického rozpoznávání znaků (OCR). Tyto nástroje často využívají metody strojového učení a umělé inteligence pro analýzu obrazových dat a převod textu z obrázků nebo dokumentů do digitální podoby. Typickým příkladem je například Google Cloud Vision AI, který nabízí pokročilé funkce pro analýzu obrazu a rozpoznávání textu.

Současný trh nabízí široké spektrum podobných nástrojů určených pro digitalizaci dokumentů, automatickou extrakci textu nebo analýzu obrazových dat. Tato řešení jsou využívána v různých oblastech, například při digitalizaci archivních materiálů, zpracování administrativních dokumentů nebo automatizaci práce s textovými daty. Přestože jednotlivé nástroje mohou používat odlišné technologie a přístupy, jejich společným cílem je převod obrazových informací na strojově čitelný text a zjednodušení další práce s dokumenty.

3 Python

Python je velmi populární vysokoúrovňový programovací jazyk zaměřený na čitelnost kódu a jednoduchost, který umožňuje psát přehledné programy s menším množstvím kódu než například C++ nebo Java (History of Python, 2025). Historie a obecné využití jazyka Python nebudou v této práci dále rozebírány. Pozornost bude věnována přehledu vybraných knihoven v prostředí Pythonu, které jsou klíčové pro realizaci OCR aplikací, a to od samotného rozpoznávání textu přes úpravu a předzpracování obrazu až po tvorbu grafického uživatelského rozhraní. Představeny budou moderní OCR knihovny založené na klasických i hlubokých metodách strojového učení, nástroje pro efektivní zpracování obrazových dat a také GUI frameworky umožňující vytvářet uživatelsky přívětivé aplikace pro různé platformy.

3.1 Knihovny pro převod textu

Jako první se zaměříme na přehled knihoven pro převod textu v prostředí Pythonu, které umožňují automatické rozpoznávání znaků z obrázků, naskenovaných dokumentů a PDF souborů. Představeny jsou jak osvědčené a dlouhodobě používané OCR nástroje, tak moderní knihovny založené na hlubokém učení, které se liší podporou jazyků, přesností, náročností použití i možnostmi přizpůsobení pro specifické aplikační scénáře.

Tesseract OCR, nejpoužívanější OCR knihovna v prostředí Pythonu, původně vyvinutá společností HP a dnes udržovaná Googlem. Jedná se o open-source a bezplatné řešení, které nabízí kvalitní rozpoznávání textu ve více než 100 jazycích včetně nelatinkových písem a dokáže pracovat s obrázky, naskenovanými dokumenty i PDF soubory. Knihovna umožňuje vlastní trénování pro specifické scénáře a dosahuje vysoké přesnosti zejména v kombinaci s nástroji pro předzpracování obrazu, jako je OpenCV. (Saive, 2024)

EasyOCR, OCR knihovna pro Python, která podporuje více než 80 jazyků a je vhodná i pro začátečníky díky jednoduchému a přehlednému API. Je postavena na metodách hlubokého učení, což jí umožňuje dosahovat vysoké přesnosti při rozpoznávání textu, včetně textu ve vícejazyčných či vertikálně orientovaných obrazech. Díky kombinaci snadného použití a pokročilých modelů představuje EasyOCR efektivní řešení pro využití moderních OCR technologií. (Saive, 2024)

Keras-OCR je knihovna pro jazyk Python, která usnadňuje úlohy optického rozpoznávání znaků (OCR) prostřednictvím frameworků Keras a TensorFlow. Nabízí předtrénované modely schopné rozpoznávat text s vysokou přesností napříč různými typy písem a stylů textu. Díky jednoduchému aplikačnímu rozhraní je její implementace do aplikací poměrně snadná. Knihovna zároveň umožňuje flexibilní konfiguraci, například úpravu velikosti vstupního obrazu nebo nastavení cílového jazyka. Jelikož se jedná o open-source projekt, podporuje spolupráci komunity a usnadňuje integraci OCR funkcí do Python aplikací. (Trivedi, 2025)

PaddleOCR, OCR knihovna vyvinutá v rámci deep learning frameworku PaddlePaddle, která podporuje více než 80 jazyků a dosahuje velmi vysoké přesnosti díky využití pokročilých modelů hlubokého učení. Vyniká zejména při zpracování obrazů s komplexním pozadím, a kromě samotného rozpoznávání textu nabízí také detekci textu a analýzu rozložení dokumentu.

Součástí knihovny jsou předtrénované modely pro různé jazyky, což usnadňuje její rychlé nasazení v praxi. (Saive, 2024)

OCRopus, open-source OCR systém vyvinutý společností Google, který je určen především pro zpracování historických dokumentů a knih, ale lze jej využít i pro obecné úlohy extrakce textu. Vyniká zejména v analýze rozložení dokumentu, díky čemuž si dokáže poradit se složitou strukturou stránek. Je navržen modulárně, což umožňuje snadné přizpůsobení a rozšiřování funkcionality, a podporuje zpracování vícestránkových dokumentů i rozsáhlých datových sad, což jej činí vhodným nástrojem pro digitalizační a archivní projekty. (Saive, 2024)

Kraken, open-source systém pro automatické rozpoznávání textu (ATR), který je optimalizován zejména pro historické dokumenty a ne-latinská písma. Byl navržen jako univerzální nástroj pro oblast humanitních věd, kde pomáhá řešit specifické problémy spojené s digitalizací starších dokumentů, rukopisů a méně běžných jazyků nebo písem.

Velkou výhodou Kraken je jeho přizpůsobitelnost a možnost trénování vlastních modelů. Díky tomu je vhodný i pro jazyky nebo typy písem, které nejsou běžně podporovány v komerčních OCR nástrojích. Nástroj nabízí dvě hlavní možnosti použití – rozhraní příkazového řádku pro běžné uživatele, které umožňuje vytvářet automatizované zpracovatelské workflow, a také API pro vývojáře, kteří chtějí Kraken integrovat do vlastních aplikací nebo projektů. (Kraken, 2015)

3.2 Knihovny pro úpravu obrazu

Další část se věnuje knihovnám určeným pro úpravu a předzpracování obrazu v prostředí Pythonu, které hrají klíčovou roli při zvyšování kvality vstupních dat pro následné zpracování. Představeny jsou nástroje pro komplexní počítačové vidění, základní i pokročilé obrazové operace a práci s maticemi obrazových dat, které umožňují efektivní manipulaci, analýzu a přípravu obrazů pro další algoritmické zpracování.

OpenCV, knihovna pro počítačové vidění původně vytvořená společností Intel a dnes udržovaná globální komunitou vývojářů, která patří mezi základní nástroje v oblasti computer vision a strojového učení. Nabízí široké spektrum funkcí pro zpracování obrazu, detekci objektů, rozpoznávání obličejů, rozšířenou realitu či robotiku a díky Python rozhraní umožňuje rychlý vývoj a prototypování. Díky multiplatformní podpoře a snadné integraci s knihovnami jako NumPy, SciPy nebo TensorFlow slouží OpenCV jako univerzální základ pro tvorbu inovativních aplikací v různých oblastech. (Khandelwal, 2025)

PIL (Python Image Library) a její moderní větev **Pillow** představují výkonnou knihovnu pro práci s obrazy v Pythonu, která podporuje širokou škálu formátů, jako jsou JPEG, PNG, TIFF, GIF nebo BMP. Prostřednictvím modulu Image umožňuje provádět základní i pokročilejší úpravy obrázků, například jejich načítání, zobrazování, ukládání, změnu velikosti, ořez, rotaci, převracení či převod do odstínů šedi. Díky jednoduchému rozhraní a bohaté funkční výbavě je Pillow vhodná jak pro rychlé úpravy obrázků, tak i jako součást větších aplikací zaměřených na zpracování obrazu. (Khandelwal, 2025)

NumPy umožňuje provádět základní techniky zpracování obrazu, jako je převracení obrázků, jednoduchá filtrace nebo analýza obrazových dat. Obrázky jsou v NumPy reprezentovány jako vícerozměrná pole (NdArray), přičemž barevný obraz má tři rozměry odpovídající RGB kanálům, které lze snadno oddělit pomocí indexování. Díky jednoduchým operacím, například

vertikálnímu či horizontálnímu převrácení, podmínkovým úpravám pixelů nebo práci s jednotlivými barevnými kanály, je NumPy vhodným nástrojem pro základní manipulaci a předzpracování obrazů, často využívaným ve spojení s dalšími knihovnami pro počítačové vidění. (Khandelwal, 2025)

3.3 Nástroje pro uživatelské rozhraní

Poslední část se zaměřuje na nástroje pro tvorbu grafického uživatelského rozhraní v prostředí Pythonu, které umožňují vytvářet přehledné a uživatelsky přívětivé aplikace pracující s obrazovými i textovými daty. Představeny jsou základní i pokročilé GUI frameworky, které se liší mírou složitosti, rozsahem funkcí a podporovanými platformami, a poskytují tak vhodné možnosti jak pro začátečníky, tak pro pokročilé vývojáře při návrhu desktopových i mobilních aplikací.

Tkinter, nejčastěji používaný GUI toolkit v Pythonu, vytvořený Fredrikem Lundhem jako standardní rozhraní k Tk GUI toolkitu s Python bindingy. Je součástí základní instalace Pythonu na všech hlavních operačních systémech, takže nevyžaduje žádnou dodatečnou instalaci. Tkinter pracuje s grafickými prvky označovanými jako widgety, mezi které patří například tlačítka, rámce, popisky, zaškrtávací políčka, dialogy pro práci se soubory nebo kreslicí plátno. Díky jednoduchému použití, přehlednému API a multiplatformní podpoře (Windows, macOS, Linux) je Tkinter vhodný zejména pro začátečníky a pro tvorbu základních až středně složitých grafických aplikací. (D. Costa, 2024)

PyQt5, multiplatformní knihovna pro tvorbu grafických uživatelských rozhraní v Pythonu, postavená na frameworku Qt5 a vyvíjená společností Riverbank Computing. Nabízí rozsáhlé Python bindingy a podporuje vývoj aplikací pro Windows, macOS, Linux, iOS i Android. Pro tvorbu GUI poskytuje moduly QtGui a QtDesigner, které umožňují jak vizuální návrh pomocí drag-and-drop, tak programovou tvorbu rozhraní, což ji činí vhodnou pro malé i rozsáhlé projekty. Díky více než 35 rozšiřujícím modulům, dobré dokumentaci a široké komunitní podpoře představuje PyQt5 komplexní řešení, které přesahuje samotnou tvorbu GUI (D. Costa, 2024).

Kivy, open-source GUI framework napsaný v kombinaci Pythonu a Cythonu, který je určen především pro tvorbu multitouch aplikací a přirozených uživatelských rozhraní (NUI). Umožňuje vývojářům napsat aplikaci jednou a nasadit ji na více platforem, přičemž vestavěná podpora OpenGL ES 2 dovoluje využívat moderní a výkonné grafické techniky. Kivy je často používán zejména pro aplikace na Androidu a iOS, ale nachází uplatnění i na systémech Linux, Windows, macOS a zařízeních Raspberry Pi, což z něj činí flexibilní řešení pro dotykově orientované aplikace. (D. Costa, 2024)

Gradio, open-source knihovna pro jazyk Python, která umožňuje snadnou tvorbu uživatelsky přívětivých webových rozhraní pro modely strojového učení, API nebo libovolné Python funkce. Původně byla vytvořena jako nástroj pro prezentaci modelů strojového učení širšímu publiku, postupně se však vyvinula v univerzální platformu pro tvorbu interaktivních aplikací v různých oblastech. Díky jednoduchosti použití je oblíbená zejména mezi vývojáři, pedagogy a datovými analytiky, kteří potřebují srozumitelně prezentovat komplexní modely i neodborným uživatelům. (Martinez, 2025)

Streamlit, open-source knihovna pro jazyk Python, která umožňuje rychlou tvorbu interaktivních webových aplikací zaměřených na práci s daty. Je navržena tak, aby zjednodušila proces převodu analytických skriptů do uživatelsky přívětivého webového rozhraní bez nutnosti znalosti HTML, CSS nebo JavaScriptu. Aplikace jsou vytvářeny čistě v Pythonu a umožňují prezentovat výsledky analýz v přehledné a vizuálně atraktivní podobě. Hlavním cílem Streamlitu je propojit analýzu dat s koncovým uživatelem a usnadnit sdílení výsledků v interaktivní formě.

Mezi hlavní vlastnosti Streamlitu patří vestavěné interaktivní prvky, jako jsou tlačítka, posuvníky či výběrové seznamy, které umožňují dynamickou práci s aplikací v reálném čase. Aplikace se automaticky aktualizují při každé změně vstupu, což zajišťuje plynulou uživatelskou zkušenost. Streamlit rovněž podporuje integraci s běžnými knihovnami pro vizualizaci dat (např. Seaborn či Plotly), umožňuje rychlé prototypování a díky své otevřenosti nabízí možnost rozšíření pomocí komunitních doplňků. Hotové aplikace lze snadno nasadit na cloudové platformy, jako je Streamlit Community Cloud, Heroku nebo AWS. (Gurova, 2025)

3.3.1 Doplňkové knihovny

Torchvision, knihovna pro počítačové vidění v jazyce Python, která je součástí projektu PyTorch. Poskytuje předpřipravené nástroje pro práci s obrazovými a video daty v oblasti hlubokého učení. Nabízí moduly pro načítání standardních datasetů, předzpracování a augmentaci obrazu, konstrukci neuronových sítí a další pomocné funkce. Součástí knihovny jsou také předtrénované modely známých architektur, jako jsou ResNet, VGG, MobileNet nebo Faster R-CNN, které lze využít pro klasifikaci, detekci objektů či segmentaci obrazu. Díky modulárnímu návrhu mohou vývojáři využívat pouze vybrané části knihovny podle svých potřeb.

Torchvision je plně integrován s frameworkem PyTorch a využívá jeho tensorové výpočty i akceleraci pomocí GPU. Datasetové třídy umožňují efektivní načítání dat z disku bez nadměrné spotřeby paměti a podporují paralelní zpracování pomocí DataLoaderu. Knihovna spolupracuje s nástroji jako PIL (Pillow) či OpenCV pro práci s obrazovými daty a lze ji snadno kombinovat s dalšími Python knihovnami nebo frameworky, například PyTorch Lightning. Díky optimalizovaným implementacím a podpoře GPU umožňuje vytvářet výkonné a škálovatelné aplikace pro počítačové vidění bez nutnosti implementovat základní komponenty od začátku. (Hynkova, 2025)

Keras, framework určený pro vývoj a výzkum v oblasti hlubokého učení (deep learning). Poskytuje jednoduché a přehledné aplikační rozhraní (API), které usnadňuje vytváření neuronových sítí. Díky tomu mohou vývojáři snadněji navrhovat, trénovat a testovat modely strojového učení bez nutnosti řešit složitou implementaci na nízké úrovni. Keras tak zjednodušuje práci s neuronovými sítěmi a umožňuje rychlejší experimentování s různými architekturami modelů. Keras je možné nasadit na různých platformách a prostředích, například pomocí Pythonu nebo Node.js, a podporuje také nasazení na mobilních systémech, jako jsou iOS a Android. Při vytváření modelů nabízí několik přístupů, například Sequential model pro jednodušší lineární architektury, Functional API pro složitější modely s více vstupy a výstupy a subclassing, který umožňuje vytvářet plně přizpůsobené modely podle specifických požadavků projektu. Díky své jednoduchosti a flexibilitě je Keras jedním z nejpoužívanějších nástrojů pro práci s hlubokým učením. (Coursera, 2025)

4 Návrh aplikace

Tato kapitola se zaměřuje na praktickou implementaci aplikace pro převod ručně psaného textu do digitálně upravitelné podoby. Nejprve jsou popsány jazykové modely využívané knihovnou Kraken, jejich dostupnost a způsob jejich získání z externích repozitářů. Dále je popsán proces výběru vhodného jazykového modelu pro rozpoznávání textu v českém, anglickém a německém jazyce.

