CADT
IDT
វិទ្យាស្ថានបច្ចេកវិទ្យាឌីជីថល
Institute of Digital Technology

ក្រសួងប្រៃសណីយ៍និងទូរគមនាគមន៍
MINISTRY OF POST AND TELECOMMUNICATIONS

1st Internship report on

Automating the Digitization of Printed and Handwritten Khmer Documents Using OCR

at

CADT
IDRI Institute of Digital Research & Innovation

Submitted By:

Mr. Lay Sopanha

Under the supervision of:

Dr. Va Hongly

Under the advisory of:

Ms. Mat Nab

Date of Defense: **August 27th, 2025**

This report is submitted in partial fulfillment of the requirements for the Bachelor of Computer Science degree with a specialization in Data Science in

Department of Computer Science

Faculty of Digital Engineering

www.cadt.edu.kh

បណ្ឌិត្យសភាបច្ចេកវិទ្យាឌីជីថលកម្ពុជា

វិទ្យាស្ថានបច្ចេកវិទ្យាឌីជីថល

មហាវិទ្យាល័យ វិស្វកម្មឌីជីថល

ដេប៉ាតឺម៉ង់ វិទ្យាសាស្ត្រកុំព្យូទ័រ

គម្រោងសញ្ញាបត្របរិញ្ញាបត្រ

របស់និស្សិត ៖ លោក ឡាយ សុបញ្ញា

កាលបរិច្ឆេទការពារនិក្ខេបបទ ៖ ថ្ងៃទី២៧ ខែសីហា ឆ្នាំ ២០២៥

អនុញ្ញាតអោយការពារគម្រោង ៖

ប្រធានបណ្ឌិត្យសភាបច្ចេកវិទ្យាឌីជីថលកម្ពុជា_____

រាជធានីភ្នំពេញ ថ្ងៃទី ____ ខែ _____ ឆ្នាំ _____

ប្រធានបទ ៖ស្វ័យប្រវត្តិកម្មការបំលែងឯកសារខ្មែរ អក្សរពុម្ព និងអក្សរសរសេ ដោយប្រើ OCR

សហគ្រាស ៖វិទ្យាស្ថានស្រាវជ្រាវនិងនវានុវត្តន៍ឌីជីថល

គ្រូជឹកនាំគម្រោង ៖កញ្ញា ម៉ាត់ ណាប់ _____

អ្នកទទួលខុសត្រូវក្នុងសហគ្រាស ៖លោក បណ្ឌិត ង៉ា ហុងលី _____

រាជធានីភ្នំពេញ

Cambodia Academy of Digital Technology

Institute of Digital Technology

Faculty of Digital Engineering

Department of Computer Science

INTERNSHIP REPORT OF

Mr. Lay Sopanha

Defense Date: August 27th, 2025

Internship Defense Authorization

President: _____

Phnom Penh,

Topic	**: Automating the Digitization of Printed and Handwritten Khmer Documents Using OCR**
Establishment	**: Institute of Digital Research & Innovation (IDRI)**
Advisor	**: Ms. Mat Nab** _____
Supervisor	**: Dr. Va Hongly** _____

PHNOM PENH

ACKNOWLEDGMENT

I would like to extend my heartfelt gratitude to all the individuals and institutions who contributed to the successful completion of my internship. This journey has been both remarkable and enriching, made possible by the guidance, encouragement, and support of many dedicated individuals.

First, I extend my express my deepest gratitude to **H.E. Dr. Seng Sopheap**, President of **CADT**, and **H.E. Hean Samboeun**, Vice President of **CADT**, for their commendable leadership and unwavering commitment to student success. Their dedication has greatly enriched our academic journey and made this internship program a reality.

I am also grateful to the Department of Computer Science for their efforts in coordinating the internship program and facilitating communication between academic and professional spheres, ensuring a smooth and meaningful experience.

Special thanks go to my academic advisor, **Ms. Mat Nab**, whose guidance, thoughtful feedback, and support were crucial throughout this project. Her expertise in Machine Learning and Deep Learning and her insightful suggestions helped me navigate the technical challenges, especially when the project required a strategic pivot toward a more innovative solution.

I sincerely thank my internship supervisor at the Institute of Digital Research & Innovation (IDRI), **Dr. Va Hongly**, for his mentorship and practical insights. His professional guidance and access to essential resources played a key role in shaping the final project and enhancing my understanding of real-world data science applications in Optical Character Recognition.

Lastly, my deepest gratitude goes to my beloved family. Your belief in me and continuous encouragement have been the emotional foundation of this journey. Your love and support have been the cornerstone of my perseverance and success.

To all who contributed to this endeavor, I offer my deepest appreciation.

មូលន័យសង្ខេប

របាយការណ៍នេះកត់ត្រាកម្មសិក្សារយៈពេលពីរខែរបស់ខ្ញុំនៅវិទ្យាស្ថានស្រាវជ្រាវ និងនវានុវត្តន៍ឌីជីថល (IDRI) លើគម្រោង "ការបំប្លែងឯកសារអក្សរពុម្ព និងអក្សរសរសេរដោយដៃខ្មែរទៅជាទម្រង់ឌីជីថលដោយប្រើ OCR"។ ការបំប្លែងឯកសារដែលមានទាំងអក្សរពុម្ព និងអក្សរសរសេរដោយដៃ ដូចជា សំបុត្រកំណើតខ្មែរ គឺជាបញ្ហាប្រឈមសម្រាប់ OCR ដោយសារតួអក្សរខ្មែរលំបាក និងកង្វះទិន្នន័យ។ គម្រោងចាប់ផ្តើមដោយមានគោលដៅបង្កើត OCR pipeline ពេញលេញ ប្រើ YOLOv8 សម្រាប់ចាប់អក្សរ និង TrOCR សម្រាប់សម្គាល់អក្សរ។ ទោះជាយ៉ាងណា ការវិភាគដំបូងបង្ហាញថាជាកង្វះធនធាន និងទិន្នន័យរារាំងការអនុវត្ត។ ដូច្នេះ គម្រោងបានប្តូរទិសដៅទៅ pipeline ម៉ូឌុលមានប្រសិទ្ធភាព ដោយប្រើ PaddleOCR ដែលបានបង្វឹកជាមុនលើទិន្នន័យបែបឯកសារ និងជៀសវាងការបង្កើតម៉ូឌុលចាប់អក្សរថ្មី។ ការរួមចំណែកស្នូល គឺការបង្កើត MobileNetV3 classifier មានទម្ងន់ស្រាល និងត្រឹមត្រូវខ្ពស់ សម្រាប់បែងចែកអក្សរពុម្ព និងអក្សរសរសេរដោយដៃ។ គំរូនេះបានបង្វឹកលើទិន្នន័យដែលបានបង្កើត និងដាក់ស្លាកដោយខ្លួនឯង។ ការចាត់ថ្នាក់អនុញ្ញាតឲ្យផ្នែកនីមួយៗនៃអត្ថបទ ត្រូវបានផ្ញើទៅកាន់ OCR សមស្រប ដូចជា Tesseract សម្រាប់អក្សរពុម្ព និង TrOCR (នឹងបង្វឹកបន្ថែមក្រោយ) សម្រាប់អក្សរសរសេរដោយដៃ។ ការរចនាម៉ូឌុលនេះ មិនត្រឹមតែដោះស្រាយបញ្ហាធនធានទេ ប៉ុន្តែថែមទាំងបង្កើតគ្រឹះអាចពង្រីក សម្រាប់បំប្លែងឯកសារខ្មែរចម្រុះទៅជាទម្រង់ឌីជីថលដោយស្វ័យប្រវត្តិ។