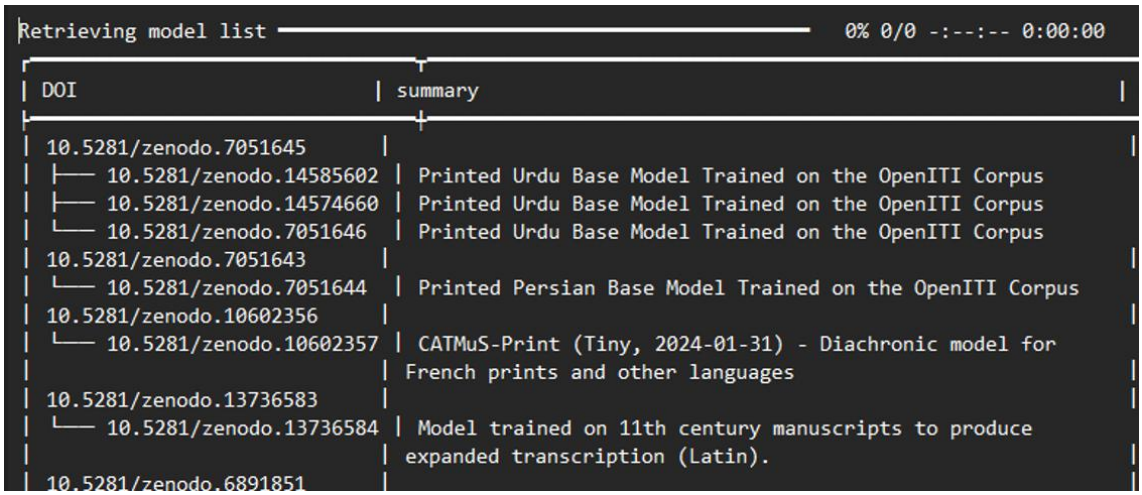
V následujících částech kapitoly je představena struktura vytvořené aplikace a postup jejího spuštění. Popsán je také samotný průběh aplikace od spuštění prostřednictvím příkazového řádku až po zobrazení uživatelského rozhraní ve webovém prohlížeči. Důraz je kladen na jednotlivé kroky zpracování vstupního dokumentu, které zahrnují nahrání obrazového souboru, volbu metody předzpracování, výběr jazykového modelu a kontrolu segmentace textových řádků.

Závěrečná část kapitoly se věnuje podrobnějšímu popisu uživatelského rozhraní a klíčových částí zdrojového kódu aplikace. Jednotlivé podkapitoly vysvětlují funkce využité při zpracování obrazu, segmentaci textu a samotném rozpoznávání pomocí knihovny Kraken. Součástí je také popis generování výsledného textového výstupu a jeho uložení do souboru, aby bylo možné získaný text dále upravovat nebo archivovat.

4.1 Jazykové modely

Knihovna Kraken sama o sobě neobsahuje žádné jazykové modely pro rozpoznávání textu. Tyto modely je nutné stáhnout z externích úložišť a následně je použít při samotném procesu rozpoznávání. Přehled dostupných jazykových modelů lze zobrazit v příkazovém řádků pomocí příkazu **kraken list**, který vypíše seznam modelů dostupných ke stažení. Ukázka výstupu tohoto příkazu je uvedena níže.

Vybraný jazykový model je možné v příkazovém řádku stáhnout pomocí příkazu **kraken get**, který model automaticky stáhne a uloží do lokálního prostředí. Alternativně lze většinu jazykových modelů stáhnout také prostřednictvím webového prohlížeče z externího úložiště **Zenodo.org**.

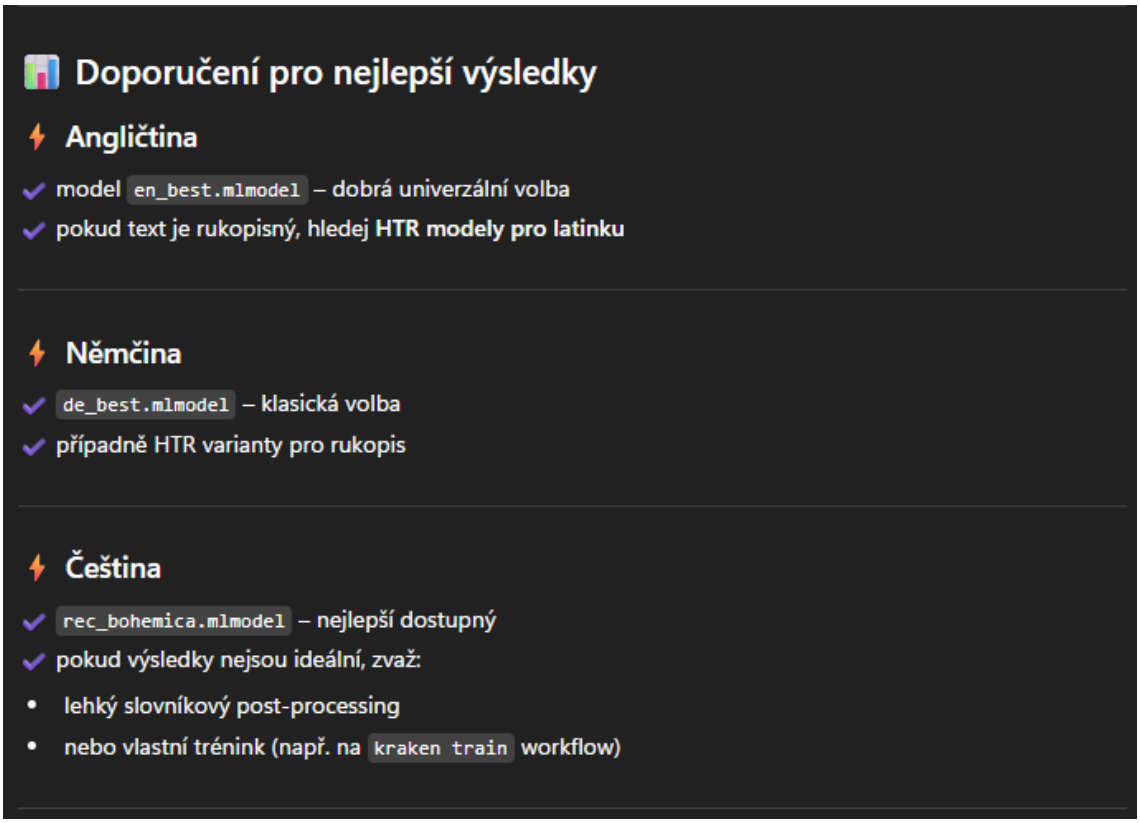


Obr. 1: Výstup příkazu kraken list
Zdroj: Vlastní zpracování

4.1.1 Výběr modelu

Při návrhu aplikace bylo cílem rozpoznání převážně českého textu. Při vyhledávání vhodného jazykového modelu pomocí příkazu **kraken list** však nebyl nalezen model určený přímo pro jeho rozpoznávání. Z tohoto důvodu bylo využito doporučení nástroje ChatGPT, který navrhl použití modelu **rec_bohemica.mlmodel**. Tento model je určen především pro rozpoznávání historických dokumentů pocházejících z českých zemí a je proto vhodný i pro zpracování historických textů v českém jazyce.

Jako alternativní modely byly dále doporučeny **en_best.mlmodel** pro anglický text a **de_best.mlmodel** pro německý text. Modely **rec_bohemica.mlmodel** a **en_best.mlmodel** jsou dostupné na externím úložišti Zenodo. Model **de_best.mlmodel** však nebyl v době zpracování práce dostupný, a proto byl jako náhrada zvolen jiný model určený pro rozpoznávání německého rukopisu, konkrétně **german_handwriting.mlmodel**. Odkazy na externí úložiště obsahující jednotlivé jazykové modely jsou uvedeny v části *Seznam použité literatury*.

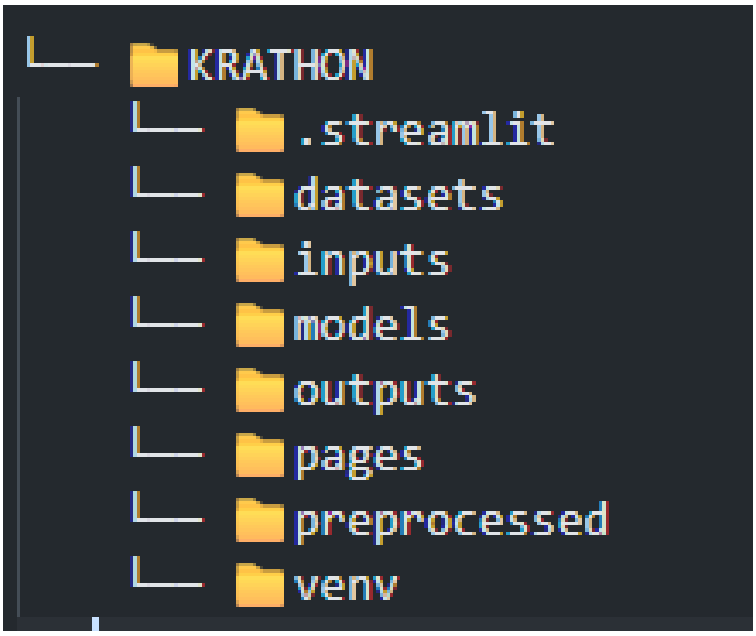


Obr. 2: Výsledek dotazu na jazykový model pro Kraken
Zdroj: OpenAI – ChatGTP 5.2 (2026)

4.2 Struktura a knihovny aplikace

Základní princip aplikace spočívá v tom, že uživatel nejprve vloží vstupní soubor, následně zvolí požadovaný způsob předzpracování obrazu a vybere jazykový model pro rozpoznávání textu. Poté aplikace provede samotné rozpoznání textu a výsledný výstup zobrazí uživateli s možností jeho uložení do textového souboru.

Na základě tohoto konceptu byla navržena struktura programu, která popisuje jednotlivé části aplikace a jejich vzájemné propojení. Základní struktura aplikace je zobrazena níže. Kompletní zobrazení struktury i s umístěním jednotlivých souborů je pak uvedeno v příloze B.1. Hlavní adresář programu obsahuje několik podadresářů určených pro ukládání a načítání souborů a jednotlivých funkcí aplikace. Součástí hlavního adresáře je také soubor **main.py**, pomocí kterého se aplikace spouští. V následující části budou stručně popsány jednotlivé podadresáře programu a jejich funkce v rámci celé aplikace.



Obr. 3: Základní struktura aplikace
Zdroj: Vlastní zpracování

.streamlit – adresář sloužící pro konfiguraci uživatelského rozhraní aplikace. Obsahuje konfigurační soubor **config.toml**, ve kterém jsou definovány parametry vzhledu rozhraní. V rámci této aplikace je zde nastaveno například pozadí a barva fontu.

datasets – v tomto adresáři jsou uloženy všechny soubory potřebné pro trénování jazykového modelu, například připravené obrazové řádky a jejich odpovídající textové přepisy.

inputs – tento adresář slouží k ukládání vstupních souborů aplikace. Přestože aplikace nevyžaduje, aby byly vstupní soubory umístěny právě zde, je tento adresář součástí struktury projektu a může být využit pro organizaci testovacích dokumentů.

models – adresář určený pro ukládání jazykových modelů používaných v aplikaci. V současné verzi aplikace je zde uloženo pět jazykových modelů pro rozpoznávání textu a jeden model určený pro detekci textových řádků.

outputs – adresář sloužící k ukládání výstupních souborů generovaných aplikací. Nachází se zde také soubor **output_processing.py**, který obsahuje funkce pro zpracování, generování a ukládání výsledných výstupů programu.

pages – adresář obsahující jednotlivé stránky uživatelského rozhraní aplikace. Pro správnou funkčnost funkce **page_link**, která je součástí knihovny **Streamlit**, musí být tento adresář umístěn v kořenovém adresáři projektu. Každá stránka uživatelského rozhraní je zde implementována jako samostatný soubor s příponou ***.py**.

preprocessed – tento adresář slouží k ukládání předzpracovaných obrazových souborů. Nachází se zde také soubor **image_processing.py**, který obsahuje funkce pro úpravu obrazu, například převod do odstínů šedi nebo binarizaci.

venv – adresář obsahující virtuální prostředí Pythonu. Jsou v něm uloženy všechny knihovny a závislosti potřebné pro spuštění a správnou funkčnost aplikace.

4.2.1 Knihovny

Pro rozpoznávání a převod textu byla zvolena knihovna Kraken (verze 6.0.3). Tato knihovna byla vybrána především proto, že je vhodná pro práci s historickými dokumenty a ručně psaným textem. Další výhodou knihovny Kraken je možnost trénování vlastních jazykových modelů, což umožňuje přizpůsobit rozpoznávání specifickým typům dokumentů a zvyšovat tak přesnost výsledků. Z důvodu kompatibility s knihovnou Kraken byla pro vývoj aplikace použita verze Python 3.11 namísto aktuální verze Python 3.14.

Knihovna Kraken je postavena na frameworku PyTorch (verze 2.7.1), který slouží jako základ pro práci s neuronovými sítěmi a umožňuje efektivní provádění výpočtů při rozpoznávání textu. PyTorch zároveň podporuje využití grafické karty (GPU), což může výrazně urychlit proces trénování i samotného rozpoznávání.

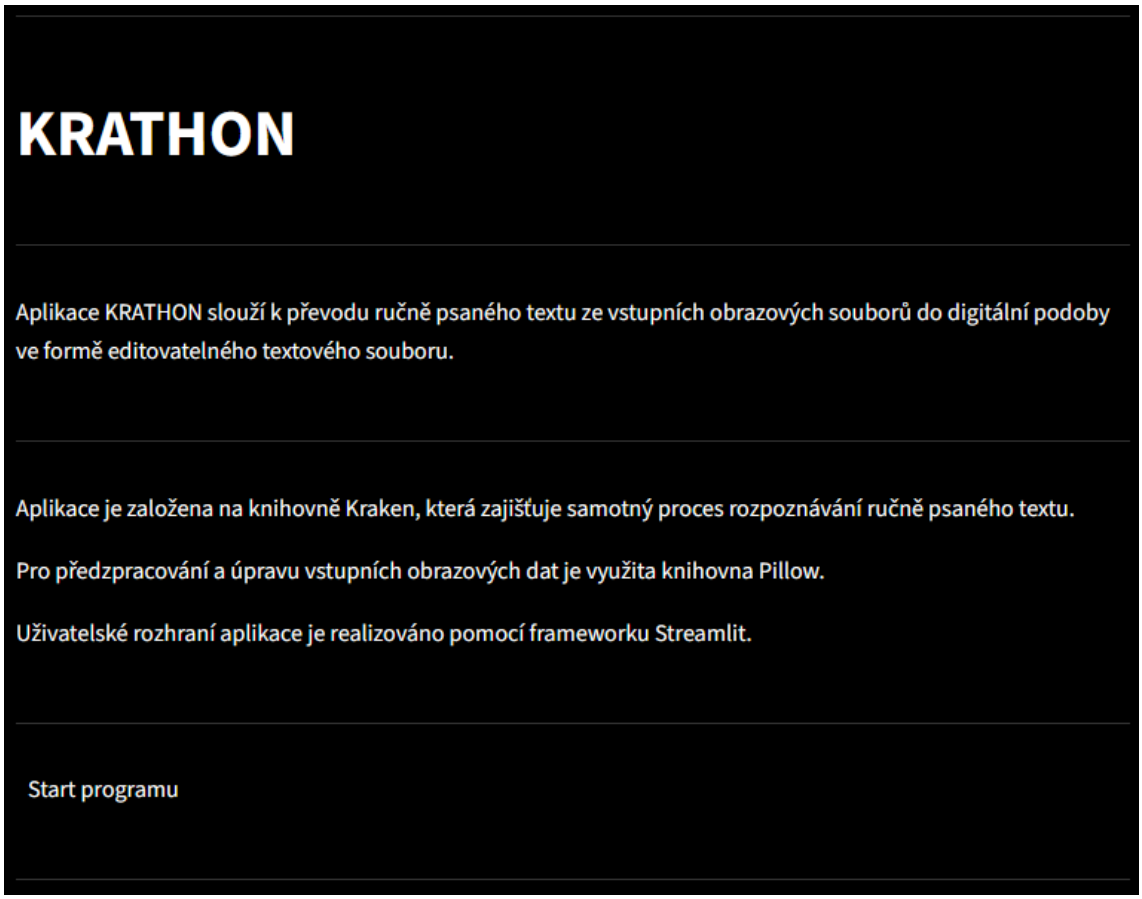
Další použitou knihovnou je Pillow (PIL, verze 12.1.0), která slouží pro práci s obrazovými soubory. Pomocí této knihovny je možné načítat, upravovat a ukládat obrázky, které jsou následně zpracovávány v dalších částech aplikace. Pro vytvoření uživatelského rozhraní byla zvolena knihovna Streamlit (verze 1.54.0). Tato knihovna umožňuje snadnou tvorbu webového rozhraní přímo v jazyce Python bez nutnosti znalosti webových technologií, jako jsou HTML, CSS nebo JavaScript. Streamlit byl zvolen zejména díky své jednoduchosti, přehledné dokumentaci a rychlé možnosti vytvoření funkčního prototypu aplikace.

4.3 Spuštění a průběh aplikace

Spuštění aplikace probíhá prostřednictvím příkazového řádku. Nejprve je nutné přejít do hlavního adresáře aplikace. Následně je třeba aktivovat virtuální prostředí, které obsahuje všechny potřebné knihovny pro správný běh programu. Virtuální prostředí v Pythonu představuje izolované prostředí, ve kterém jsou nainstalovány pouze knihovny potřebné pro daný projekt, což zabraňuje konfliktům s jinými projekty nebo verzemi knihoven v systému. Ve Windows se aktivace virtuálního prostředí provádí spuštěním skriptu **Activate.ps1**, který se nachází v adresáři **Scripts** virtuálního prostředí ***venv***. Skript lze spustit příkazem **.\venv\Scripts\Activate.ps1**.

Po úspěšné aktivaci virtuálního prostředí je možné aplikaci spustit pomocí příkazu **streamlit run .\main.py**. Tímto krokem se zahájí běh aplikace a její uživatelské rozhraní je následně dostupné prostřednictvím webového prohlížeče.

Po otevření aplikace v prohlížeči se uživateli zobrazí uvítací obrazovka programu. Na této obrazovce je uveden název aplikace, její účel a přehled použitých knihoven. Pomocí tlačítka **Start programu** může uživatel zahájit proces převodu textu.



Obr. 4 Uvítací obrazovka aplikace
Zdroj: Vlastní zpracování

Prvním krokem aplikace je nahrání vstupního obrazového souboru určeného k převodu. Aplikace v současné verzi podporuje formáty **PNG** a **JPG**. Následuje volba metody předzpracování obrazu, kde má uživatel na výběr mezi dvěma základními možnostmi pro úpravu obrazu – **Greyscale** nebo **Binarizace**. Po zvolení metody předzpracování je uživatel vyzván k výběru jazykového modelu. Aktuální verze aplikace nabízí výběr z pěti jazykových modelů.

Po výběru jazykového modelu aplikace provede detekci textových řádků ve vstupním dokumentu. Následně je uživateli zobrazeno shrnutí všech provedených kroků – výběr souboru, metoda předzpracování souboru, vybraný jazykový model a připravený nařádkovaný soubor. Po potvrzení zadaných voleb proběhne samotné rozpoznání textu a výsledný přepis je zobrazen v uživatelském rozhraní. Uživatel má poté možnost proces opakovat, případně uložit rozpoznaný text do souboru ve formátu **.txt**. Po uložení výsledku lze aplikaci znovu použít pro zpracování dalšího dokumentu.

4.4 Popis kódu

Tato část práce se zaměřuje na podrobnější popis uživatelského rozhraní aplikace a vysvětlení klíčových částí zdrojového kódu. Jednotlivé podkapitoly postupně představují hlavní obrazovky aplikace a jejich funkčnost, stejně jako principy, na kterých je aplikace postavena. Popis začíná

uvítací obrazovkou a pokračuje jednotlivými kroky zpracování vstupního souboru, od jeho nahrání přes předzpracování obrazu až po samotné rozpoznání textu a uložení výsledného výstupu.

Součástí kapitoly je také vysvětlení použitých knihoven a funkcí, které zajišťují práci s obrazovými daty, segmentaci textových řádků a samotné rozpoznávání textu pomocí knihovny Kraken. Velká pozornost je věnována také způsobu ukládání uživatelských voleb a mezivýsledků pomocí objektu **st.session_state**, který umožňuje uchovávat stav aplikace mezi jednotlivými kroky uživatelského rozhraní.

Závěrečná část kapitoly popisuje proces generování textového výstupu a jeho uložení do samostatného souboru. Uživatel má možnost zkontrolovat jednotlivé kroky zpracování, například správnost segmentace řádků, a následně získat digitální přepis vstupního dokumentu. Tím je popsán kompletní průběh aplikace od načtení vstupního souboru až po vytvoření finálního textového výstupu.

4.4.1 Uvítací obrazovka – main.py

V kódu jsou nejprve importovány moduly **os** a **streamlit**. Modul **os** slouží k interakci s operačním systémem a umožňuje například práci s proměnnými prostředí nebo souborovým systémem. Modul **streamlit** je využit pro tvorbu a správu uživatelského rozhraní aplikace.

Kromě načtení modulů a vytvoření UI pomocí **st.title**, **st.divider** a **st.write** je zde také funkce **init_session().** Funkce **init_session()** slouží k inicializaci globálních proměnných uložených v objektu **st.session_state**, do kterých se v průběhu běhu aplikace ukládají jednotlivé uživatelské volby a mezivýsledky zpracování. V rámci této funkce jsou proměnné inicializovány s výchozí hodnotou **None**, tedy bez přiřazené hodnoty. Tím je zajištěno, že aplikace při svém spuštění pracuje s předem definovaným a konzistentním stavem, který je postupně aktualizován na základě interakcí uživatele. Kompletní kód je k nahlédnutí v příloze B.2.

4.4.2 Nahrání souborů pro rozpoznání – select_file.py

Kromě modulů **io** a **streamlit** jsou zde importovány také nástroje z knihovny **Pillow**, konkrétně třída **Image**. Ta slouží k otevírání a zpracování obrazových souborů a umožňuje načíst nahraný obrázek do paměti aplikace, kde s ním lze dále pracovat, například pro jeho zobrazení v uživatelském rozhraní nebo pro následné zpracování v dalších krocích aplikace.

Kromě samotného kódu zajišťujícího zobrazení uživatelského rozhraní aplikace je využita funkce **file_uploader**, která umožňuje uživateli nahrát obrazový soubor přímo z lokálního disku. Nahraný soubor je dočasně uložen do proměnné image.

Následně jsou vyhodnoceny dvě podmínky. Pokud uživatel soubor úspěšně nahraje, jsou jeho název a binární obsah (hodnota typu *bytes*, tedy surová data souboru) uloženy do globálních proměnných v objektu **st.session_state**. Uložení binárních dat umožňuje s obrazem dále pracovat bez nutnosti opakovaného načítání souboru z disku. Po úspěšném nahrání je uživateli zobrazena informační hláška potvrzující nahrání souboru.

V případě, že je globální proměnná **file_bytes** vyplněna, je nahraný soubor následně otevřen a zobrazen přímo v uživatelském rozhraní aplikace. Tím má uživatel možnost vizuálně ověřit, že byl vybrán správný vstupní soubor.

Pokud uživatel žádný soubor nenahraje, aplikace mu nedovolí pokračovat k dalším krokům zpracování. Tato skutečnost je ošetřena pomocí parametru **disabled** ve funkci **page_link**, konkrétně výrazem **disabled = not st.session_state.get("file_bytes")**. Parametr **disabled** v takovém případě deaktivuje tlačítko pro pokračování, dokud není nahrán platný vstupní soubor. Kompletní kód je k nahlédnutí v příloze B.3.

4.4.3 Volba předzpracování obrazu – select_preprocessing_method.py

Kromě modulů **io** a **streamlit** jsou importovány také funkce **get_greyscale** a **get_binarized** ze souboru image_processing.py, který je umístěn v adresáři **preprocessed**. Tento přístup je zvolen pro zachování přehlednějšího a lépe udržovatelného hlavního zdrojového kódu aplikace.

Dále byly vytvořeny proměnné **GREYSCALE_FILE_PATH** a **BINARIZED_FILE_PATH**, ve kterých jsou uloženy cesty k výstupním souborům pro jednotlivé metody předzpracování. Tyto proměnné slouží jako vstupní parametry pro importované funkce a určují, do jakého umístění budou výsledné obrazové soubory ukládány.

Pro volbu typu předzpracování vstupního obrazu byla použita funkce **radio**, která je svým principem podobná přepínacím tlačítkům známým například z jazyka HTML. Uživatel má možnost vybrat jednu ze dvou dostupných metod předzpracování. Na základě této volby je následně pomocí konstrukce **match** zvolena odpovídající funkce **get_greyscale** nebo **get_binarized**. Výstup z vybrané funkce je poté zobrazen v uživatelském rozhraní a současně uložen do globální proměnné, aby jej bylo možné využít v dalších krocích zpracování. Kompletní kód je k nahlédnutí v příloze B.4.

Jelikož jsou při výběru metody předzpracování využívány funkce ze souboru **image_processing.py** (kompletní kód je uveden v příloze B.9), je vhodné stručně popsat jeho obsah. Kromě standardních importů, které se objevují i v hlavním kódu aplikace, jsou zde importovány také třídy **Image** a **ImageDraw** z knihovny Pillow a funkce **binarization**, **blla** a **vgsl** z knihovny Kraken. Funkce **ImageDraw**, **blla** a **vgsl** budou podrobněji vysvětleny v kapitole 4.4.5 Kontrola řádkování, protože souvisejí s detekcí a vizualizací textových řádků. Funkce **Image** již byla popsána v kapitole 4.4.2, a proto se jí zde dále nevěnujeme.

V této části se zaměříme na funkci **binarization** z knihovny Kraken. Tato funkce slouží k převodu vstupního obrazového souboru do binární podoby, ve které jsou pixely rozděleny na popředí (text) a pozadí. Díky tomuto převodu dochází k potlačení šumu a zvýraznění textu, což je zvláště užitečné u nekvalitních nebo nerovnoměrně osvětlených vstupních obrazů.

Funkce **get_greyscale** a **get_binarized** pracují na obdobném principu. Obě funkce přijímají jako vstup cestu k cílovému adresáři pro uložení výstupního souboru a název výsledného souboru. Samotná obrazová data jsou načítána z globální proměnné **st.session_state.file_bytes**, ve které je uložen binární obsah nahraného vstupního souboru.

Po provedení příslušného převodu je výsledný obraz uložen do předem definovaného adresáře pomocí zadané cesty a názvu souboru. Následně obě funkce vracejí výsledek zpracování prostřednictvím příkazu return.

4.4.4 Volba jazykového modelu – select_model.py

Kromě modulu **streamlit** je zde definována také sada proměnných obsahujících cesty k jednotlivým jazykovým modelům. Tyto proměnné jsou pojmenovány podle příslušných modelů a slouží k jednoznačné identifikaci dostupných možností rozpoznávání.

Uživatel zde, obdobně jako při volbě metody předzpracování, vybírá požadovaný jazykový model prostřednictvím funkce **radio**. Na základě této volby je následně pomocí konstrukce **match** přiřazen název vybraného modelu i odpovídající cesta k souboru do globálních proměnných uložených v objektu **st.session_state**. Kompletní kód je k nahlédnutí v příloze B.5.

4.4.5 Kontrola řádkování – show_lines.py

U kontroly řádkování jsou importovány funkce **draw_lines** a **get_lines** ze souboru image_processing.py (náhled kompletního kódu v příloze B.9).

Výstupy obou funkcí jsou ukládány do globálních proměnných, přičemž funkce draw_lines využívá výstup z funkce get_lines. Výsledný obraz je následně zobrazen uživateli, aby bylo možné vizuálně ověřit, zda byly textové řádky správně rozpoznány a segmentovány.

Následující část se zaměřuje na popis funkcí **get_lines** a **draw_lines**. Funkce **get_lines** využívá moduly **vgsl** a **blla** z knihovny Kraken. Modul **vgs**l slouží k načtení modelu určeného pro detekci řádků, který je následně předán funkci **blla**.

Funkce **blla** následně provádí rozpoznání a segmentaci textových řádků ve vstupním obrazu. Pro svou správnou funkčnost vyžaduje minimálně dva povinné parametry, a to předzpracovaný obrazový soubor a odpovídající BLLA model určený pro rozpoznání řádků.

Dále máme funkci **draw_lines**, která slouží k vizualizaci výsledků segmentace textových řádků. Na základě výstupu funkce **get_lines** načte původní obraz, do kterého pomocí funkce ImageDrav z knihovny Pillow vykreslí červené čáry znázorňující průběh rozpoznaných textových řádků. Výsledný obraz je následně uložen na disk a vrácen jako výstup funkce pro další zobrazení v uživatelském rozhraní. Kompletní kód je k nahlédnutí v příloze B.6.

4.4.6 Shrnutí před samotným rozpoznáním – show_summary.py

Kód zajišťující shrnutí celého procesu je postaven výhradně na funkcích knihovny Streamlit. Prostřednictvím uživatelského rozhraní jsou uživateli přehledně zobrazeny všechny vstupy a volby, které v průběhu práce s aplikací provedl. Veškeré potřebné informace jsou načítány z globálních proměnných uložených v objektu **st.session_state**. Kompletní kód je k nahlédnutí v příloze B.7.

4.4.7 Výstup programu – show_output.py

V kódu zajišťujícím výstupní zpracování souboru jsou importovány funkce **models** z knihovny kraken.lib a funkce **get_content** ze souboru **output_processing**. Modul **models** slouží k načtení zvoleného jazykového modelu používaného pro rozpoznávání textu. Funkce **get_content** následně zajišťuje zpracování vstupních dat a vytvoření výsledného textového výstupu programu.

Funkce **get_content** zajišťuje zpracování vstupních dat a vytvoření výsledného textového výstupu programu. Mezi její vstupní parametry patří zvolený jazykový model, předzpracovaný obrazový soubor a výstup segmentace textových řádků, které jsou následně využity při samotném procesu rozpoznávání textu.

V souboru output_processing.py (náhled kompletního kódu v příloze B.10), je importována funkce **rpred**, která slouží k samotnému rozpoznávání textu. Dále je zde definována proměnná OUTPUT_PATH, která určuje cestu k adresáři, do něhož je ukládán výsledný výstupní soubor.

Funkce **get_content** pak slouží k provedení samotného rozpoznávání textu pomocí knihovny Kraken. Jako vstup přijímá zvolený jazykový model, předzpracovaný obraz a výstup segmentace textových řádků. Tyto vstupy jsou předány funkci **rpred**, která na jejich základě rozpozná text v jednotlivých řádcích. Použití parametrů **pad** a **bidi_reordering** napomáhá zlepšení kvality rozpoznávání a zajišťuje správné pořadí znaků ve výsledném textu.

Výstupy rozpoznávání jsou postupně ukládány do seznamu, přičemž jsou zpracovány pouze řádky obsahující platnou predikci. Po dokončení rozpoznávání jsou jednotlivé řádky spojeny do jednoho textového řetězce odděleného znaky nového řádku. Funkce následně vrací výsledný textový výstup, který představuje digitální přepis vstupního obrazového souboru. Textový výstup je následně zobrazen v uživatelském rozhraní. Kompletní kód je k nahlédnutí v příloze B.8.

4.4.8 Uložení výstupu – save_file.py

Pro uložení výsledného souboru je využita funkce **save_content**, která se nachází v souboru output_processing.py. Tato funkce zajišťuje uložení výstupu funkce **get_content** do textového souboru, čímž umožňuje uživateli získat rozpoznaný text ve formě samostatného a dále editovatelného souboru.

Kromě funkce **save_content** je v daném souboru definována také funkce **reset_session**. Ta umožňuje uživateli v posledním kroku aplikace pokračovat v rozpoznávání dalšího souboru. Funkce **reset_session** zajišťuje vymazání globálních proměnných uložených v objektu **st.session_state** a jejich opětovnou inicializaci s hodnotou **None**, čímž je aplikace připravena na nový průchod zpracování.

Na začátku funkce **save_content** je z globální proměnné **st.session_state.file_name** získán název původního vstupního souboru. Tento název je následně upraven pomocí metody **rsplit**, která oddělí příponu souboru od jeho názvu. Díky tomu je možné zachovat původní název souboru a pouze změnit jeho příponu na **.txt**.

Následně je otevřen nový textový soubor v adresáři definovaném proměnnou **OUTPUT_PATH**. Do takto vytvořeného souboru je poté zapsán textový obsah získaný z funkce **get_content**. Kompletní kód je k nahlédnutí v příloze B.11.

5 Vývoj a testování aplikace

Tato kapitola se zaměřuje na praktické testování vyvinuté aplikace a na vyhodnocení její schopnosti rozpoznávat text z obrazových dokumentů. Nejprve je popsán průběh prvních testů programu, které probíhaly ještě před implementací grafického uživatelského rozhraní, pouze v prostředí příkazového řádku. Tento přístup umožnil postupně ověřit správnou funkčnost jednotlivých částí aplikace, zejména zpracování obrazu, segmentaci textových řádků a samotné rozpoznávání textu pomocí jazykového modelu. Součástí testování bylo také vyhodnocení přesnosti rozpoznávání pomocí metrik CER (Character Error Rate) a WER (Word Error Rate).

Další část kapitoly je věnována trénování jazykového modelu a jeho postupnému zlepšování pomocí připravených trénovacích dat. Je zde popsána příprava datasetu, průběh jednotlivých tréninkových cyklů i výsledky validačních testů. Výstupy modelu jsou následně porovnávány s referenčním dokumentem a také s výsledky nástroje Transkribus pro rozpoznávání dokumentů. Kapitola se dále zaměřuje na vliv kvality vstupních dokumentů na přesnost rozpoznávání, kdy jsou testovány různé typy degradace obrazu, například natočení, rozmazání nebo simulace starého papíru. Cílem této kapitoly je ověřit funkčnost aplikace v praxi a zhodnotit faktory, které mají zásadní vliv na kvalitu výsledného převodu textu.

5.1 První testování programu

Pro účely testování funkčnosti jednotlivých částí programu byl na začátku vývoje používán program bez uživatelského rozhraní, tedy pouze v prostředí příkazového řádku. Tento přístup umožnil jednodušší ladění jednotlivých funkcí a rychlejší ověřování jejich správné funkčnosti.