ABSTRACT

This report documents the two months of my internship at Institute of Digital Research & Innovation (IDRI) on **Automating the Digitization of Printed and Handwritten Khmer Documents Using OCR**. The digitization of structured documents containing a mix of printed and handwritten text, such as Khmer birth certificates, presents a significant challenge for traditional Optical Character Recognition (OCR) systems. This challenge is compounded by both the inherent complexity of the Khmer script and the scarcity of large-scale, domain-specific datasets. This project initially set out to develop a comprehensive end-to-end OCR pipeline leveraging a YOLOv8 object detector and a Transformer-based Recognition (TrOCR) model. However, initial analysis revealed that significant computational and data-related constraints would impede this approach's feasibility. Consequently, this report details a strategic pivot to a more modular, efficient, and innovative pipeline. The revised approach leverages the pre-trained PaddleOCR framework for robust document layout and table analysis, thereby passing the need for a custom-trained detection model. The core technical contribution of this project is the development of a lightweight, highly accurate MobileNetV3 classifier that distinguishes between printed and handwritten text snippets. Trained on a custom-annotated dataset of Khmer birth certificate excerpts, this text-type classification enables an intelligent downstream workflow in which text segments can be routed to the most appropriate, specialized OCR engine such as Tesseract for printed text and a future, fine-tuned TrOCR for handwriting. This modular design not only overcomes the initial resource limitations but also establishes a more practical and scalable foundation for the automated digitization of complex, mixed-media Khmer documents.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Explanation
AI	Artificial Intelligence
CCL	Connected-Component Labeling
CER	Character Error Rate
CNN	Convolutional Neural Network
GPU	Graphics Processing Unit
HOG	Histogram of Oriented Gradients
IDRI	Institute of Digital Research & Innovation
JSON	JavaScript Object Notation
mAP	mean Average Precision
ML	Machine Learning
MLOps	Machine Learning Operations
NER	Named Entity Recognition
NLP	Natural Language Processing
OCR	Optical Character Recognition
PoC	Proof of Concept
SVM	Support Vector Machine
TrOCR	Transformer Optical Character Recognition
ViT	Vision Transformer
WER	Word Error Rate
YOLO	You Only Look Once

I. INTROUDCTION

This section provides a brief overview of my internship, including its purpose, duration, and organizational context. As part of the 3rd year requirement for the Bachelor of Science in Computer Science, I completed a two-month internship from May to July 2025 at the Institute of Digital Research & Innovation (IDRI). The internship focused on supporting Cambodia's digital transformation by contributing to a project aimed at automating the extraction of information from official documents. The following sections of this report will detail the project objectives, methods, outcomes, and the experience gained throughout the internship and the organization I am interning at.

1.1. Presentation of Internship

As part of the final-year Bachelor of Science curriculum, I completed my internship at the Institute of Digital Research & Innovation (IDRI) from May to July 2025, supervised by Dr. Va Hongly. I worked as a Data Scientist, contributing to a project aimed at digitizing Khmer birth certificates through automated text extraction.

My responsibilities included preparing and annotating data, training and evaluating machine learning models, and designing a modular pipeline to separate and process printed and handwritten text. This hands-on experience allowed me to apply theoretical knowledge to a real-world problem, improve my understanding of OCR and data workflows, and collaborate effectively in a research-driven environment.

Challenges included limited data, complex document structure, and the need to adjust the project's scope to improve feasibility. Overall, the internship strengthened my problem-solving skills and gave me practical experience in building solutions for national digital transformation efforts.

1.1.1. Objective of Internship

The primary objective of this internship was to gain hands-on experience in the end-to-end development of a real-world machine learning project. Key goals included:

- Applying theoretical knowledge of computer vision and deep learning to solve the practical challenge of Khmer Optical Character Recognition (OCR).
- Gaining proficiency in modern MLOps tools and cloud-based platforms, specifically **Lightning AI** for model training and **Weights & Biases** for experiment tracking.

- Developing problem-solving and research skills by navigating technical limitations, such as hardware constraints and limited datasets, and adapting the project scope accordingly.
- Mastering the workflow of a data science project, from data collection and annotation to model implementation, evaluation, and pipeline integration using frameworks like **PaddleOCR**.

1.1.2. Duration of Internship

The internship at the Institute of Digital Research & Innovation (IDRI) was conducted over a period of 3 months. The official timeline for the project was from May 2nd, 2025, to July 31st, 2025. This period encompassed all phases of the project, including initial research, data preparation, model development and training, and final reporting. My working shift was full-time (7 hours per day) throughout this period, allowing for deep immersion and dedicated progress on the project, as highlighted in **Table 1**.

Table 1: Working hours

Time	Monday	Tuesday	Wednesday	Thursday	Friday
9:00AM – 11:45 AM	Yes	Yes	Yes	Yes	Yes
11:45AM – 1:00PM	LUNCH BREAK				
1:00PM – 4:30PM	Yes	Yes	Yes	Yes	Yes

1.2. Presentation of Organization

This section provides a comprehensive overview of the host institution, the Institute of Digital Research & Innovation (IDRI), covering its background, services, organizational structure, and mission.

1.2.1. General Information of Company

The Institute of Digital Research & Innovation (IDRI), operating under the Cambodia Academy of Digital Technology (CADT), is a leading public research institution dedicated to advancing digital technology and fostering innovation in Cambodia. Established as the nation's first-of-its-kind research institute for digital technology, IDRI plays a pivotal role in the country's technological advancement. It was formed as part of the transformation of the National Institute of Posts, Telecoms & ICT (NIPTICT) into CADT in 2020. Today, IDRI focuses on key areas of digital transformation, including artificial intelligence, data science,

and other emerging technologies, aiming to develop solutions that address national challenges and contribute to the growth of the digital economy. Through its research projects, strategic collaborations, and capacity-building initiatives, IDRI strives to be a center of excellence for digital innovation in the region.

1.2.2. Service of Organization

IDRI's services and products are centered around a core of research, innovation, and development to support Cambodia's digital society. Key functions include:

- **Research and Development:** Conducting specialized R&D with a focus on cutting-edge fields such as Artificial Intelligence (AI), Natural Language Processing (NLP), Data Science, and Cybersecurity.
- **Technology Transfer:** Ensuring that technological developments are made accessible to a wider range of users, facilitating their exploitation into new products, applications, and services.
- **Policy Research:** Performing rigorous studies and facilitating dialogues on digital development policy to assess the impact of technology adoption on the economy and society.
- **Digital Innovation Center:** Providing essential facilities like a Makerspace, co-innovation space, and startup offices to foster entrepreneurship and tech transfer.

1.2.3. Vision and Mission

- **Mission:** IDRI's mission is to conduct high-impact R&D and technology transfer in digital technology, with a focus on emerging fields like AI, IoT, and Data Science. It aims to promote digital innovation and technopreneurship, conduct rigorous policy studies, and, most importantly, recognize and nurture Cambodia's digital talent.
- **Vision:** To be Cambodia's flagship research and innovation institute with international recognition in digital technology, addressing key social and economic challenges amid rapid digital transformation.

1.2.4. Organization Chart

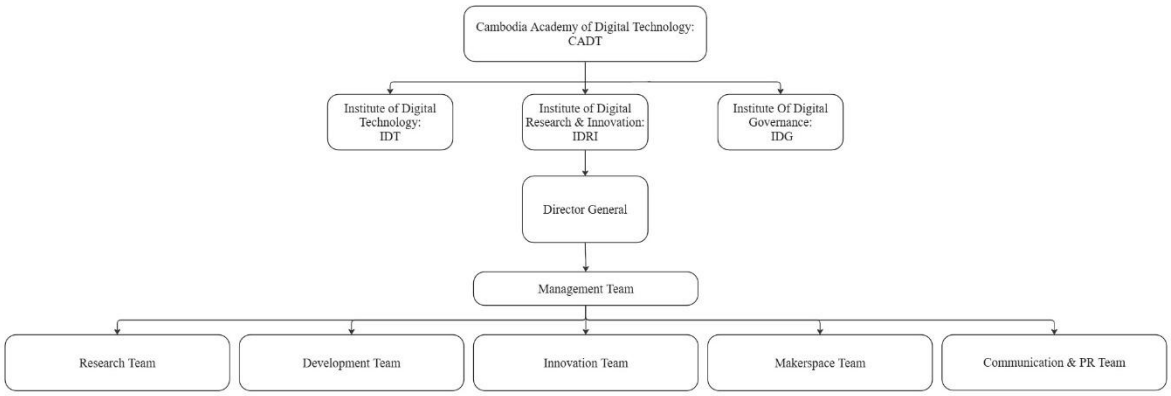

Figure 1: IDRI Organization Chart

1.2.5. Address and contact

- **Address:** National Road 6A, Kthor, Prek Leap Chroy Changvar, Phnom Penh, Cambodia