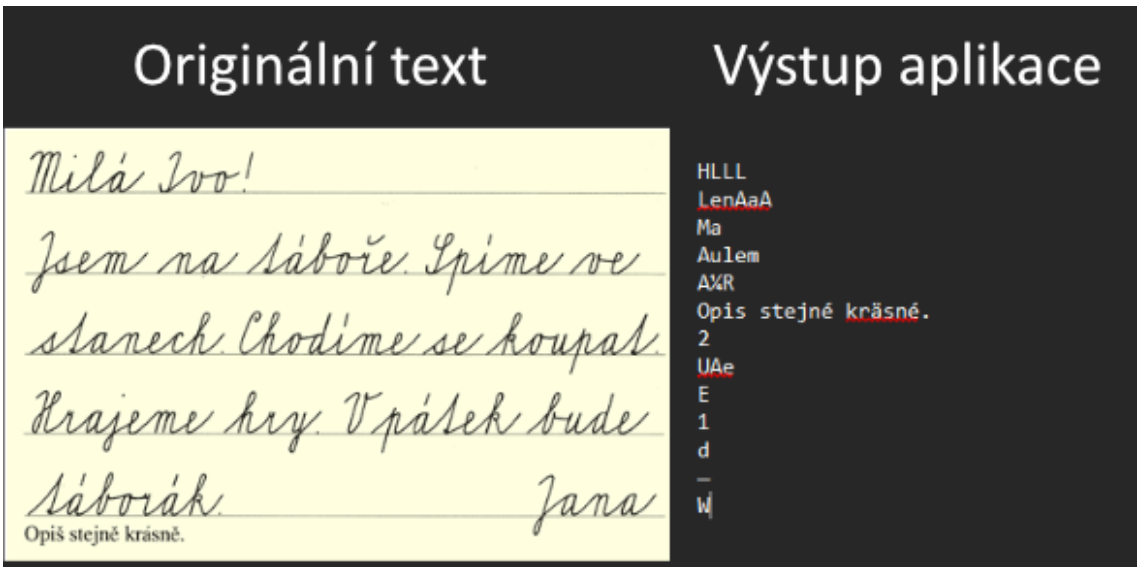
V této fázi byly všechny vstupní parametry, jako například cesty k obrazovým souborům, jazykové modely a umístění výstupního souboru, definovány přímo v proměnných programu. Tyto proměnné byly následně předávány jednotlivým funkcím, které zajišťovaly zpracování obrazu, rozpoznání textu a uložení výsledného výstupu. Tento způsob práce umožnil postupně testovat a ověřovat jednotlivé části programu ještě před implementací uživatelského rozhraní.

```
#Input path
INPUT_FOLDER = '.\\input\\'

#Test files
TEST_IMG = '.\\input\\file_01.jpg'
GREYSCALE_TEST_IMG = '.\\preprocess\\greyscale\\file_01.jpg'
BINARIZED_TEST_IMG = '.\\preprocess\\binarized\\file_01.jpg'

#Language model paths
EN_BEST_MODEL = '.\\models\\en_best.mlmodel'
GERMAN_HANDWRITING_MODEL = '.\\models\\german_handwriting.mlmodel'
REC_BOHEMICA_ORIGINAL = '.\\models\\rec_bohemica.mlmodel'

#Lines recognition model path
BLLA_MODEL_PATH = '.\\models\\lines_recognition\\blla.mlmodel'

#Output path
OUTPUT_PATH = '.\\output\\'
```

Obr. 5: Ukázka nastavení testovacích proměnných

Zdroj: Vlastní zpracování

Jakmile byla ověřena funkčnost jednotlivých funkcí programu, byl proveden první test převodu vstupního dokumentu. V této fázi však byly výsledky rozpoznávání velmi nepřesné. Rozpoznaný text se ve značné míře lišil od původního vstupu a mnoho znaků bylo identifikováno nesprávně nebo nebylo rozpoznáno vůbec.



Obr. 6: Výstup prvního testování aplikace

Zdroj: Normalizované školní písmo, 2007

Po neúspěšném prvním převodu byl program vyzkoušen i na dalších dokumentech. Výsledky však byly podobné jako u prvního testovaného souboru a přesnost rozpoznávání zůstala

nedostatečná. Z tohoto důvodu přišlo na řadu trénování jazykového modelu s cílem zlepšit kvalitu rozpoznávání textu.

Před zahájením trénování byl vybrán referenční vstupní dokument, na kterém budou postupně porovnávány výsledky jednotlivých fází tréninku jazykového modelu. Jako referenční dokument byla zvolena 10. strana českokrumlovské kroniky, která je uvedena v příloze práce (Příloha A.1). Tato stránka byla zároveň ručně přepsána do textového souboru, aby bylo možné provést přesné vyhodnocení výsledků rozpoznávání.

Pro výpočet přesnosti byl využity online nástroje **Character Error Rate (CER) Calculator a Word Error Rate Calculator**, které umožňují porovnat rozpoznaný text s referenčním přepisem a vypočítat hodnoty **CER** a **WER**. Odkaz na webové stránky obou nástrojů je možné nalézt v sekci *Seznam použité literatury*. Podle výsledků tohoto nástroje dosahoval jazykový model **rec_bohemica** před zahájením trénování hodnot **CER 58,07 %** a **WER 91,43 %**. Níže je uveden výstup jazykového modelu před zahájením trénování.

```
10
31. Fectra apttahit aeter sras satitidy aidedty stuem
Muatndes saus norig atrrtendts Kowortdistn v Öhodo
vudw v poa 1983
Doon ränomnto Zorptaisten v dos 1983 ic darateriro
den prordenn serdz efung, gaernn Ktorehe
tenn a slatui prshtarn 7. 18. zen 1953 0 zde
Eurotine a hathrus fron raites Gidr
. Rntn sinu Prängelose vestg. pocn zlau Ernka ero¬
v §2 spte va 39 Procub
v. Zust prängdo wrste. kumte eesta v sea 1953 i
Noman v poen 1952 oipta o 10 pront„ na Voru
prorlas v Ling Obdsk mta ersta v Mco.
Dopsila  rsta unoa aeothu suetezitzes sio salr
ozoen ranodnito horgs bäřa. dahow pařuuplorn dn
mobka & nta a nonut v j sortinent zindna
e Fetraunannde mesttu a puugorito zz
r. Davedear vom tedndrg. da ranedera nen. pfra
enes usta p aus some namräbrngee deszu a znd
Ieao
v. Lanetel da„ poo znutetidten nagotrauvat dako¬
s Feuren iche oatnic otorten narodmle Aotpo¬
dara d Faturus zagen zenedäthd eester dta pro.
betena Pata srahen Pahauisaa potuteh pran o ze.
nu Palde v nila) d Ba zaiiina burotnd zain
Eratoanok Seu d2. 3 a ödustlias borshautrich zuuidel
au dede vhung osteent uug usttergd wide liitehnct
Hartdstiges vrottu votistuatg sinr rieur a Eeommen
1o51
```

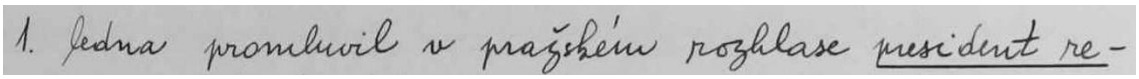
Obr. 7: Výstup aplikace před začátkem trénování

Zdroj: Vlastní zpracování

5.2 Trénink jazykového modelu

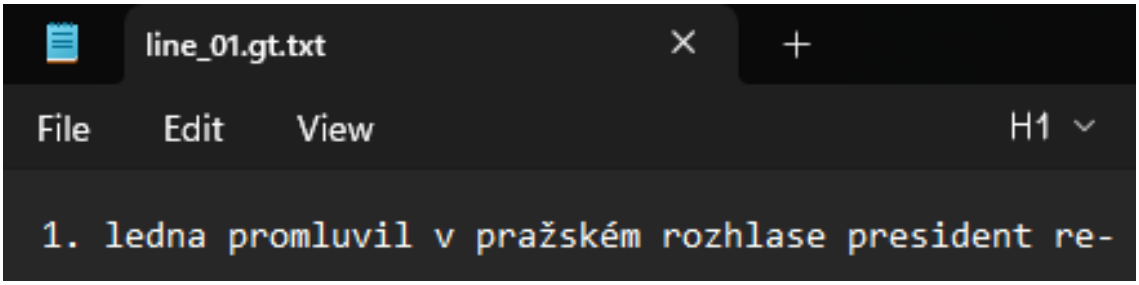
Před samotným trénováním jazykového modelu bylo nejprve nutné připravit trénovací data. Jako vstupní materiál pro trénink bylo zvoleno prvních 23 stran českokrumlovské kroniky z roku 1954. Tyto stránky byly nejprve pomocí jednoduchého programu napsaného v jazyce Python převedeny do odstínů šedi (**Greyscale**), což umožňuje efektivnější zpracování obrazových dat v dalších krocích.

Následně bylo pomocí stejného programu provedeno grafické rozdělení textu na jednotlivé řádky. Toto řádkování slouží jako referenční podklad pro další zpracování a zároveň umožňuje vizuálně ověřit, zda program jednotlivé řádky správně detekuje. Na základě takto připravených podkladů byly jednotlivé stránky rozděleny na samostatné textové řádky, které následně sloužily jako vstup pro trénování modelu. Pro první fázi tréninku bylo tímto způsobem připraveno celkem 200 textových řádků, které byly následně rozděleny v poměru 80% pro trénink, 10% procent pro validaci a 10% pro testování. Tréninkové a validační řádky se využívají při trénování jazykového modelu. Testovací řádky se následně použijí pro vyhodnocení tréninkového cyklu. Příklad takto připraveného řádku je uveden níže.



Obr. 8: Ukázka připraveného řádku pro trénink
Zdroj: Vlastní zpracování

Po rozdělení přišlo na řadu přepsání jednotlivých řádků do textových souborů. Tyto soubory budou sloužit pro kontrolu při trénování.



Obr. 9: Ukázka "Ground truth" souboru pro validaci
Zdroj: Vlastní zpracování

5.2.1 HW konfigurace

Před samotným popisem jednotlivých tréninkových cyklů je vhodné uvést hardwarovou konfiguraci zařízení, na kterém byly tyto cykly prováděny. Stejně jako vývoj aplikace byl i trénink jazykového modelu realizován na operačním systému Windows 11.

Výpočetní výkon systému zajišťuje osmijádrový procesor AMD Ryzen 7 7800X3D, doplněný o operační paměť DDR5 o celkové kapacitě 64 GB. Pro grafické výpočty je k dispozici grafická karta NVIDIA GeForce RTX 3080, která může být využita také pro akceleraci výpočtů při trénování modelu. Uložení dat zajišťuje SSD disk Kingston s kapacitou 2 TB, připojený prostřednictvím rozhraní M.2, které umožňuje vysokorychlostní čtení a zápis dat.

5.2.2 První trénink a test modelu na větším dokumentu

Po dokončení přípravy trénovacích dat bylo možné přistoupit k samotnému trénování jazykového modelu. Trénování bylo nastaveno na 30 trénovacích cyklů, po jejichž dokončení byl proces ukončen. Jeden trénovací cyklus v tomto případě trval přibližně pět minut.

Po dokončení trénování byl následně proveden test rozpoznávání na připravených testovacích souborech. Tento krok slouží k ověření, jak přesně model dokáže rozpoznat text na datech, která nebyla použita přímo při trénování. Výstup z tohoto testovacího cyklu je uveden níže.

```
=== report  ===

998     Characters
139     Errors
86.07%  Character Accuracy
86.07%  Character Accuracy (Case-insensitive)
47.71%  Word Accuracy

15      Insertions
16      Deletions
108     Substitutions

Count   Missed  %Right
183     15      91.80%  Common
815     108     86.75%  Latin
```

Obr. 10: Validační report po prvním tréninkovém cyklu
Zdroj: Vlastní zpracování

První validační report byl vyhodnocen na datasetu obsahujícím celkem 998 znaků. Z tohoto počtu bylo zaznamenáno 139 chyb, což odpovídá přesnosti rozpoznání znaků přibližně 86 %. Přesnost rozpoznání při ignorování velikosti písmen byla prakticky shodná, což naznačuje, že většina chyb nebyla způsobena záměnou velkých a malých písmen. Přesnost na úrovni celých slov dosáhla hodnoty 47,71 %, což je výrazně nižší než přesnost na úrovni znaků. Tento rozdíl je způsoben tím, že i jediná chyba ve slově způsobí, že je celé slovo považováno za nesprávně rozpoznané.

Detailnější analýza chyb ukazuje, že nejčastějším typem chyby byly záměny znaků (substitutions), kterých bylo zaznamenáno 108. Kromě toho model vytvořil 15 vložených znaků (insertions) a 16 vynechaných znaků (deletions). Při rozdělení znaků podle kategorií dosahovala přesnost u běžných znaků přibližně 91,8 % (mezery, interpunkce, číslice), zatímco u znaků latinské abecedy (písmena i s diakritikou) byla přesnost kolem 86,75 %. Tyto výsledky naznačují, že model byl v této fázi tréninku schopen rozpoznávat většinu znaků správně, avšak stále se objevovalo větší množství chyb zejména u písmen.

5.2.3 Druhý trénink a validace výstupu

Protože model po prvním trénování zatím nedosahoval dostatečné přesnosti, bylo nutné pokračovat v dalším trénování s rozšířenou sadou vstupních dat. Trénovací dataset byl proto doplněn o dalších 300 textových řádků. Cílem tohoto kroku bylo poskytnout modelu větší množství trénovacích dat, což může přispět ke zlepšení kvality rozpoznávání.

Druhý trénovací proces byl časově náročnější než první. Přestože byla snaha využít pro trénování grafickou kartu, nedošlo ke správnému nastavení prostředí v příkazovém řádku. Celý proces tak trval přibližně 12 hodin a během této doby proběhlo celkem 42 trénovacích cyklů. Po dokončení trénování byl opět proveden testovací cyklus, jehož cílem bylo ověřit, zda došlo ke zlepšení přesnosti rozpoznávání. Výsledky tohoto testování jsou uvedeny níže.

```
=== report  ===

2380    Characters
182     Errors
92.35%  Character Accuracy
92.39%  Character Accuracy (Case-insensitive)
64.43%  Word Accuracy

48      Insertions
29      Deletions
105     Substitutions

Count   Missed  %Right
1873    113     93.97%  Latin
507     40      92.11%  Common
```

Obr. 11: Validační report po druhém tréninkovém cyklu

Zdroj: Vlastní zpracování

Druhý validační report byl proveden na větším souboru dat obsahujícím 2380 znaků. Celkem bylo zaznamenáno 182 chyb, což odpovídá přesnosti rozpoznání znaků přibližně 92 %. Přesnost při ignorování velikosti písmen dosahovala téměř stejné hodnoty (92,39 %), což opět potvrzuje, že rozdíly mezi velkými a malými písmeny neměly významný vliv na výsledky rozpoznávání. Přesnost rozpoznání celých slov se zvýšila na 64,43 %, což představuje výrazné zlepšení oproti prvnímu validačnímu reportu.

Z hlediska typů chyb byly opět nejčastější záměny znaků, kterých bylo zaznamenáno 105. Dále se objevilo 48 vložených znaků a 29 vynechaných znaků. Při rozdělení podle kategorií znaků dosáhla přesnost u latinské abecedy přibližně 93,97 % a u běžných znaků přibližně 92,11 %.

Celkové výsledky tak ukazují zlepšení přesnosti rozpoznávání oproti prvnímu reportu, což naznačuje, že další trénování modelu vedlo k lepšímu přizpůsobení modelu zpracovávaným datům a ke snížení celkové chybovosti.

5.2.4 Porovnání modelů po prvním a druhém tréninkovém cyklu

Výsledky testování po tréninku jazykových modelů byly již částečně představeny v předchozích kapitolách 5.2.1 a 5.2.2. V této části se zaměříme na konkrétní výstupy jazykových modelů při zpracování vybraného dokumentu a na jejich vzájemné porovnání. Stejně jako před zahájením trénování bude pro toto srovnání použita 10. strana českokrumlovské kroniky.