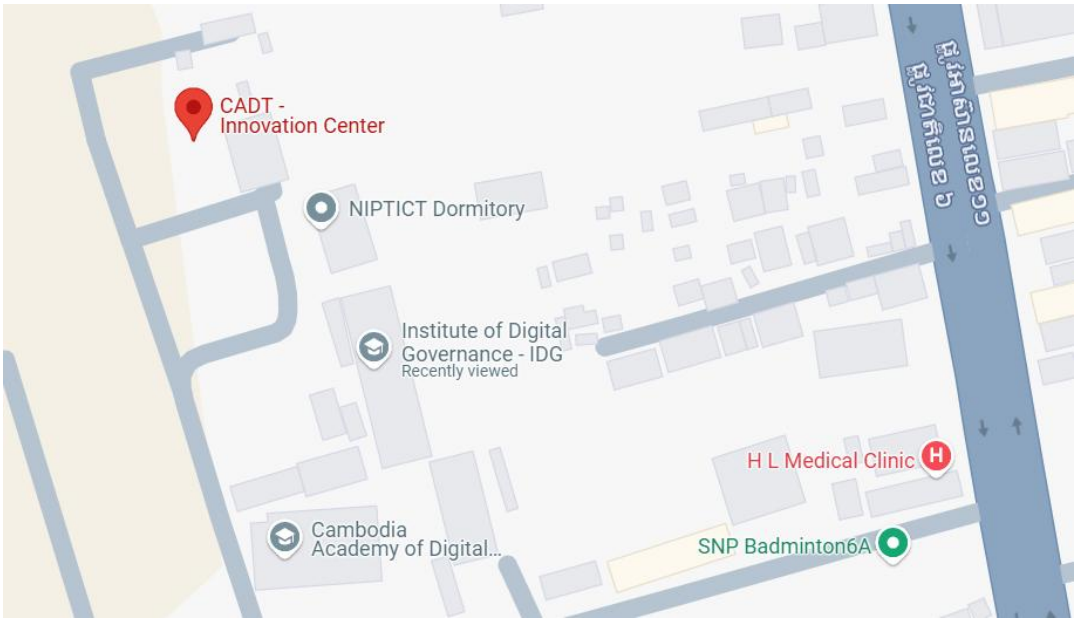

Figure 2: Institute of Digital Research & Innovation (IDRI) Address

- **Website:** https://idri.edu.kh/
- **Phone:** +855 10 344 040

II. PRESENTATION OF THE PROJECT

This chapter provides a detailed overview of the research project, beginning with its initial conception and objectives. It will then explain the necessary pivot in project scope due to real-world constraints and outline the revised, final objectives. Finally, it will present the project plan and development methodology.

2.1. General Presentation of Project

As Cambodia moves toward digitizing official documents, birth certificates present unique challenges due to their combination of printed and handwritten Khmer text. While full digitization is the long-term goal, this project focuses specifically on the accurate extraction of textual information from these certificates. Khmer OCR is particularly difficult due to the script's complexity including numerous consonants, vowels, diacritics, and stacked character forms and is further complicated by limited datasets and inconsistent handwriting. Existing OCR tools often fail on such semi-structured, noisy documents.

This work aims to develop a specialized system for reliably extracting both printed and handwritten Khmer text (and occasional Latin script) from scanned birth certificates. The project, titled **"Automating the Digitization of Printed and Handwritten Khmer Documents Using OCR,"** addresses the significant challenge of extracting structured information from scanned Cambodian birth certificates. These documents are semi-structured and frequently contain a complex mix of printed template text and handwritten entries, making automated data extraction difficult for standard OCR tools.

The original objective was to develop a comprehensive, end-to-end deep learning pipeline that would first localize key information fields (e.g., Name, Date of Birth) using an object detection model such as YOLOv8 and then recognize the text within those fields using an advanced sequence-to-sequence model like TrOCR. The goal was to convert scanned paper documents into an accessible, structured, and machine-readable format, thereby streamlining administrative processes and improving data accessibility for government agencies and citizens alike.

This OCR system forms the machine learning backbone of a broader web-based application designed by the entire project team. While the overall system includes components such as a user interface and backend infrastructure to manage scanned documents and extracted data, my primary responsibility in this project is the design, training, and evaluation of the OCR pipeline specifically, handling the machine learning aspects that enable accurate text detection and recognition within complex Khmer birth certificates.

2.2. Problematic

As Cambodia accelerates its digital transformation initiatives, the digitization of vast archives of official documents, particularly birth certificates, presents a significant and complex challenge. These documents are fundamental for civil registration and public administration, yet they exist primarily in paper format, making data management, retrieval, and analysis inefficient and prone to error.

The core problem lies in the semi-structured nature of these documents and the specific complexities of the Khmer script:

- **Mixed-Text Content:** Cambodian birth certificates are not uniform; they contain a combination of machine-printed template fields and handwritten entries. Standard Optical Character Recognition (OCR) systems are often optimized for one type of text and struggle to process this mixed content accurately. For example, an OCR engine proficient with printed fonts will typically fail when encountering handwritten names, dates, or locations.
- **Complexity of Khmer Script:** The Khmer script is inherently complex for automated recognition. It features a large set of characters, including numerous consonants, vowels, and diacritics (sub-script and super-script vowels, consonant shifters) that can stack vertically. This non-linear character arrangement poses a significant hurdle for traditional OCR systems that are designed for linear scripts like Latin.
- **Data Scarcity and Inconsistency:** There is a lack of large, publicly available, and accurately annotated datasets for Khmer document OCR. Furthermore, the handwriting on existing documents varies widely in style, quality, and clarity, making it difficult to train a single, robust recognition model.

These factors combine to create a critical bottleneck in the national push for digitalization. Manually transcribing these documents is a slow, costly, and labor-intensive process. Therefore, there is a clear and urgent need for a specialized, automated system capable of intelligently and accurately digitizing these complex, mixed-text Khmer documents.

2.3. Objectives

To answer the problems outlined above, this project aims to develop a specialized and modular OCR pipeline capable of accurately digitizing semi-structured Khmer documents. The system is designed as a proof-of-concept to demonstrate a viable solution for automated data extraction from Cambodian birth certificates

Key objectives include:

:

- **Accurate Layout and Field Detection:** To utilize the pre-trained models within the **PaddleOCR** framework to accurately identify and localize high-level document regions, such as text blocks and tables, on scanned birth certificates.
- **Semantic Text-Type Classification:** To develop and train a custom text classifier (**MobileNetV3)** which is a Deep learning Model capable of distinguishing between snippets of printed_text and handwritten_text. This was the core innovation to overcome the limitations of a single OCR tool.
- **Targeted Text Recognition:** To integrate a reliable OCR engine (**Tesseract OCR**) that could be selectively applied to regions classified as "printed," leveraging its strength in handling clear, machine-printed text.
- **Pipeline Integration:** To combine these components into a single, coherent pipeline that ingests a raw document image and outputs a structured JSON file containing the detected layout, text classifications, and extracted text.

2.4. Planning of Project

The project was managed using a structured, phased methodology inspired by Agile principles to accommodate the research-oriented nature of this work. This approach provided the necessary flexibility to adapt to technical findings and challenges throughout the development lifecycle.

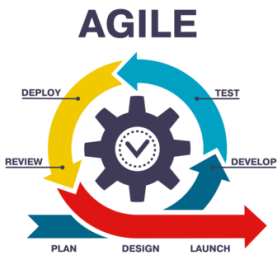

Figure 3: Agile Methodology

The project began with a comprehensive plan based on initial research into state-of-the-art OCR technologies. **2.4.1 Original Project Plan** details the original set of planned activities, which outlined an end-to-end deep learning pipeline.

2.4.1 Original Project Plan

The project was initially conceived with a state-of-the-art, end-to-end deep learning approach. The plan, summarized in **Table 2**, was to leverage a powerful object detector for localization and a transformer-based model for recognition.

Table 2: Original Planned Project Phases (YOLOv8 + TrOCR Concept)

Phase	Name	Key Objectives
Phase 1	Foundational Research & Data Strategy	• Conduct a literature review on modern OCR models (YOLOv8, TrOCR). • Define a strategy for collecting and synthesizing a diverse dataset of Khmer documents.
Phase 2	Field Localization	• Annotate key fields on Cambodian birth certificates. • Train and fine-tune a YOLOv8 model to accurately detect the location of these fields.
Phase 3	Text Recognition	• Annotate a corpus of Khmer text for training a recognition model. • Train and fine-tune a TrOCR model to read the text within the fields detected by YOLOv8.
Phase 4	Pipeline Integration & Evaluation	• Combine the YOLOv8 and TrOCR models into a single, automated pipeline. • Evaluate the end-to-end performance using standard OCR metrics (mAP, CER, WER).

Phase	Name	Key Objectives
Phase 5	Deployment & Reporting	• Develop a simple web application to demonstrate the pipeline's functionality. • Document all findings, methodologies, and results in a final report.

While this initial plan served as a strong starting point, the project's execution had to be adapted to address significant hardware and data constraints identified during the initial phase. The actual project execution therefore involved a strategic pivot to a more modular and resource-efficient pipeline.

2.4.2. Revised and Implemented Plan

After the initial research phase, it became clear that the original plan required computational resources that were beyond the available resources for the internship. The dataset was not collected as planned, the computing resources needed were massively more than expected and the planned scope required more time than the internship constraints allowed. Therefore, a strategic pivot was made to a more modular and resource-efficient pipeline.

This new plan, detailed in **Table 3**, leverages powerful pre-trained models and introduces a custom-built classifier to achieve the project's goals in a more practical and innovative way. The focus shifted from a single end-to-end model to an intelligent, multi-stage system.

Table 3: Revised and Implemented Project Plan (PaddleOCR Pipeline)

Phase	Name	Key Objectives & Technologies
Stage 1	Layout & Table Analysis	• Utilize the pre-trained PaddleOCR framework to analyze the document's physical structure. • Employ layout and table detection models to identify text regions and cells without custom training.
Stage 2	Semantic Text-Type Classification	• Develop a custom dataset by annotating text snippets as "printed" or "handwritten" using PPOCRLabel. • Train a lightweight MobileNetV3 classifier on the Lightning AI cloud platform to accurately distinguish between the two text types.
Stage 3	Targeted OCR & Data Assembly	• Apply Tesseract OCR selectively to the regions classified as "printed" for high-accuracy text extraction. • Combine the layout information, classification labels, and recognized text into a single, structured JSON output.
Stage 4	Evaluation & Reporting	• Evaluate the performance of the key components, particularly the text classifier's accuracy.

Phase	Name	Key Objectives & Technologies
		• Document the new pipeline's architecture, implementation, and results in the final report.

2.4.3. Project Execution Timeline

The execution of this project, from the initial research based on the original plan to the final implementation of the revised pipeline, was completed over a three-month period. The Gantt chart in **Table 4** provides a visual summary of the actual project timeline. It highlights the key phases of work on a weekly basis, clearly illustrating the strategic pivot that occurred at the end of May. This transition from a purely research-focused stage to an adaptive development cycle demonstrates the project's agile and iterative nature, which was crucial for successfully navigating the technical challenges and delivering a functional proof-of-concept within the internship's timeframe

Table 4: Actual Project Execution Timeline

Task / Week	May 1	May 2	May 3	May 4	Jun 5	Jun 6	Jun 7	Jun 8	Jul 9	Jul 10	Jul 11	Jul 12
Phase 1: Research & Initial Design												
Phase 2: Project Pivot & Prototyping												
Phase 3: Text Classifier Development												
Phase 4: Integration & Reporting												

III. LITERATURE REVIEW

This chapter reviews existing literature and technologies relevant to Optical Character Recognition (OCR), surveying the evolution of methodologies from classical computer vision techniques to modern deep learning architectures. A particular focus is placed on the challenges associated with low-resource and graphically complex scripts like Khmer, as these challenges directly informed the design choices and strategic decisions made during this project. The review will cover foundational algorithms, state-of-the-art models for text detection and recognition, and the efficient architecture that enabled the final pipeline's implementation.

3.1. Foundational OCR Methodologies

Before the widespread adoption of deep learning, optical character recognition (OCR) was a field dominated by multi-stage pipelines that relied on classical image processing and pattern recognition techniques. These foundational methods are essential to review, as they establish the core challenges of character segmentation and classification that modern architectures aim to solve more effectively. This section will explore the cornerstone algorithms that formed the basis of early OCR systems, highlighting both their contributions and the limitations that motivated the shift towards more advanced, learning-based models.

3.1.1. Image Preprocessing and Segmentation

A critical prerequisite for any recognition task is the isolation of a character from its background and from other characters. This was typically achieved through binarization (converting to black and white) and segmentation (dividing into separate parts).

Edge Detection: The process often began with identifying the boundaries of characters. As analyzed by Vijayarani and Vinupriya (2013), edge detection algorithms are fundamental for this task. They compare two primary methods: the Sobel operator, which computes a 2D spatial gradient on an image to emphasize regions of high frequency, and the Canny edge detector. The Canny algorithm is a more sophisticated, multi-stage method that includes noise reduction with a Gaussian filter, gradient calculation, non-maximum suppression to thin edges, and double thresholding with hysteresis to eliminate weak edges. The study concludes that the Canny method, while more computationally intensive, generally produces superior, cleaner, and more continuous edges, which is a critical requirement for accurate character segmentation. [1]

Connected-Component Labeling (CCL): Once an image is binarized, CCL is a vital algorithm for grouping pixels into distinct objects. In their comprehensive survey, He et al.

(2017) describe CCL as an algorithm that scans a binary image and assigns a unique label to each cluster of connected foreground pixels. This transforms a sea of pixels into a set of discrete, countable objects, as illustrated in **Figure 4**.

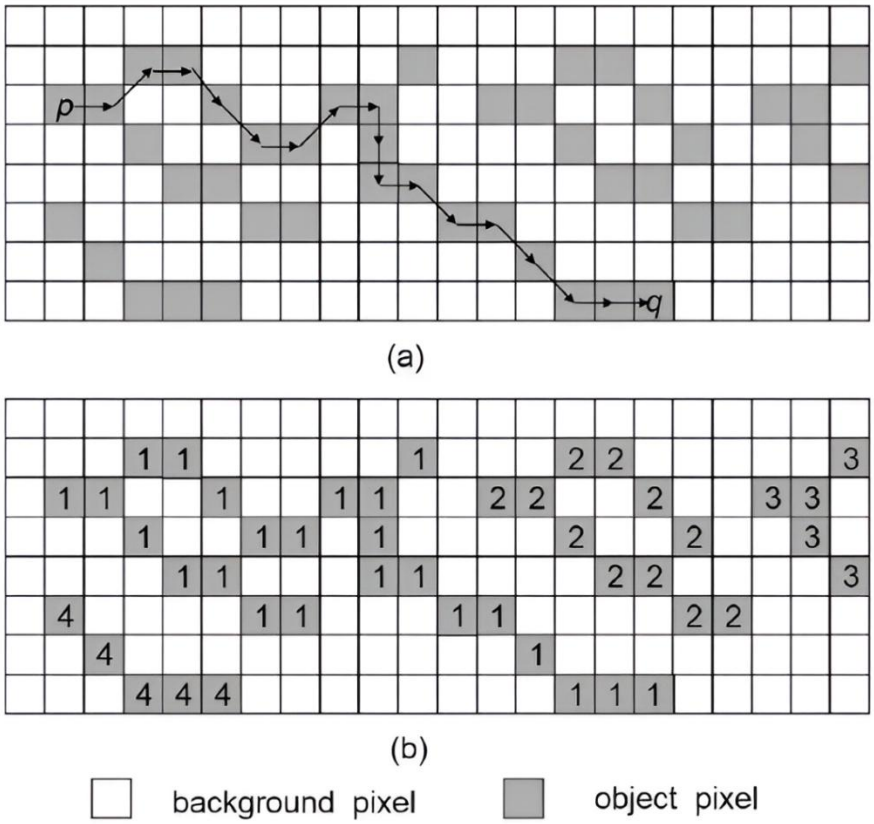

Figure 4: An illustration showing how distinct groups of foreground pixels (a) are assigned unique labels to identify them as separate objects (b) in the Connected-Component Labeling process. (From He et al. [2]).