Pro předzpracování vstupního obrazu byla zvolena metoda **Greyscale**. Nejprve bude představen výstup modelu po prvním tréninkovém cyklu, který následně poslouží jako základ pro porovnání s výsledky dosaženými po dalším trénování modelu.

```
19
31. ledtna svyttázl ztáítru íářacd ztadtástřchý seýstacdtty zluena
státrnátv ptáru rozavje udrvdtního krorportářztvr v teskroslo-
pekerskru ve (roce 1953
Iozoý rárovtrní to horsv dotřrtu ve roca 1953 je cherratktaníro-
oáí provevdtenírmu perně žeu za tory, zrušerníu lístk ovve le.
pryzstá raru a vobáscdrrím prrobtášen z 15., zaa 1953 0 zoýde,
Vorrotrne u kraltenu fírooně arašěta Přotr-
V. PVrčenna ptárnu porcěrnaystove zrobry: roem plau hautsá zýpo-
Prr teak zpbnču na 99 Veroecut
II. V zzůst pračray slovne zaýroby. hautá zroba ze roca 1953 ne-
Irgroracákrh  prokerau 1952. ztorapta o 15 taroceret, naa Vloneasku
vozprostto, ve tánaje brdobří hautá ztrota  1400.
Vsoýpíta sa vvrobra urnoha vyrobků ddvělejžítých pro sdatž
12yjoz j srapro notrríko trorpo nehářě sto. takozt pačuuelo ch
výrobtzu se zlepnta a rozšnřít za její porti nrreato zejia ra.
sa (potreaváínářstrýete zojrotkůů a (ernůrraystonato zbrože,
VI. Zazvásdtera revrna tacetuíey.:  byta zanedena reze. přříhra-
era rýrota u náz dobruch rrevyrátčnacých ztrojů a zará -
zerau,
17. Zesučdčě lství : rozovj zemčdětsto repoterač orvat dako-
sprzru tenapru jatv vztoatních vdučtví veárodrni tv Čarospo-
sháížeykoví. K datříau zožšení zeručdětske žrobyy byha pro-
pestena (šacha o rabren Sretanizaca potudehu prac v ze-
prčs dtáškstvi ze zoýsvsta , abr boa zvžšena trrurotna zain-
tenasovanosk člevnně J2 1 a jeodrrottivč tavoryv hařících zeruřtot-
dě, Prdtry zvoý rennay svýkrapend rerry něhbených vvejotvč ležítějšícho
zerčoděť statjetu vZýrobsků pobssyturuty úsrro stáčetu a provvozcur
1354
```

Obr. 12: Výstup aplikace po prvním tréninkovém cyklu

Zdroj: Vlastní zpracování

Výstup aplikace byl, stejně jako u původního jazykového modelu, vyhodnocen pomocí nástrojů Character Error Rate (CER) Calculator **a** Word Error Rate Calculator.

Po prvním tréninkovém cyklu dosáhl model hodnot **CER 48,27 %** a **WER 89,80 %**. Tyto hodnoty naznačují, že model v této fázi tréninku stále vykazuje poměrně vysokou chybovost, zejména na úrovni celých slov. Výsledky proto ukazují, že pro dosažení vyšší přesnosti rozpoznávání je nutné model dále trénovat a rozšířit množství trénovacích dat.

```
1
11., fadaa vyttázůt ztsítené úířad zdastáztíck arjzbootly ztren
pkátrníto pkáu rozavoje edrvadaního kosspodářztaí v leskroslo -
Netarzken ae 720300 k2732
Ivrozý árvntast to foos totšptov  raca 175 Z je cerca ttas soo-
ín paaravdenní, perů zara ra farey, zhreeštreíu fíztt vre l
pzztá rarca a vlo ctariar, prasotlá žěnín, z 65. zarr 1953   zary se
Karevytrré 2 kalterarí tízaroč rro šů k  fiska,
7. ČLarženvá pttárnca fretrnayetoe jnzby : rčr pllám karuké výr o
Pr ta k szpkruč u na) 37 paozcevet,
I. Vzrlzt pračay zloné ýrat , kuta noíryla s roca 195 z ae
Izkoratkkh2 os prokern 693 ) ztvreflla s 12 forgeecef : pe e Šboecezka
vzprgztfo, e tánežž pbrdzb hurtrá trata a 140)
Iv pita pa v)roka manoka vrzstů daůležítjý k puo dat
rsyz z ; rad  notrad ko trospoadářztv , takoo zk kauť raeejeskva k
ro ropkc j zlepjní ty  rozšařít ze jeji pozti arrerato zejrená see a
na (atrahinkztrejet rrznobků a fračarecystorná (to zk vže,
14, ) apoá okerna gree faekkení ky : byta zaneddean praze ,: re2 prcs ,
spěne rszka a aáez dprudt  reháněaých ztrpjač a zaní -
změna
77. Za rědě kství : prosogj zaažtátytut nepatraš zrya k dta ko -
oyava, teraspera zatio vyztatarok pdačdí nadazntast lv lesoppo :
dožgtaví, t dalžímu zíšení zenačětsk protu br la pro-
pastanaa pizasa atres , Meclaaí za ca patrasteh pra , í zp ze -
raž oů kztai ze zazjpita , atr tyla zayýžena fraeastuat z asin-
tarasovauazto člesaať )Z)) a jestarsttiní trovzy doaísevt t zrašt k
edě baztos zarj peray prýkrapraí cerae ěkterayct raejotač kez ité (řích
zeraldkkstazlbu V2ýkobků pobugtruraky arnrt stála a fro40zac2
(3 1.
```

Obr. 13: Výstup aplikace po druhém tréninkovém cyklu

Zdroj: Vlastní zpracování

Výstup aplikace po druhém tréninkovém cyklu byl opět porovnán s originálním referenčním souborem. U tohoto modelu byla naměřena hodnota **CER 63,38 %** a hodnota **WER 94,63 %**. Jak je možné z výsledků pozorovat, hodnoty přesnosti jsou v tomto případě horší než u prvního jazykového modelu.

Je však třeba vzít v úvahu, že výstup aplikace nemusí být při každém spuštění zcela totožný, a proto se mohou hodnoty přesnosti při opakovaném rozpoznávání mírně lišit. Z dosažených výsledků nicméně vyplývá, že pro dosažení lepší přesnosti rozpoznávání bude nutné provést další trénink modelu s větším množstvím trénovacích dat.

5.2.5 Porovnání výstupu aplikace s online nástrojem Transkribus

Ačkoli jazykový model použitý v aplikaci zatím není dostatečně natrénován, je vhodné jeho výsledky porovnat s již existujícím řešením. Pro toto srovnání byl využit nástroj **Transkribus**, který je běžně používán pro rozpoznávání historických dokumentů. Výstup z tohoto nástroje je uveden níže.

```
10
31. ledna vyhlásil státní úrad statistický výsledky plněm
stáchního plánu rozvoje národního hospodářství v Áskoslo-
vensku u roce 1953.
Rozvoj národníko hospodáiství v roce 1953 je charakderiso-
ván provedením peněžní reformy, zrušením lístkového
systému a vládnum prohlášennu z 15. zán' 1953 o zvýšém
hmotné a kulturní úrovně našeho lidu
I. Plněmí plánu průmyslové výroby: roční plán hmbé víro
by byl spluen na 99 procent.
II. Vzrůst průmyslové výroby: hmbé víroba v isce 1953 ne
sroonání s rokem 1952 stoupla o 10 procent; na Hovensku
vzrostla v témy obdobi hmbé nýroba o 14%
Zvýšila se výroba mnoha vjrobků důležilijch pro ďalši
rozvoj narodního hospodařství. kkost puěmyslových
výrobků se zlepšila a rozšířil se její sortiment zejména
u fotravinážskéjch výrobků a průmyslového zboží
III. Zavádem nové techniky: byla zanedena resp. připia-
vena výroba u nés dosud nevyráběných stroju a zaří-
zeni.
IV. Zemědělskví: rozvoj zemeďelshví nepokračoval dako-
vým tempem jako oztatních odvečwi národního hospo-
dážství. K dalžímu zvýšení zemědělské výroby byla pro-
vedena rada opacřem. Mechanisace polních precí v ze-
mědělství se zvýšila; aby byla zvyšěna huotná zain-
teresovanost člemů SZD a jednotlivě hospodańcích zeměděl
ců, byly zvýšeny výkupní ceny ňekterých nejdůleži těích
zemedělských výrobků poskytnuty investicuí a provozní
1954

Text Recognition powered by transkribus.ai
```

Obr. 14: Výstup nástroje Transkribus
Zdroj: Vlastní zpracování

Z výstupu nástroje Transkribus je patrné, že přesnost rozpoznání vstupního dokumentu je výrazně vyšší než u vytvořené aplikace. V tomto případě dosahuje hodnota **CER 8,37 %** a hodnota **WER 31,47 %**, což představuje podstatně nižší chybovost při rozpoznávání textu.

Pro srovnání, před zahájením trénování jazykového modelu v rámci vytvořené aplikace dosahoval model **rec_bohemica** hodnot **CER 58,07 %** a **WER 91,43 %**. Po prvním tréninkovém cyklu došlo ke zlepšení, kdy hodnoty klesly na **CER 48,27 %** a **WER 89,80 %**. Po druhém tréninkovém cyklu byly naměřeny hodnoty **CER 63,38 %** a **WER 94,63 %**, což naznačuje, že model v této fázi tréninku nebyl ještě dostatečně stabilní a výsledky se mohou lišit v závislosti na kvalitě vstupních dat a dalších faktorech.

Ve srovnání s nástrojem Transkribus je tedy patrné, že komerčně vyvíjené řešení dosahuje výrazně vyšší přesnosti rozpoznávání. Tento rozdíl je pravděpodobně způsoben zejména rozsáhlými trénovacími daty a dlouhodobým vývojem modelů, které jsou v těchto nástrojích používány. Výsledky experimentu zároveň ukazují, že pro dosažení podobné úrovně přesnosti by bylo nutné model trénovat na výrazně větším množství dat a provést další optimalizaci procesu rozpoznávání.

5.3 Testování aplikace s nekvalitními vstupními soubory

V kapitole 1.4 teoretické části byl popsán vliv kvality vstupního dokumentu na přesnost rozpoznávání textu. Za účelem ověření tohoto vlivu byly vstupní obrazové soubory upraveny pomocí grafického programu **Affinity** tak, aby simulovaly různé typy nekvalitních vstupů do aplikace.

Takto upravené soubory byly následně vloženy do aplikace a byl proveden proces rozpoznávání textu. Výsledky jednotlivých testů a jejich vliv na přesnost rozpoznávání jsou podrobněji popsány v následující části.

Pro první test přesnosti byl vstupní dokument natočen o 10 stupňů (Příloha A.2). Takto upravený dokument byl následně vložen do aplikace. Detekce textových řádků proběhla bez problémů, avšak komplikace nastaly při samotném rozpoznávání textu. Již při vizuálním zhodnocení výsledku bylo patrné, že přesnost rozpoznání je velmi nízká. Výsledné hodnoty dosáhly **CER 80,24 %** a **WER 95,12 %**, což potvrzuje výrazný negativní vliv natočení dokumentu na kvalitu rozpoznávání.

V dalším testu byl vstupní soubor upraven tak, aby simuloval vzhled staršího papíru (Příloha A.3). Upravený dokument byl opět vložen do aplikace. Detekce řádků proběhla správně, avšak podobně jako v předchozím případě se problémy objevily při samotném rozpoznávání textu. V tomto případě byly naměřeny hodnoty **CER 69,61 %** a **WER 93,73 %**.

Pro třetí test byl vstupní dokument rozmazán (Příloha A.4). Dokument prošel všemi kroky aplikace a rozpoznání textových řádků proběhlo bez obtíží. Přesnost samotného rozpoznání však byla i v tomto případě velmi nízká. Naměřené hodnoty dosáhly **CER 77,30 %** a **WER 95,68 %**, což potvrzuje, že rozmazání textu má výrazný negativní vliv na kvalitu výsledného převodu.

V posledním testu byly řádky vstupního dokumentu deformovány (Příloha A.5). V tomto případě proběhla správně jak detekce řádků, tak samotné rozpoznání textu. Přestože jsou výsledné

hodnoty **CER 57,84 %** a **WER 92,53 %** stále poměrně vysoké, aplikace si s tímto typem úpravy poradila nejlépe ze všech testovaných variant.

Z provedených testů vyplývá, že kvalita vstupního dokumentu má zásadní vliv na přesnost rozpoznávání textu. I když detekce textových řádků ve všech případech proběhla správně, samotné rozpoznávání textu bylo výrazně ovlivněno různými typy degradace obrazu. Největší negativní vliv mělo natočení dokumentu a rozmazání textu, zatímco deformace řádků měla na výslednou přesnost menší dopad. Tyto výsledky potvrzují, že pro dosažení kvalitního rozpoznávání je nezbytné pracovat s co nejkvalitnějšími vstupními dokumenty nebo provést jejich vhodné předzpracování před samotným převodem textu.

6 Problémy při vytváření aplikace

Tato kapitola se zaměřuje na technické problémy, které se objevily během vývoje aplikace, a na jejich následné řešení. Při práci s knihovnou Kraken a dalšími závislostmi bylo nutné řešit zejména otázky kompatibility prostředí a správného nastavení jednotlivých komponent. Tyto problémy měly přímý vliv na funkčnost aplikace, a proto bylo důležité najít vhodná řešení zajišťující její stabilní běh.

První část kapitoly se věnuje vytvoření virtuálního prostředí a instalaci knihovny Kraken, kde bylo nutné přizpůsobit verzi Pythonu kvůli omezené kompatibilitě. Druhá část se zabývá problémem s načítáním modelu pro detekci textových řádků, který je klíčový pro správnou funkci rozpoznávání textu.

6.1 Virtuální prostředí a instalace Kraken

Jak již bylo uvedeno v kapitole 4.2, složka **venv** slouží jako virtuální prostředí, ve kterém jsou nainstalovány všechny knihovny potřebné pro správnou funkčnost programu. Virtuální prostředí umožňuje oddělit závislosti projektu od ostatních instalací Pythonu v systému, čímž přispívá ke stabilitě aplikace a usnadňuje její správu.

Při jeho vytváření však nastal problém související s verzí Pythonu. Knihovna Kraken není plně kompatibilní s nejnovější verzí Pythonu, a proto bylo nutné použít starší podporovanou verzi, konkrétně Python 3.11. Virtuální prostředí bylo následně vytvořeno pomocí příkazu **python3.11 -m venv venv**. Po jeho vytvoření již bylo možné bez problémů nainstalovat všechny potřebné knihovny a pokračovat ve vývoji aplikace.

6.2 Knihovna blla

Při vývoji testovací verze aplikace nastal problém související s knihovnou **blla.mlmodel**, která je využívána pro detekci a segmentaci textových řádků ve vstupním obrazu. Tento model je součástí knihovny Kraken a slouží k analýze rozložení dokumentu, konkrétně k identifikaci jednotlivých textových řádků před samotným rozpoznáváním textu. Bez správné detekce řádků není možné provést následné rozpoznání textu dostatečně přesně, protože OCR model očekává vstup ve formě jednotlivých textových řádků.

V průběhu vývoje se však ukázalo, že program nedokázal tento model automaticky načíst z instalačního adresáře knihovny Kraken. Aby bylo možné zajistit stabilní fungování aplikace, byl soubor **blla.mlmodel** ručně zkopírován do adresáře *models* v rámci projektu. Cesta k tomuto souboru je následně pevně definována na začátku programu, což umožňuje jeho spolehlivé načtení při každém spuštění aplikace.

Závěr

Cílem bakalářské práce bylo vytvořit aplikaci pro převod textu z obrazových souborů do digitálně upravitelné podoby, což se podařilo splnit. V rámci práce byla nejprve představena problematika optického rozpoznávání znaků (OCR), včetně jejího historického vývoje, principu fungování a faktorů ovlivňujících přesnost rozpoznávání. Dále byl proveden přehled vybraných současných řešení, která ilustrují aktuální možnosti a přístupy v oblasti rozpoznávání textu.

V praktické části byla navržena a implementována aplikace v jazyce Python s využitím knihovny Kraken a dalších podpůrných nástrojů. Byla vytvořena struktura aplikace, uživatelské rozhraní a jednotlivé kroky zpracování, od nahrání vstupního souboru přes předzpracování obrazu až po samotné rozpoznání textu a jeho uložení. Součástí práce bylo také trénování vlastního jazykového modelu a jeho postupné zlepšování pomocí rozšiřování trénovacích dat.

Výsledky testování ukázaly, že kvalita rozpoznávání je výrazně ovlivněna jak kvalitou vstupních dokumentů, tak úrovní natrénování jazykového modelu. I přes zlepšení dosažené během trénovacích cyklů zůstává přesnost vytvořeného řešení nižší ve srovnání s pokročilými nástroji, jako je například Transkribus. Tento rozdíl je pravděpodobně způsoben především omezeným množstvím trénovacích dat a kratší dobou vývoje modelu.

Přínosem práce je vytvoření funkční aplikace a praktické ověření možností využití OCR technologií v prostředí Pythonu. Získané poznatky ukazují, že pro dosažení vyšší přesnosti by bylo vhodné pokračovat v trénování modelu na rozsáhlejších datech, případně optimalizovat jednotlivé kroky předzpracování obrazu. Do budoucna by bylo možné aplikaci dále rozšířit například o podporu dalších jazykových modelů, automatizované zlepšování výsledků nebo integraci pokročilejších metod umělé inteligence.

Seznam použité literatury

A generalized model for English printed text. Online. 2019, Published February 26, 2019. Dostupné z: https://zenodo.org/records/2577813. [cit. 2026-03-08].

ABBYY. *ABBYY FineReader PDF: the smarter PDF solution*. Online. 2026. Dostupné z: https://pdf.abbyy.com/. [cit. 2026-03-16].