While foundational, the authors note that CCL's performance is entirely dependent on the quality of the input binary image. For a complex script like Khmer, which contains numerous detached diacritics and vertically stacked components, a simple global threshold for binarization can easily cause a single character to be fragmented into multiple components. This "over-segmentation" leads to catastrophic failures in the subsequent recognition stage, highlighting the need for more context-aware segmentation methods [2].

3.1.2. Classical Feature Extraction and Classification

After segmenting characters, traditional systems relied on handcrafted features to represent and classify them.

Feature descriptors: A variety of techniques have been explored for feature engineering. Chey et al. (2005) proposed a method for Khmer using wavelet descriptors. This involves first skeletonizing the character to create a normalized, one-pixel-thick representation,

and then applying a wavelet transform to capture frequency components at different scales. The resulting coefficients form a feature vector that describes the character's shape [3]. While innovative, this approach is sensitive to variations in stroke thickness and style.

A more robust and widely adopted feature descriptor is the **Histogram of Oriented Gradients (HOG)**, which was analyzed in the context of text recognition by Tian et al. (2013). The HOG method involves:

1. Dividing the image into small, connected regions called cells.
2. Compiling a histogram of gradient directions for the pixels within each cell.
3. Concatenating these histograms to form a single feature vector. This method proved highly effective due to its robustness to local geometric and illumination changes [4].

Classification with Support Vector Machines (SVM): The feature vectors extracted from these methods were then fed into a classifier. As demonstrated by Sok and Taing (2014) for Khmer OCR, the Support Vector Machine (SVM) was a powerful choice for this task [5]. An SVM is a supervised learning model that works by finding an optimal hyperplane that best separates the data points of different classes in a high-dimensional space. The "optimal" hyperplane is the one that maximizes the margin, the distance between the hyperplane and the nearest data points from each class. This concept is illustrated in **Figure 5**.

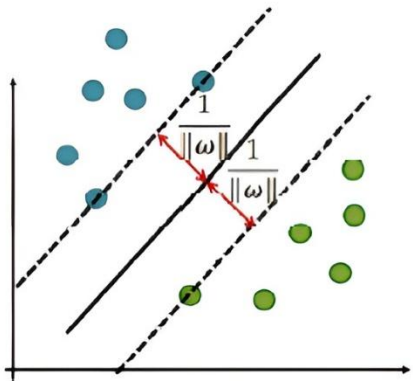

Figure 5: An illustration of an optimal hyperplane in an SVM, which maximizes the margin between the two classes. (From Sok & Taing [5]).

Mathematically, for a simple linear case, the SVM solves the following optimization problem:

$$min_{w,b}\frac{1}{2}\|\vec{w}\|^2 subject\ to\ y_1(\vec{w}\vec{x}_i - b) \geq 1, \forall i \quad (1)$$

- where x_i are the feature vectors, y_i are the class labels, and the goal is to find the weight vector w and bias b that define the hyperplane. The key limitation of this entire approach is that its success is fundamentally bottlenecked by the quality and

expressiveness of the handcrafted features (HOG, Wavelets, etc.). This separation of feature extraction and classification is what deep learning aims to unify [5].

3.2. Modern Deep Learning Pipelines for OCR

The limitations of classical OCR methods, particularly their reliance on handcrafted features and sensitivity to variations in noise and style, catalyzed a paradigm shift in the field toward deep learning. As surveyed by Long et al. (2020), modern approaches are characterized by the use of deep neural networks, which learn hierarchical feature representations directly from pixel data, eliminating the need for manual feature engineering [6].

These modern systems are typically structured as a two-stage pipeline, a concept thoroughly reviewed by Ye and Doermann (2015) in their comprehensive survey of text recognition in imagery [7]. This standard pipeline, illustrated in **Figure 6**, consists of:

1. **Text Detection**: This initial stage localizes text within an image. The goal is to produce bounding boxes that tightly enclose every line or word of text. This is essentially an object detection task.
2. **Text Recognition**: The image patches cropped from the detected bounding boxes are then passed to a second model. This stage transcribes the sequence of pixels within the patch into a sequence of characters. This is a sequence recognition task.

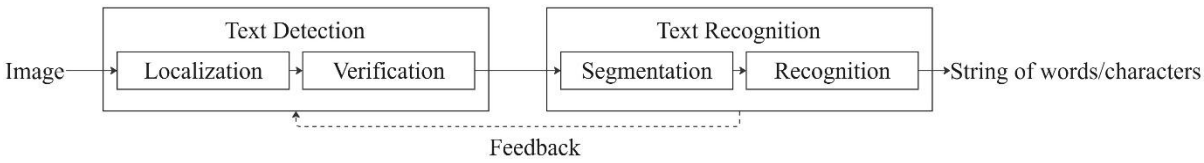

Figure 6: The standard two-stage pipeline for modern OCR systems, separating the tasks of text detection and text recognition. (Adapted from Ye & Doermann, 2015)

This modular, two-stage approach allows for specialized models to be developed for each sub-problem. The following sections will review the state-of-the-art architectures for both text detection and text recognition, which informed the design choices, and the eventual pivot made in this project.

3.3. Scene Text Detection

The first stage in a modern OCR pipeline is text detection, an object detection task focused on localizing text within an image. The goal is to produce precise bounding boxes around words or text lines, which can then be passed to a recognition model. The literature contains a variety of approaches, but recent state-of-the-art methods are dominated by deep learning-based, single-shot detectors that offer a strong balance of speed and accuracy.

3.3.1. Regression-Based Detectors (YOLO and EAST)

Unlike older, multi-stage detectors (like the R-CNN family), single-shot detectors treat text detection as a regression problem, predicting bounding box coordinates and class confidences directly from feature maps in a single pass. The YOLO (You Only Look Once) family of models, particularly later versions like YOLOv4 and beyond, exemplifies this approach. As described by Wang et al. (2021) in their work on R-YOLO, the architecture uses a powerful CNN backbone (like CSPDarknet53) to extract features at multiple scales and directly regresses the bounding box parameters [8]. A similar and highly influential model is EAST (An Efficient and Accurate Scene Text Detector), which, as proposed by Zhou et al. (2017), uses a Fully Convolutional Network (FCN) to directly predict word or text-line bounding boxes of arbitrary orientations from the full image, eliminating intermediate steps and enabling real-time performance. It can directly predict not just standard axis-aligned boxes but also rotated rectangles (RBOX) or even arbitrarily shaped quadrilaterals (QUAD) to better capture the geometry of scene text [9]. This is illustrated in **Figure 7**

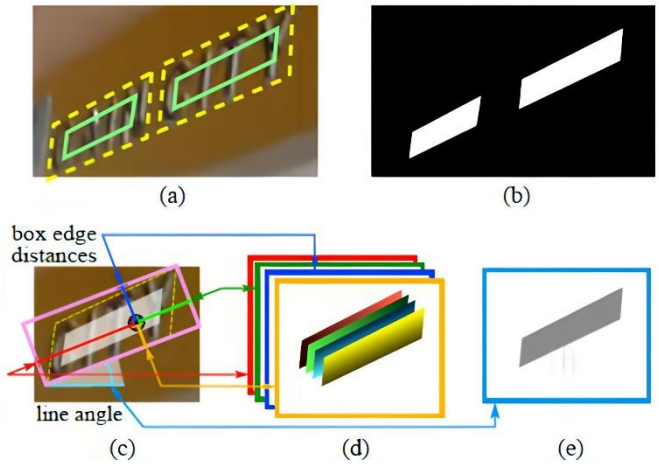

Figure 7: The label generation process for the EAST detector. (From Zhou et al. [9]).

The primary strength of these models is their high efficiency and accuracy. By eliminating intermediate steps, they can perform detection in real-time. This powerful, flexible approach to localization was the foundation for the detection component of this internship's initial project plan, which intended to use YOLOv8 to find key fields in the birth certificates.

3.3.2. Layout-Specific Models (PaddleOCR)

While general object detectors are powerful, the field has also seen the rise of comprehensive, open-source, multi-purpose toolkits specifically designed for document analysis. PaddleOCR is a leading example, providing a suite of pre-trained models that go beyond simple text detection to include full document parsing capabilities [10]. As detailed in

their technical report, the system is modular and includes components for layout analysis, table recognition, and key information extraction, making it an end-to-end document understanding solution.

The primary strength of the PaddleOCR ecosystem is its modularity and the availability of high-quality, pre-trained models for a wide range of document-related tasks. It offers specialized models like PP-DocLayout for identifying the logical structure of a page and PP-StructureV3 for parsing tables and figures. This "toolkit" approach allows developers to select the precise components they need without having to build everything from scratch. **Figure 8** illustrates the evolution and the rich set of capabilities within the PaddleOCR framework.

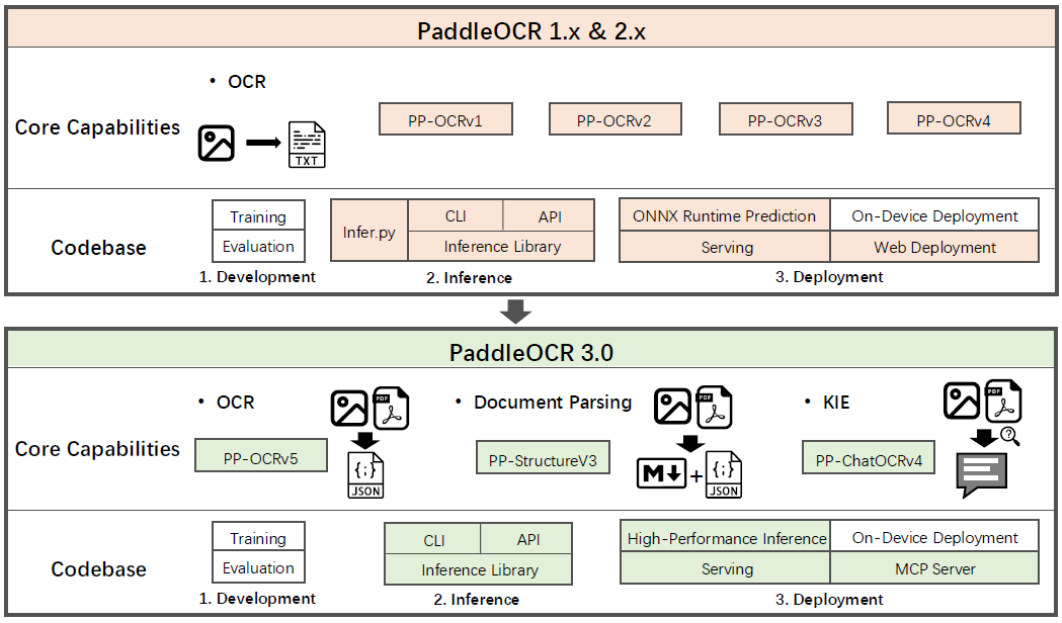

Figure 8: The evolution and core capabilities of the PaddleOCR framework, highlighting its modular components for OCR, Document Parsing, and Key Information Extraction (KIE). (Adapted from Cui et al., 2025).

This pre-trained, modular approach directly addresses the data scarcity and resource limitations faced in this project. Instead of training a general-purpose detector like YOLOv8 on a small, custom dataset, we were able to leverage PaddleOCR's robust, pre-trained layout and table detection models. This strategic choice provided a highly accurate detection front-end for our pipeline, allowing the project's development efforts to be focused on the novel text-type classification task.

3.4: Scene Text Recognition

Once text regions have been detected and localized, the second stage of the pipeline involves transcribing the pixel data within those regions into a character sequence. This is a sequence recognition task where the complexity varies greatly depending on the model

architecture. This section reviews the key architecture, from established baselines to the state-of-the-art models that informed this project.

3.4.1. CNN-RNN Architectures

A seminal and highly influential architecture for text recognition is the Convolutional Recurrent Neural Network (CRNN), by Shi, Bai, and Yao (2017) in their paper, " An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition " [11]. The CRNN architecture, illustrated in **Figure 9**, consists of three main parts:

1. **Convolutional Layers:** A deep CNN is used as a feature extractor to convert the input image into a sequence of feature vectors.
2. **Recurrent Layers:** A deep Bidirectional LSTM (Bi-LSTM) network is built on top of the CNN to model the contextual dependencies within the feature sequence.
3. **Transcription Layer:** A Connectionist Temporal Classification (CTC) loss layer is used to translate the per-frame predictions from the RNN into a final label sequence, without requiring precise alignment between the input frames and output characters.

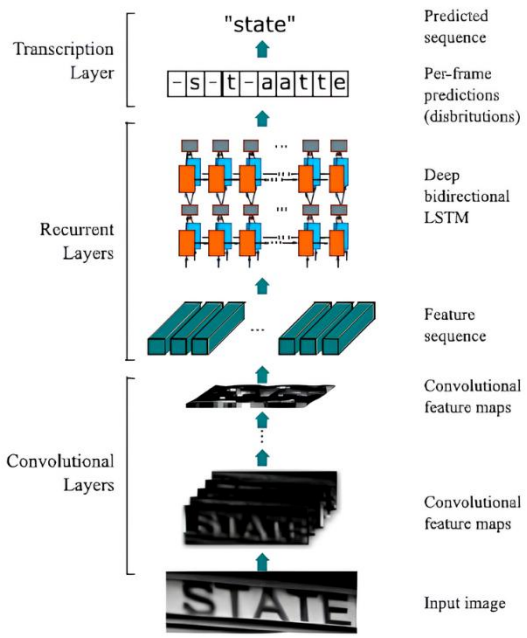

Figure 9: The architecture of the CRNN model, integrating convolutional layers for feature extraction, recurrent layers for sequence modeling, and a transcription layer (Adapted from Shi, Bai, and Yao (2017)).

The CRNN model became a dominant baseline because it is end-to-end trainable and can naturally handle sequences of arbitrary length. Its architecture elegantly combines the strong visual feature extraction of CNNs with the powerful sequence modeling capabilities

of RNNs, making it highly effective for text recognition. A similar Gated Recurrent Convolutional Neural Network (GRCNN) was later proposed by Wang and Hu (2017), further validating the power of combining convolutional and recurrent components [12]. The CRNN approach serves as a critical benchmark against which newer models are often compared.

3.4.2. Transformer-Based Architectures (TrOCR)

More recently, architectures based on the Transformer model, first introduced by Vaswani et al. (2017) in "Attention Is All You Need" [13], as shown in **Figure 10**, have pushed the state of the art. The Vision Transformer (ViT) first adapted this for computer vision by treating an image as a sequence of patches [14]. Building on this, Li et al. (2022) proposed TrOCR, a pure encoder-decoder Transformer model specifically for OCR [15]. It uses a ViT to encode an image patch and a pre-trained text Transformer decoder (like BERT [16]) to autoregressively generate the output text.