ADOBE. *Now your scanner is in your back pocket, so you can scan a document wherever you go*. Online. 2026. Dostupné z: https://www.adobe.com/acrobat/mobile/scanner-app.html. [cit. 2026-03-16].

AMAZON. *What is Amazon Textract?* Online. 2026, 2026. Dostupné z: https://docs.aws.amazon.com/textract/latest/dg/what-is.html. [cit. 2026-03-03].

COURSERA. *What Is Keras? Your 2026 Guide*. Online. 2025, Dec 9, 2025. Dostupné z: https://www.coursera.org/articles/what-is-keras. [cit. 2026-03-16].

D. COSTA, Claire. *Top 12 Python GUI Frameworks for Developers*. Online. Builtin. 2025, Sep 20, 2024. Dostupné z: https://builtin.com/software-engineering-perspectives/python-gui. [cit. 2025-12-17].

GUROVA, Dora. *What is Streamlit: All Why's and How's Answered*. Online. UI Bakery. 2025, Updated: November 4, 2025. Dostupné z: https://uibakery.io/blog/what-is-streamlit. [cit. 2026-03-03].

History of Python. Online. 2025, Last Updated : 04 Oct, 2025. Dostupné z: https://www.geeksforgeeks.org/python/history-of-python/. [cit. 2025-12-21].

HOLDSWORTH, Jim. *What is optical character recognition (OCR)?* Online. IBM. 2025. Dostupné z: https://www.ibm.com/think/topics/optical-character-recognition. [cit. 2025-11-20].

HTR model for German manuscripts trained from several datasets. Online. 2023, Published May 13, 2023. Dostupné z: https://zenodo.org/records/7933463. [cit. 2026-03-08].

HYNKOVA, Katerina. *Ultimate guide to torchvision library in Python*. Online. Deepnote. 2025, Updated on August 22, 2025. Dostupné z: https://deepnote.com/blog/ultimate-guide-to-torchvision-library-in-python. [cit. 2026-03-03].

Character Error Rate (CER) Calculator. Online. 2026. Dostupné z: https://taylorarchibald.com/cer/. [cit. 2026-03-12].

CHOWDHARY, Krishi. *Best OCR software of 2025*. Online. Techradar. 2025, last updated October 24, 2025. Dostupné z: https://www.techradar.com/best/best-ocr-software. [cit. 2025-12-17].

KHANDELWAL, Neetika. *Image Processing in Python: Algorithms, Tools, and Methods You Should Know*. Online. Neptune.ai. 2025, 21st January, 2025. Dostupné z: https://neptune.ai/blog/image-processing-python. [cit. 2025-12-17].

Kraken HTR recognition model, Bohemian provenance 19th century. Online. 2024, Published June 15, 2024. Dostupné z: https://zenodo.org/records/11673242. [cit. 2026-03-08].

Kraken. Online. 2015, 2026. Dostupné z: https://kraken.re/main/index.html. [cit. 2026-03-08].

MARTINEZ, Hector. *Introduction to Gradio for Building Interactive Applications*. Online. Pyimagesearch. 2025, February 3, 2025. Dostupné z: https://pyimagesearch.com/2025/02/03/introduction-to-gradio-for-building-interactive-applications/. [cit. 2026-03-03].

MARTINEZ, Jorge. *What is the OCR Accuracy and How it Can be Improved*. Online. DocuClipper. 2025, January 6, 2025. Dostupné z: https://www.docuclipper.com/blog/ocr-accuracy/. [cit. 2026-03-02].

MICROSOFT. *Azure Document Intelligence in Foundry Tools*. Online. 2026, 2026. Dostupné z: https://azure.microsoft.com/en-us/products/ai-foundry/tools/document-intelligence. [cit. 2026-03-03].

Normalizované školní písmo. Online. 2007, 2007. Dostupné z: https://typomil.com/2007/01/normalizovane-skolni-pismo/. [cit. 2026-03-08].

OPENAI. *ChatGPT - 5.2*. Online. 2026, March 2026. Dostupné z: https://chatgpt.com/. [cit. 2026-03-10].

READ-COOP. *Transkribus*. Online. 2019, 2026. Dostupné z: https://www.transkribus.org/. [cit. 2026-03-02].

ROSSUM. *What is OCR Technology?* Online. 2023, July 27, 2023. Dostupné z: https://rossum.ai/blog/what-is-ocr-technology/. [cit. 2025-11-20].

SAIVE, Ravi. *Top 7 Python OCR Libraries for Text Extraction from Images*. Online. Tecmint. 2024, November 15, 2024. Dostupné z: https://www.tecmint.com/python-text-extraction-from-images/. [cit. 2026-03-16].

SHAIP. *Co je optické rozpoznávání znaků (OCR) – význam, typy, výhody a aplikace*. Online. 2022, May 10, 2022. Dostupné z: https://cs.shaip.com/blog/ocr-overview-and-applications/. [cit. 2025-11-20].

Státní okresní archiv Český Krumlov; Městský národní výbor Český Krumlov; inv. č. 3. Kronika města Český Krumlov. Datace (1945) 1954–1955. In: *DigiArchiv Státního oblastního archivu v Třeboni* [online]. Státní oblastní archiv v Třeboni, © 2007–2026, poslední aktualizace 29. 12. 2014, [cit. 2026-03-10]. Dostupné z: https://digi.ceskearchivy.cz/1337

Streamlit documentation. Online. 2026, 2026. Dostupné z: https://docs.streamlit.io/. [cit. 2026-03-08].

TRIVEDI, Ayushi. *Top 8 OCR Libraries in Python to Extract Text from Image*. Online. Analytics Vidhya. 2025, Last Updated : 16 Apr, 2025. Dostupné z: https://www.analyticsvidhya.com/blog/2024/04/ocr-libraries-in-python/. [cit. 2025-12-17].

TRIVEDI, Ayushi. *Top 8 OCR Libraries in Python to Extract Text from Image*. Online. Analytics Vidhya. 2025, 16 Apr, 2025. Dostupné z: https://www.analyticsvidhya.com/blog/2024/04/ocr-libraries-in-python/. [cit. 2026-03-16].

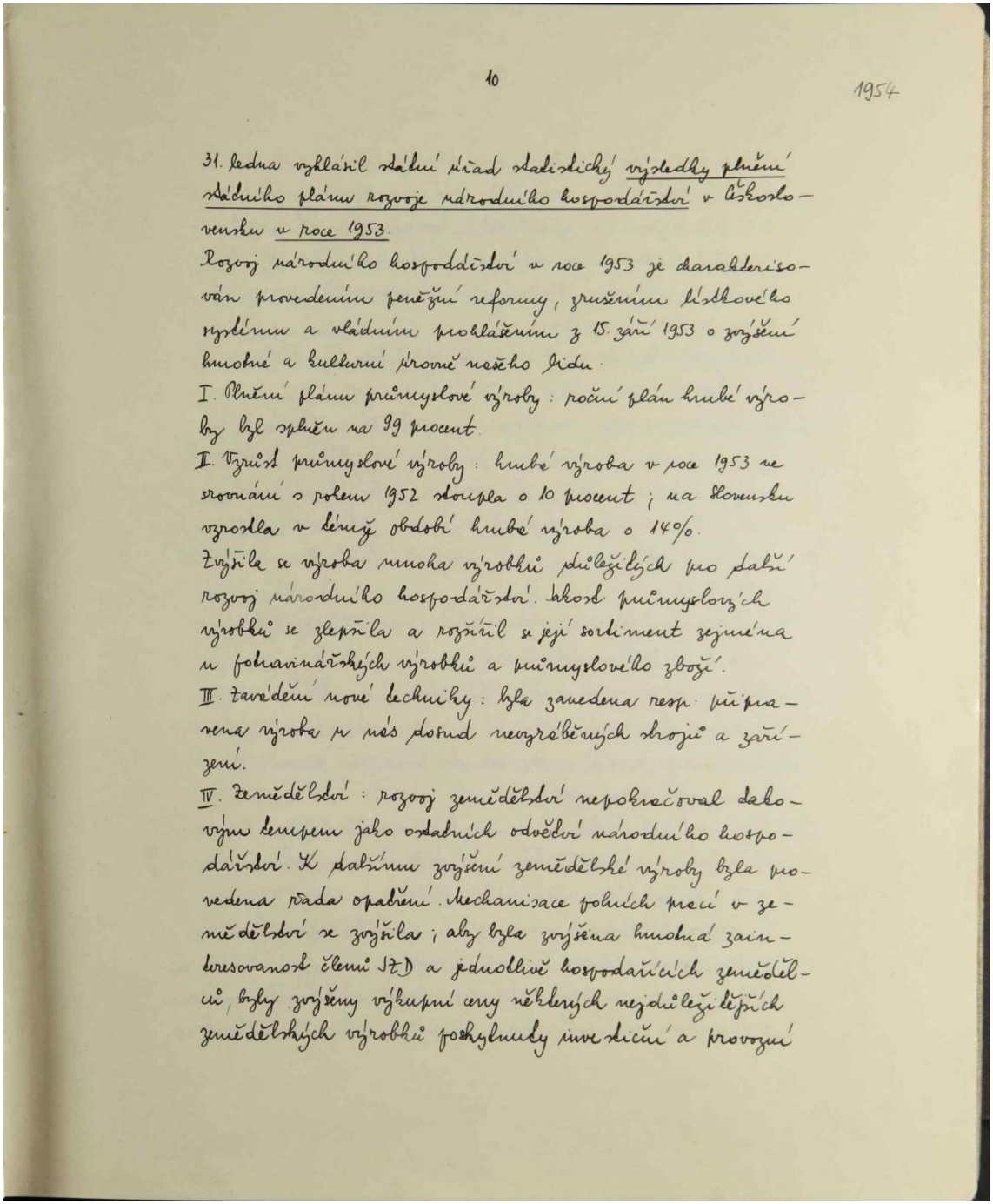
Word Error Rate Calculator. Online. 2026. Dostupné z: https://kensho.com/scribe/wer-calculator. [cit. 2026-03-14].

ZENODO. *About Zenodo*. Online. 2025. Dostupné z: https://about.zenodo.org/. [cit. 2026-03-01].

Přílohy

Přílohy A – Vstupní soubory

Příloha A.1 10. strana českokrumlovské kroniky



10

1954

31. ledna vyhlásil státní úřad statistický výsledky plnění státního plánu rozvoje národního hospodářství v Československu v roce 1953.

Rozvoj národního hospodářství v roce 1953 je charakterisován provedením peněžní reformy, zrušením lístkového systému a vládním prohlášením z 15. září 1953 o zvýšení hmotné a kulturní úrovně našeho lidu.

I. Plnění plánu průmyslové výroby: roční plán hrubé výroby byl splněn na 99 procent.

II. Vzrůst průmyslové výroby: hrubá výroba v roce 1953 ve srovnání s rokem 1952 stoupla o 10 procent; na Slovensku vzrostla v témž období hrubá výroba o 14%.

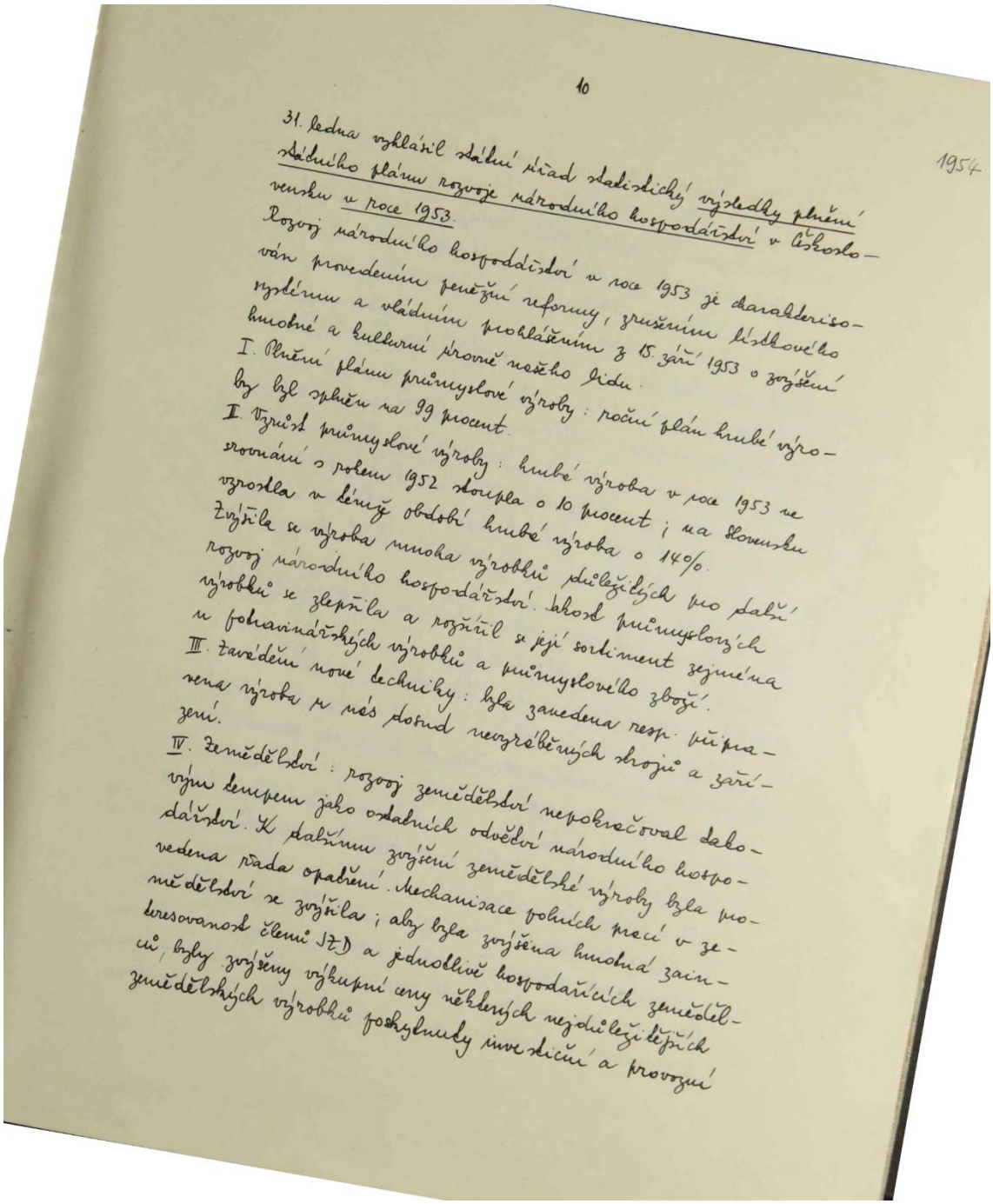
Zvýšila se výroba mnoha výrobků důležitých pro další rozvoj národního hospodářství. Jakost průmyslových výrobků se zlepšila a rozšířil se její sortiment zejména u potravinářských výrobků a průmyslového zboží.

III. Zavádění nové techniky: byla zavedena resp. připravena výroba u nás dosud nevyráběných strojů a zařízení.

IV. Zemědělství: rozvoj zemědělství nepokračoval takovým tempem jako odvětví národního hospodářství. K dalšímu zvýšení zemědělské výroby byla provedena řada opatření. Mechanisace polních prací v zemědělství se zvýšila; aby byla zvýšena hmotná zainteresovanost členů JZD a jednotlivě hospodařících zemědělců, byly zvýšeny výkupní ceny některých nejdůležitějších zemědělských výrobků poskytnuty investiční a provozní

Zdroj: Státní okresní archiv Český Krumlov (2014)

Příloha A.2 Vstupní soubor – natočený



10

1954

31. ledna vydával státní úřad statistický výsledky plnění státního plánu rozvoje národního hospodářství v Československu v roce 1953.

Rozvoj národního hospodářství v roce 1953 je charakterizován provedením peněžní reformy, zrušením lístkového hmotné a kulturní úrovně našeho lidu.

I. Plnění plánu průmyslové výroby: roční plán hrubé výroby byl splněn na 99 procent.

II. Vzrůst průmyslové výroby: hrubá výroba v roce 1953 ve srovnání s rokem 1952 stoupla o 10 procent; na Slovensku vzrostla v témž období hrubá výroba o 14%.