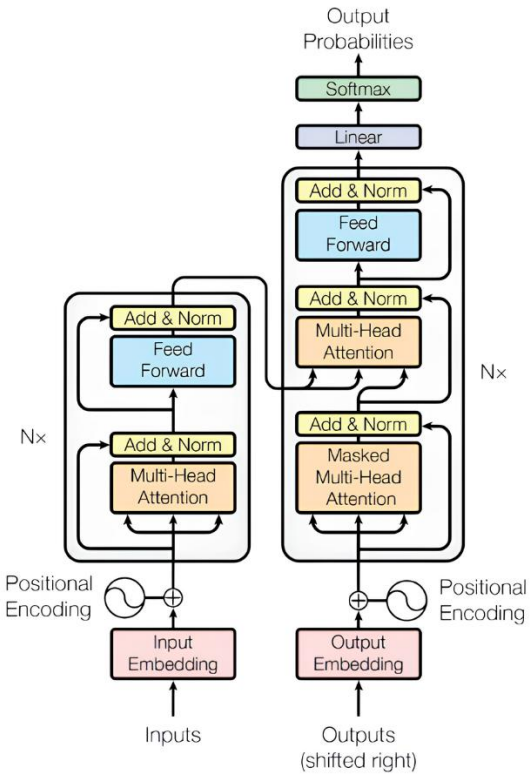

Figure 10: The Transformer - model architecture (From Vaswani et al. [13])

TrOCR's main contribution is its convolution-free, end-to-end architecture that jointly leverages massive, pre-trained models from both the vision and language domains. This allows it to achieve outstanding performance on both printed and handwritten text without complex

post-processing steps. Buoy et al. (2023) explored a similar convolutional Transformer-based model specifically for low-resource scripts like Khmer, noting the effectiveness of the Transformer architecture but also the challenges of data scarcity [17].

The primary drawback of TrOCR is its significant computational cost and its dependency on massive pre-training datasets. As highlighted in the original paper, fine-tuning requires powerful GPUs, and its performance on low-resource scripts without extensive, domain-specific pre-training can be suboptimal. This high resource requirement was the direct reason this project pivoted to a more lightweight and practical solution.

3.4.3. Practical Toolkits and Efficient Models

While state-of-the-art models like TrOCR demonstrate immense power, their resource requirements often make them impractical for projects with limited data or computational budgets. This has led to the development of comprehensive toolkits and lightweight architectures that prioritize efficiency and accessibility.

The PaddleOCR Ecosystem: PaddleOCR is a powerful, open-source OCR toolkit developed by Baidu. As detailed in their PaddleOCR 3.0 Technical Report, it is more than a single model; it is a full ecosystem of tools designed for end-to-end document understanding. The framework includes a suite of highly efficient, pre-trained models for layout detection **(PP-DocLayout)**, table recognition **(PP-StructureV3)**, and multilingual text recognition **(PP-OCRv5)** [10]. The core strength of PaddleOCR is its modularity and the quality of its pre-trained models. It provides developers with ready-to-use components that can be integrated into custom pipelines. This allows for a practical "best-of-breed" approach, where a developer can leverage a powerful pre-trained model for one part of a task (like **layout detection**) while focusing their own development efforts on another.

The PaddleOCR framework was the cornerstone of this project's successful pivot. By utilizing its pre-trained PP-DocLayout_plus-L and RT-DETR-L_wired_table_cell_det models, we were able to achieve robust layout and table detection on Cambodian birth certificates without needing to create a massive, manually annotated dataset. This strategic decision enabled the project to focus its limited resources on solving the more nuanced problem of classifying handwritten versus printed text.

MobileNetV3 for Lightweight Classification: For the custom text-type classification sub-task, a highly efficient model was necessary. MobileNetV3, introduced by Howard et al. (2019), is a neural network architecture specifically designed for high performance on resource-constrained devices [18]. It achieves this efficiency through architectural innovations

like the use of an inverted residual structure, linear bottlenecks, and a novel h-swish activation function, all optimized via Neural Architecture Search (NAS).

MobileNetV3's defining feature is its exceptional balance of high accuracy and low computational cost. It is a premier choice for tasks that require a small model footprint and fast inference without a significant sacrifice in performance. Relevance to Project: MobileNetV3 was the ideal architecture for the custom-built classifier in our pipeline. Its lightweight design allowed rapid training on a small dataset (even on a cloud GPU like the NVIDIA T4) and resulted in a highly accurate model (99.1% validation accuracy) for distinguishing between printed and handwritten text. This choice exemplifies the project's overall philosophy of building a powerful system from efficient, practical components.

3.4.4. Tesseract: A Foundational and Practical OCR Engine

The introduction of the **LSTM engine** in **Tesseract 4.0** marked a significant leap in its core recognition capabilities. However, its performance remains highly sensitive to the quality of the input image, especially on "unfriendly" or noisy samples captured in real-world conditions. Research has therefore focused not just on the engine itself, but on intelligent preprocessing pipelines that optimize images specifically for Tesseract.

A notable study by Sporici et al. [19] addresses this challenge directly. They demonstrate that while Tesseract 4.0 performs well on clean inputs, its accuracy degrades sharply on challenging images, sometimes failing to recognize any text at all. Their solution was to develop an adaptive preprocessing step that uses a reinforcement learning model to find the optimal **convolution-based filters** to apply to an image *before* it is sent to Tesseract. This method effectively learns to transform an image in a way that is "Tesseract-friendly," even if the resulting image looks unnatural to the human eye.

The results of their approach were dramatic. On a challenging dataset, their convolution-based preprocessing method boosted the character-level accuracy of Tesseract 4.0 from 0.134 to 0.616 (a +359% relative increase) and the F1 score from 0.163 to 0.729 (a +347% relative increase).

This research underscores the critical importance of the preprocessing stage in any modern OCR pipeline. It proves that even a state-of-the-art engine like Tesseract is not a standalone solution, but one component in a larger system. This paper validates our project's focus on creating a robust, multi-stage workflow where input images are carefully prepared and normalized before being passed to a recognition engine. It confirms that intelligent preprocessing is a key factor in achieving high accuracy in real-world applications.

It is worth noting that the current stable version, **Tesseract 5**, uses the same underlying LSTM architecture as version 4 but with significant performance improvements and a more modular training system. The findings on the importance of preprocessing remain equally, if not more, relevant for the latest versions of the engine.

IV. PROJECT ANALYSIS AND CONCEPTS

This chapter details the foundational analysis of the project, outlining the system's core functionalities and requirements. The analysis is derived from the primary problems and objectives established in Chapter II, focusing on what the system must achieve, and the quality attributes it must possess. Furthermore, this chapter specifies the data requirements and presents a high-level conceptual workflow of the implemented OCR pipeline.

4.1. System Requirements

A thorough requirements analysis was conducted to ensure the final system would effectively address the challenge of digitizing mixed-text Khmer documents in a practical and resource-efficient manner. These requirements are divided into functional and non-functional categories.

4.1.1. Functional Requirements

Functional requirements define the specific behaviors and actions the system must perform. They represent the core deliverables of the project as describe in **Table 5**.

Table 5: Functional Requirements

ID	Requirement	Description
FR-1	Image Ingestion	The system must be able to accept a scanned document image as input in common formats (e.g., JPEG, PNG, BMP).
FR-2	Image Preprocessing	The system must perform a series of automated preprocessing steps to normalize the input image, including Grayscale Conversion, Noise Reduction, and Skew Correction.
FR-3	Layout Analysis	The system must analyze the document's physical structure to accurately detect the bounding boxes of high-level layout elements, including text blocks and tables.
FR-4	Text-Type Classification	For each detected text block, the system must classify its content as either printed or handwritten. This is a critical function for enabling selective downstream processing.

ID	Requirement	Description
FR-5	Targeted OCR	The system must apply an OCR engine selectively to the text regions based on their classification label (e.g., applying Tesseract primarily to printed text).
FR-6	Structured Data Output	The system must produce a machine-readable, structured output (e.g., a JSON file) that consolidates the results, including bounding box coordinates, text-type labels, and the transcribed text.
FR-7	Key Field Identification	The system's layout analysis and post-processing must be capable of identifying and labeling the following 11 predefined key fields from a Cambodian birth certificate:

Table 6: Predefined Key Information Fields for Extraction

No	Field Name	Description
1	first_name	First name of the person as written on the birth certificate.
2	last_name	Last name of the person as written on the birth certificate.
3	gender	Gender of the person (e.g., Male, Female).
4	first_name_en (Optional)	First name of the person written in English, if available.
5	last_name_en (Optional)	Last name of the person written in English, if available.
6	nationality	Nationality of the person (e.g., Cambodian).
7	date_of_birth	Full date of birth including day, month, and year.
8	place_of_birth	Detailed place of birth including district, commune, city, and country.
9	parent_full_name_father	Full name of the father as written in Khmer.
10	parent_full_name_mother	Full name of the mother as written in Khmer.
13	parent_nationality_father	Nationality of the father.
14	parent_nationality_mother	Nationality of the mother.