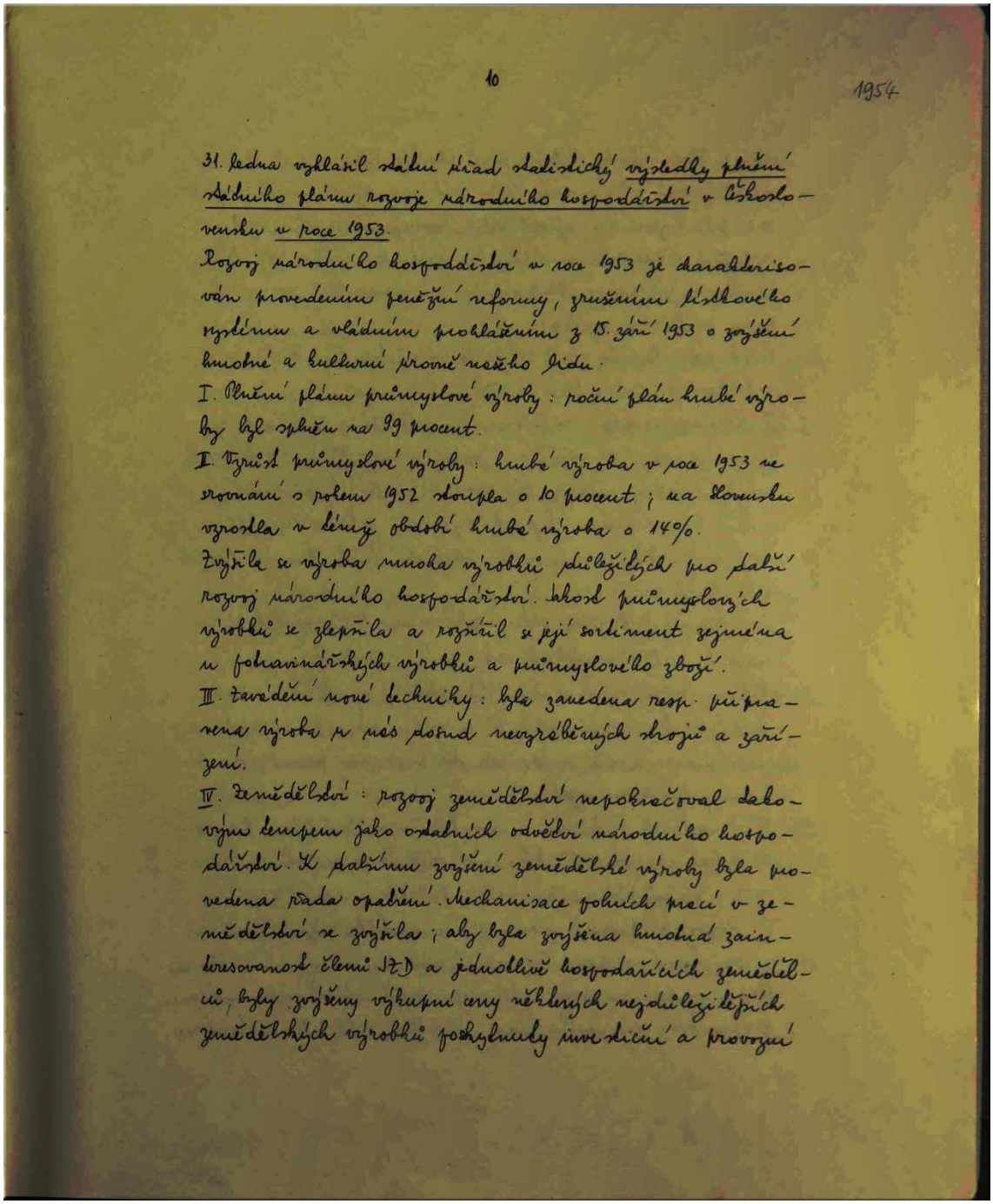
Zvýšila se výroba mnoha výrobků důležitých pro další rozvoj národního hospodářství. Sortiment průmyslových výrobků se zlepšila a rozšířil se její sortiment zejména u potravinářských výrobků a průmyslového zboží.

III. Zavádění nové techniky: byla zavedena resp. připravena výroba u nás dosud nevyráběných strojů a zařízení.

IV. Zemědělství: rozvoj zemědělství nepokračoval dalším tempem jako odvětví národního hospodářství. K dalšímu zvýšení zemědělské výroby byla provedena řada opatření. Mechanisace pohlavních prací v zemědělství se zvýšila; aby byla zvýšena hmotná zainteresovanost členů JZD a jednotlivě hospodařících zemědělců, byly zvýšeny výkupní ceny některých nejdůležitějších zemědělských výrobků, poskytnuty investiční a provozní

Zdroj: Vlastní zpracování

Příloha A.3 Vstupní soubor – ztmavený



10

1954

31. ledna vyhlásil státní úřad statistický výsledky plnění státního plánu rozvoje národního hospodářství v Československu v roce 1953.

Rozvoj národního hospodářství v roce 1953 je charakterizován provedením peněžní reformy, zrušením lístkového systému a vládním prohlášením z 15. září 1953 o zvýšení hmotné a kulturní úrovně našeho lidu.

I. Plnění plánu průmyslové výroby: roční plán hrubé výroby byl splněn na 99 procent.

II. Vzrůst průmyslové výroby: hrubá výroba v roce 1953 ve srovnání s rokem 1952 stoupla o 10 procent; na Slovensku vzrostla v témž období hrubá výroba o 14%.

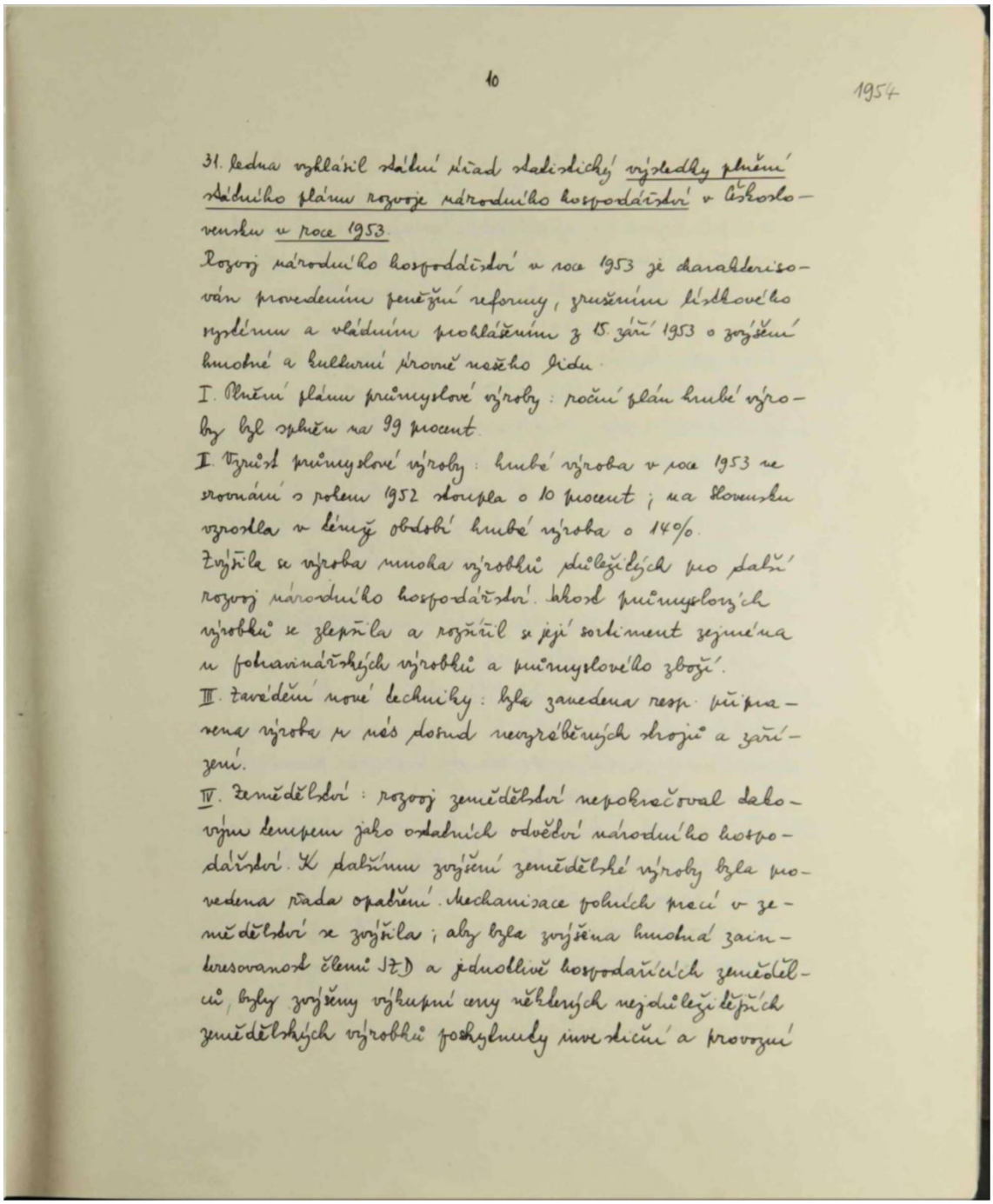
Zvýšila se výroba mnoha výrobků důležitých pro další rozvoj národního hospodářství. Jakost průmyslových výrobků se zlepšila a rozšířil se její sortiment zejména u potravinářských výrobků a průmyslového zboží.

III. Zavádění nové techniky: byla zavedena resp. připravena výroba mnoha dosud nevyráběných strojů a zařízení.

IV. Zemědělství: rozvoj zemědělství nepokračoval takovým tempem jako odvětví národního hospodářství. K dalšímu zvýšení zemědělské výroby byla provedena řada opatření. Mechanisace polních prací v zemědělství se zvýšila; aby byla zvýšena hmotná zainteresovanost členů JZD a jednotlivě hospodařících zemědělců, byly zvýšeny výkupní ceny některých nejdůležitějších zemědělských výrobků, poskytnuty investiční a provozní

Zdroj: Vlastní zpracování

Příloha A.4 Vstupní soubor – rozmazaný



10

1954

31. ledna vyhlásil státní úřad statistický výsledky plnění státního plánu rozvoje národního hospodářství v Československu v roce 1953.

Rozvoj národního hospodářství v roce 1953 je charakterizován provedením peněžní reformy, zrušením lístkového systému a vládním prohlášením z 15. září 1953 o zvýšení hmotné a kulturní úrovně našeho lidu.

I. Plnění plánu průmyslové výroby: roční plán hrubé výroby byl splněn na 99 procent.

II. Vzrůst průmyslové výroby: hrubá výroba v roce 1953 ve srovnání s rokem 1952 stoupla o 10 procent; na Slovensku vzrostla v témž období hrubá výroba o 14%.

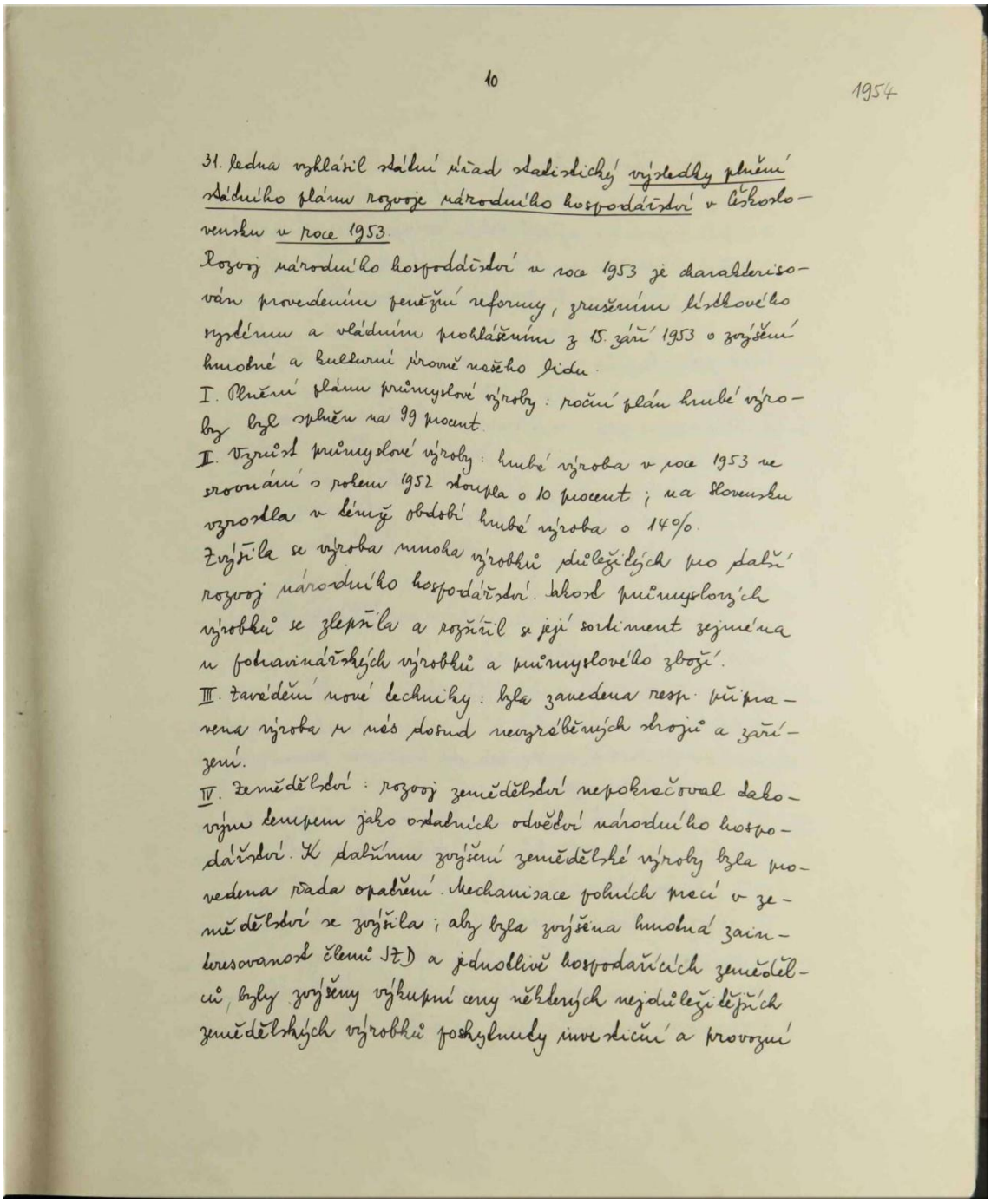
Zvýšila se výroba mnoha výrobků důležitých pro další rozvoj národního hospodářství. Jakost průmyslových výrobků se zlepšila a rozšířil se její sortiment zejména u potravinářských výrobků a průmyslového zboží.

III. Zavádění nové techniky: byla zavedena resp. připravena výroba u nás dosud nevyráběných strojů a zařízení.

IV. Zemědělství: rozvoj zemědělství nepokračoval takovým tempem jako odvětví národního hospodářství. K dalšímu zvýšení zemědělské výroby byla provedena řada opatření. Mechanisace pohlních prací v zemědělství se zvýšila; aby byla zvýšena hmotná zainteresovanost členů JZD a jednotlivě hospodařících zemědělců, byly zvýšeny výkupní ceny některých nejdůležitějších zemědělských výrobků, poskytnuty investiční a provozní

Zdroj: Vlastní zpracování

Příloha A.5 Vstupní soubor – defekt řádků



10 1954

31. ledna vyhlásil státní úřad statistický výsledky plnění státního plánu rozvoje národního hospodářství v Československu v roce 1953.

Rozvoj národního hospodářství v roce 1953 je charakterizován provedením peněžní reformy, zrušením lístkového systému a vládním prohlášením z 15. září 1953 o zvýšení hmotné a kulturní úrovně našeho lidu.

I. Plnění plánu průmyslové výroby: roční plán hrubé výroby byl splněn na 99 procent.

II. Vzrůst průmyslové výroby: hrubá výroba v roce 1953 ve srovnání s rokem 1952 stoupla o 10 procent; na Slovensku vzrostla v témž období hrubá výroba o 14%.

Zvýšila se výroba mnoha výrobků důležitých pro další rozvoj národního hospodářství. Jakost průmyslových výrobků se zlepšila a rozšířil se její sortiment zejména u potravinářských výrobků a průmyslového zboží.

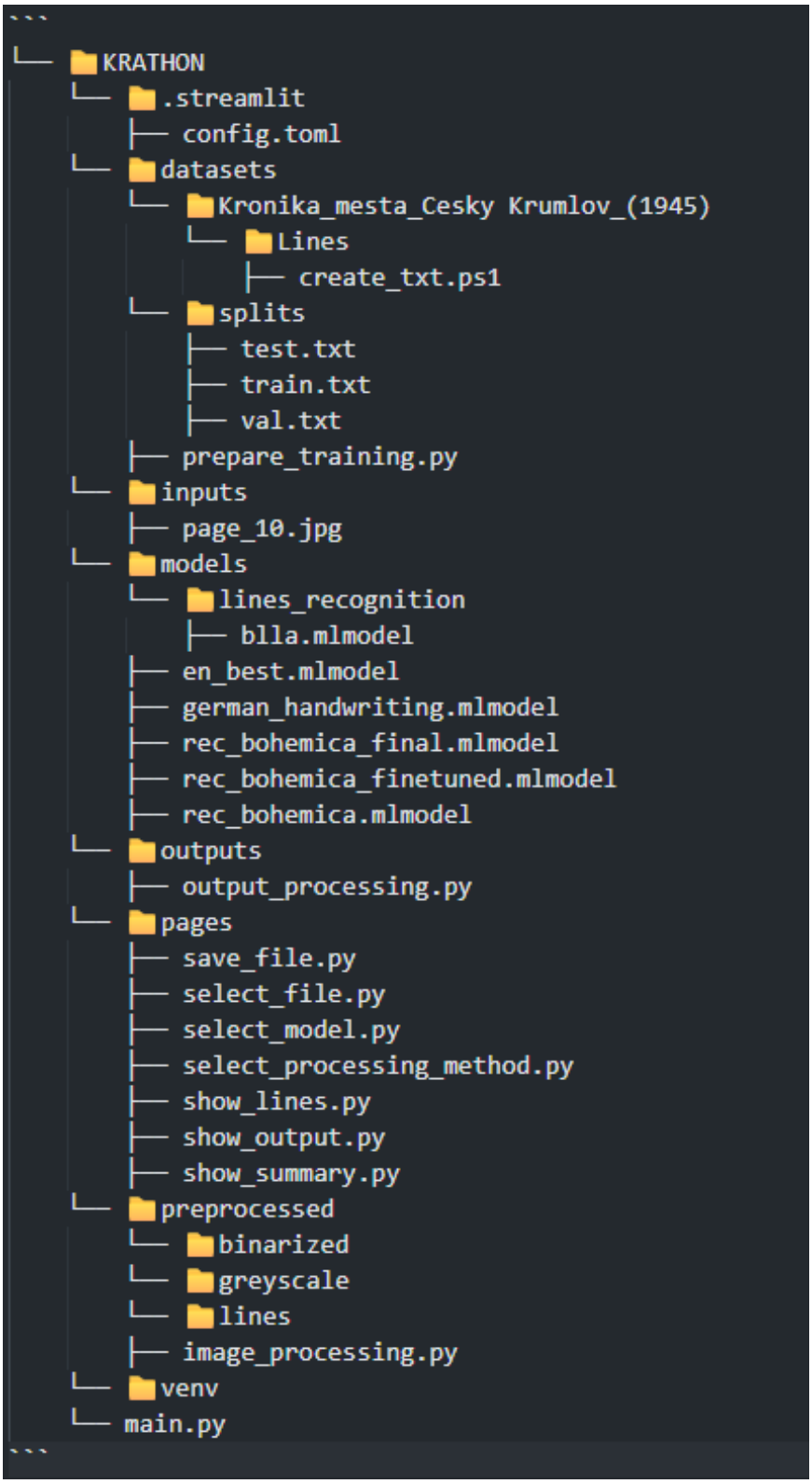
III. Zavádění nové techniky: byla zavedena resp. připravena výroba mnoha dosud nevyráběných strojů a zařízení.

IV. Zemědělství: rozvoj zemědělství nepokračoval takovým tempem jako odvětví národního hospodářství. K dalšímu zvýšení zemědělské výroby byla provedena řada opatření. Mechanizace polních prací v zemědělství se zvýšila; aby byla zvýšena hmotná zainteresovanost členů JZD a jednotlivě hospodařících zemědělců, byly zvýšeny výkupní ceny některých nejdůležitějších zemědělských výrobků, poskytnuty investiční a provozní

Zdroj: Vlastní zpracování

Přílohy B – Struktura aplikace a ukázky kódu

Příloha B.1 – Struktura aplikace



Zdroj: Vlastní zpracování

Příloha B.2 – main.py

```python
import os

import streamlit as st

def init_session():
    st.session_state.setdefault("file_name", None)
    st.session_state.setdefault("file_bytes", None)

    st.session_state.setdefault("processing_method_name", None)
    st.session_state.setdefault("processing_method_image", None)

    st.session_state.setdefault("recognition_model_name", None)
    st.session_state.setdefault("recognition_model_path", None)

    st.session_state.setdefault("get_lines_output", None)
    st.session_state.setdefault("draw_lines_output", None)


    st.session_state.setdefault("preview_bytes", None)
    st.session_state.setdefault("kraken_output", None)

st.set_page_config(page_title="KRATHON", layout="wide")

init_session()

with st.container(horizontal_alignment = "center"):
    with st.container(width = 750):
        st.divider()
        st.title("KRATHON")
        st.divider()
        st.write("Aplikace KRATHON slouží k převodu ručně psaného textu ze vstupních obrazových souborů do digitální podoby ve formě editovatelného textového souboru.")
        st.divider()
        st.write("Aplikace je založena na knihovně Kraken, která zajišťuje samotný proces rozpoznávání ručně psaného textu.")
        st.write("Pro předzpracování a úpravu vstupních obrazových dat je využita knihovna Pillow.")
        st.write("Uživatelské rozhraní aplikace je realizováno pomocí frameworku Streamlit.")
        st.divider()
        st.page_link(".\\pages\\select_file.py", label="Start programu", width = "content")
        st.divider()
```

Zdroj: Vlastní zpracován

Příloha B.3 – select_file.py

```python
import io
import streamlit as st
from PIL import Image

st.set_page_config(page_title="KRATHON", layout="wide")
file_types = ["png", "jpg", "jpeg"]

with st.container(horizontal_alignment = "center"):
    with st.container(width = 750):
        st.divider()
        st.header("VÝBĚR SOUBORU PRO PŘEVOD")
        st.divider()
        st.write("Vyber soubor: ")

        image = st.file_uploader( label= "Vyber soubor", type = file_types, label_visibility = "hidden")



        if image is not None:
            st.session_state.file_name = image.name
            st.session_state.file_bytes = image.getvalue()
            st.success(f"Nahráno: {image.name}")

        if st.session_state.get("file_bytes"):
            img = Image.open(io.BytesIO(st.session_state.file_bytes))
            st.image(img, caption="Nahraný obrázek", width = "content")

        st.divider()
        st.page_link(".\\pages\\select_processing_method.py", label="Pokračovat",
                     disabled=not st.session_state.get("file_bytes"), width = "content")
        st.divider()
```

Zdroj: Vlastní zpracování

Příloha B.4 – select_processing_method.py

```python
import io
import streamlit as st

from preprocessed.image_processing import get_greyscale, get_binarized

GREYSCALE_FILE_PATH = '.\\preprocessed\\greyscale\\greyscale_'
BINARIZED_FILE_PATH = '.\\preprocessed\\binarized\\binarized_'


st.set_page_config(page_title="KRATHON", layout="wide")


with st.container(horizontal_alignment = "center"):
    with st.container(width = 750):
        st.divider()
        st.header("METODA PŘEDZPRACOVÁNÍ SOUBORU")
        st.divider()
        st.write("Zvol způsob předzpracování: ")


        image_processing_method = st.radio(
            "Select image processing method",
            ["Greyscale","Binarizace"],
            captions = [
                "Zachovává intenzitní informace obrazu a jemné detaily rukopisu. Výhodné pro většinu vstupních dat.",
                "Redukuje obraz na popředí a pozadí. Může zlepšit kontrast.U nekvalitních vstupů může zvýraznit šum."
            ],
            label_visibility = "hidden"
        )


        match image_processing_method:
            case "Greyscale":
                greyscale_image = get_greyscale(GREYSCALE_FILE_PATH, st.session_state.file_name)
                st.image(greyscale_image, caption = "Greyscale image", width="stretch")
                st.session_state.processing_method_name = "Greyscale"
                st.session_state.processing_method_image = greyscale_image

            case "Binarizace":
                binarized_image = get_binarized(BINARIZED_FILE_PATH, st.session_state.file_name)
                st.image(binarized_image, caption = "Binarized image", width="stretch")
                st.session_state.processing_method_name = "Binarizace"
                st.session_state.processing_method_image = binarized_image

        st.divider()
        st.page_link(".\\pages\\select_model.py", label="Pokračovat",
                     disabled=not st.session_state.get("processing_method_name"), width = "content")
        st.divider()
```

Zdroj: Vlastní zpracování

Příloha B.5 – select_model.py

```python
import streamlit as st

#Language model paths
EN_BEST_MODEL = '.\\models\\en_best.mlmodel'
GERMAN_HANDWRITING_MODEL = '.\\models\\german_handwriting.mlmodel'
REC_BOHEMICA = '.\\models\\rec_bohemica.mlmodel'
REC_BOHEMICA_FINETUNED = '.\\models\\rec_bohemica_finetuned.mlmodel'
REC_BOHEMICA_FINAL = '.\\models\\rec_bohemica_final.mlmodel'

st.set_page_config(page_title="KRATHON", layout="wide")

with st.container(horizontal_alignment = "center"):
    with st.container(width = 750):
        st.divider()
        st.header("JAZYKOVÝ MODEL")
        st.divider()
        st.write("Vyber jazykový model: ")

        language_model = st.radio(
            "Select image processing method",
            ["EN_BEST","GERMAN_HANDWRITING", "REC_BOHEMICA", "REC_BOHEMICA_FINETUNED", "REC_BOHEMICA_FINAL"],
            label_visibility = "hidden"
        )


        match language_model:
            case "EN_BEST":
                st.info("Obecný model pro anglický text.")
                st.session_state.recognition_model_name = "EN_BEST"
                st.session_state.recognition_model_path = EN_BEST_MODEL
            case "GERMAN_HANDWRITING":
                st.info("Model trénovaný na německých rukopisech.")
                st.session_state.recognition_model_name = "GERMAN_HANDWRITING"
                st.session_state.recognition_model_path = GERMAN_HANDWRITING_MODEL
            case "REC_BOHEMICA":
                st.info("Model trénovaný na historických rukopisech z českých zemí (19. století).")
                st.session_state.recognition_model_name = "REC_BOHEMICA"
                st.session_state.recognition_model_path = REC_BOHEMICA
            case "REC_BOHEMICA_FINETUNED":
                st.info("Model REC_BOHEMICA trénovaný na 200 řádcích českokrumlovské kroniky z roku 1954.")
                st.session_state.recognition_model_name = "REC_BOHEMICA_FINETUNED"
                st.session_state.recognition_model_path = REC_BOHEMICA_FINETUNED
            case "REC_BOHEMICA_FINAL":
                st.info("Model REC_BOHEMICA_FINETUNED trénovaný na 500 řádcích českokrumlovské kroniky z roku 1954.")
                st.session_state.recognition_model_name = "REC_BOHEMICA_FINAL"
                st.session_state.recognition_model_path = REC_BOHEMICA_FINAL

        st.divider()
        st.page_link(".\\pages\\show_lines.py", label="Pokračovat",
                     disabled=not st.session_state.get("recognition_model_path"), width = "content")
        st.divider()
```

Zdroj: Vlastní zpracování

Příloha B.6 – show_lines.py

```python
import streamlit as st

from preprocessed.image_processing import draw_lines, get_lines

st.set_page_config(page_title="KRATHON", layout="wide")


with st.container(horizontal_alignment = "center"):
    with st.container(width = 750):
        st.divider()
        st.header("ROZPOZNÁNÍ ŘÁDKŮ")
        st.divider()
        st.write("Náhled souboru po rozpoznání řádků: ")

        st.session_state.get_lines_output = get_lines(st.session_state.processing_method_image)
        st.session_state.draw_lines_output = draw_lines(st.session_state.get_lines_output)

        st.image(st.session_state.draw_lines_output)
        st.divider()
        st.page_link(".\\pages\\show_summary.py", label="Pokračovat", disabled=not st.session_state.get("draw_lines_output"), width = "content")
        st.divider()

```

Zdroj: Vlastní zpracování

Příloha B.7 – show_summary.py

```python
import io
import streamlit as st
from PIL import Image

st.set_page_config(page_title="KRATHON", layout="wide")

with st.container(horizontal_alignment = "center"):
    with st.container(width = 750):
        st.divider()
        st.title("KONTROLA VSTUPŮ PŘED SPUŠTĚNÍM")
        st.divider()
        st.header("Vstupní soubor: ")
        st.divider()
        st.write(Image.open(io.BytesIO(st.session_state.file_bytes)))
        st.divider()
        st.header("Metoda předzpracování: ")
        st.divider()
        st.write(st.session_state.processing_method_name)
        st.divider()
        st.title("Náhled souboru po předzpracování: ")
        st.divider()
        st.write(st.session_state.processing_method_image)
        st.divider()
        st.title("Zvolený jazykový model: ")
        st.divider()
        st.write(st.session_state.recognition_model_name)
        st.divider()
        st.title("Náhled souboru po rozpoznání řádků: ")
        st.divider()
        st.write(st.session_state.draw_lines_output)

        st.divider()
        st.page_link(".\\pages\\show_output.py", label="Pokračovat", width = "content")
        st.divider()
```

Zdroj: Vlastní zpracování

Příloha B.8 – show_output.py

```python
import streamlit as st

from kraken.lib import models
from outputs.output_processing import get_content

st.set_page_config(page_title="KRATHON", layout="wide")

with st.container(horizontal_alignment = "center"):
    with st.container(width = 750):
        st.divider()
        st.title("VÝSTUP PROGRAMU")
        st.divider()
        st.session_state.kraken_output = get_content(models.load_any(st.session_state.recognition_model_path), st.session_state.processing_method_image, st.session_state.get_lines_output)
        st.markdown(st.session_state.kraken_output)
        st.divider()
        with st.container():
            col_1, col_2, col_3 = st.columns(3, vertical_alignment = "center")
            col_1.page_link(".\\pages\\select_processing_method.py", label="Opakovat", disabled=not st.session_state.get("kraken_output"), icon = ":material/refresh:")
            col_3.page_link(".\\pages\\save_file.py", label="Uložit soubor", disabled=not st.session_state.get("kraken_output"), icon = ":material/save:")
        st.divider()
```

Zdroj: Vlastní zpracován

Příloha B.9 – image_processing.py

```python
import io
import streamlit as st
from PIL import Image, ImageDraw

from kraken import binarization, blla
from kraken.lib import vgsl

BLLA_MODEL_PATH = '.\\models\\lines_recognition\\blla.mlmodel'

LINES_FILE_PATH = '.\\preprocessed\\lines\\lines_'

def scale_image(image):
    image_size = image.size
    scale_width, scale_height = image_size
    scale_width = scale_width * 4
    scale_height = scale_height * 4
    scale = (scale_width, scale_height)
    resized_image = image.resize(scale)
    return resized_image

def get_greyscale(output_path, file_name):
    image = Image.open(io.BytesIO(st.session_state.file_bytes))
    greyscale = image.convert('L')
    greyscale.save(f"{output_path}{file_name}")
    return greyscale

def get_binarized(output_path, file_name):
    image = Image.open(io.BytesIO(st.session_state.file_bytes))
    bw_image = binarization.nlbin(image)
    bw_image.save(f"{output_path}{file_name}")
    return bw_image

def get_lines(preprocessed_image):
    blla_model = vgsl.TorchVGSLModel.load_model(BLLA_MODEL_PATH)
    return blla.segment(preprocessed_image, text_direction = 'horizontal-lr', model = blla_model, device = 'cpu')

def draw_lines(baseline_seg):

    lines = Image.open(io.BytesIO(st.session_state.file_bytes)).copy()
    draw = ImageDraw.Draw(lines)

    for line in baseline_seg.lines:
        baseline = line.baseline

        if baseline and len(baseline) > 1:
            draw.line(baseline, fill=(255, 0, 0), width=2)

    lines.save(LINES_FILE_PATH + st.session_state.file_name)
    return lines
```

Zdroj: Vlastní zpracování

Příloha B.10 – output_processing.py

```python
import streamlit as st
from kraken import rpred

OUTPUT_PATH = '.\\outputs\\'

def get_content(language_model, preprocessed_image, lines):
    pred_it = rpred.rpred(network = language_model, im = preprocessed_image, bounds = lines, pad = 32, bidi_reordering = True)

    lines_text = []

    for rec in pred_it:
        if rec.prediction:
            lines_text.append(rec.prediction)

    conversion_output = "\n".join(lines_text)
    return conversion_output

def save_content(conversion_output):
    file = st.session_state.file_name
    file_name = (file.rsplit('.', 1)[0])
    with open(f"{OUTPUT_PATH}{file_name}.txt", "w", encoding = "utf-8") as file:
        file.write(conversion_output)
```

Zdroj: Vlastní zpracování

Příloha B.11 – save_file.py

```python
import streamlit as st

from outputs.output_processing import save_content

st.set_page_config(page_title="KRATHON", layout="wide")

def reset_session():
    st.session_state.file_name = None
    st.session_state.file_bytes = None
    st.session_state.processing_method_name = None
    st.session_state.processing_method_image = None
    st.session_state.recognition_model = None
    st.session_state.get_lines_output = None
    st.session_state.draw_lines_output = None
    st.session_state.preview_bytes = None
    st.session_state.kraken_output = None

with st.container(horizontal_alignment = "center"):
    with st.container(width = 750):
        st.divider()
        st.title("SOUBOR ULOŽEN")
        st.divider()
        st.info("Textový soubor uložen do adresáře outputs.")

        save_content(st.session_state.kraken_output)

        st.divider()
        if st.page_link(".\\pages\\select_file.py", label="Novy soubor", width = "content"):
            reset_session()
        st.divider()
```

Zdroj: Vlastní zpracování