No	Field Name	Description
15	parent_date_of_birth_father	Date of birth of the father (day, month, year).
16	parent_date_of_birth_mother	Date of birth of the mother (day, month, year).
17	parent_place_of_birth_father	Place of birth of the father (district, commune, city, country, etc.).
18	parent_place_of_birth_mother	Place of birth of the mother (district, commune, city, country, etc.).

4.1.2. Non-Functional Requirements

Non-functional requirements define the quality attributes and operational characteristics of the system. They are crucial for ensuring the solution is not just functional but also robust and practical for real-world use.

Table 7: Non-Functional Requirements

ID	Requirement	Description & Justification
FR-1	Accuracy	The text-type classifier, a core custom component, must achieve a high Top-1 validation accuracy (target >95%) to ensure reliable pipeline performance. The end-to-end transcription should demonstrate a significant improvement over a naive baseline.
FR-2	Efficiency	The pipeline must be computationally efficient, avoiding the high resource demands of training large-scale models from scratch. This justifies the use of pre-trained components (PaddleOCR) and a lightweight custom classifier (MobileNetV3).
FR-3	Modularity	Each primary stage of the pipeline (Layout Analysis, Text Classification, Recognition) must be implemented as a distinct module. This design allows for individual components to be updated or replaced in the future without redesigning the entire system (e.g., swapping the handwritten OCR model).
FR-4	Robustness	The system should be robust to variations in input image quality, such as minor noise and lighting differences, which is addressed through the preprocessing and data augmentation stages.

4.2. Data Specification and Analysis

The success of any machine learning project is fundamentally dependent on the quality and characteristics of the data used. This project involved handling existing documents and creating a new, specialized dataset for the classification task.

4.2.1. Source Documents: Cambodian Birth Certificates

The primary target documents for this project were scanned Cambodian birth certificates. These documents present a significant OCR challenge due to their semi-structured layout and mixed-text nature, often containing both standardized, machine-printed Khmer fields and highly variable, handwritten entries as shown in **Figure 11**.

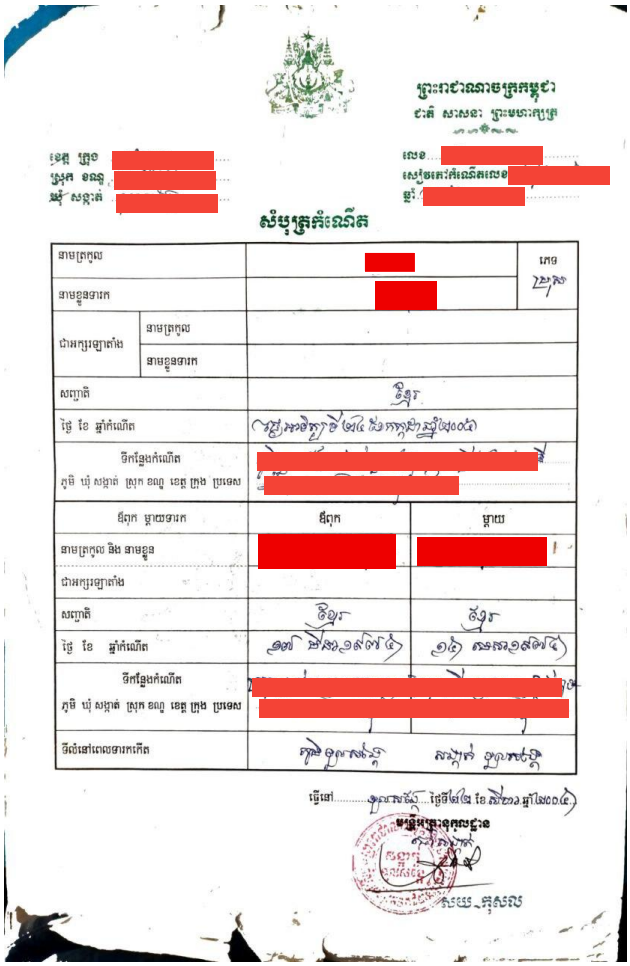

Figure 11: A Sample of Birth Certificate

4.2.2. Custom Dataset for Text-Type Classification

To train the core innovative component of the pipeline, a text-type classifier a custom dataset was created.

- Data Source: A collection of approximately 20 unique Cambodian birth certificates was used as the source for generating training and validation samples as it is only for training prototype only and not meant for real production testing.
- Annotation Process: The PPOCRLabel tool was utilized to manually crop small image snippets from the source documents. Each snippet was then manually and unambiguously labeled as either printed_text or handwritten_text.
- Final Dataset Composition: This process resulted in a dataset of 1,677 labeled image snippets. The dataset was imbalanced, reflecting the nature of the source documents. The final split used for training and validation is summarized in **Table 8**

Table 8: Dataset Splitting for Training and Evaluation for MobileNetV3 Model

Category	Validation Set	Training Set	Total
Printed	191	765	956
Handwritten	144	577	721
Total	335	1,342	1,677

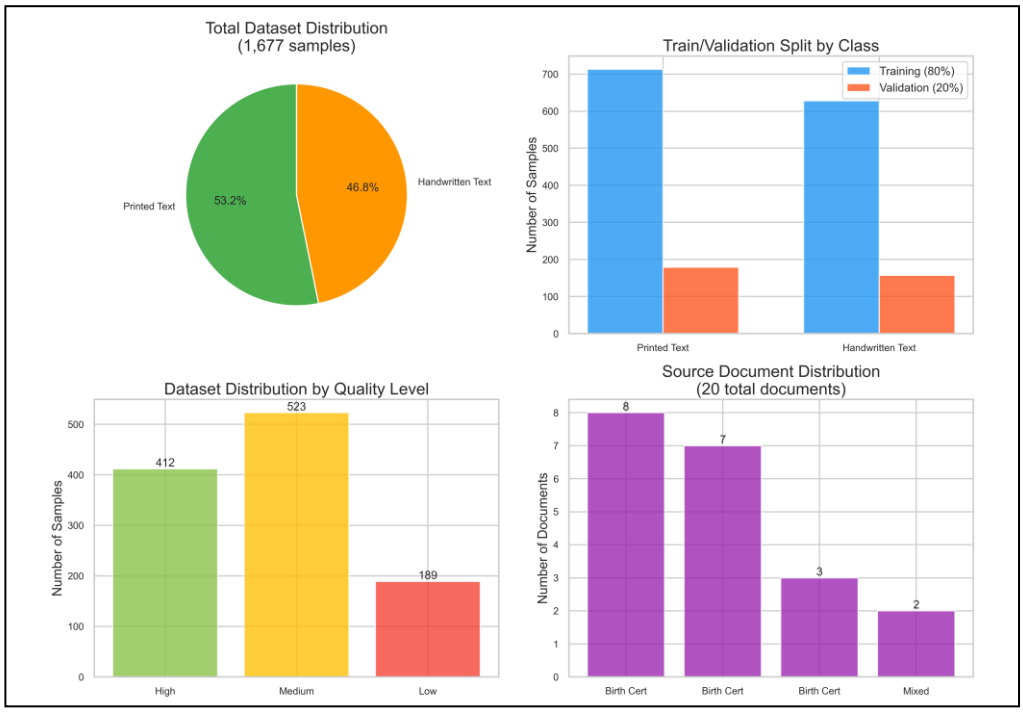

Figure 12: Detailed analysis of the custom dataset used for the text-type classifier, showing class distribution, train/validation split, source document quality, and composition.

As shown in **Figure 12** These charts provide a comprehensive visual breakdown of the custom dataset described in this section. The first chart is for the printed vs handwritten distribution, the second is for splitting validation and training sets, the third is for the quality

of the dataset collected noted that high mean its readable, medium is semi readable while low is too noisy with only a small amount of readable text and the last chart is for different type of Birth Certificate, our collection is not just the main Birth Certificate its also for a Copy version, a Verification version and the Copy of the Verification version.

4.2.3. Data Augmentation Strategy

To address dataset size limitations, the following augmentation techniques were applied:

- **Geometric Transformations**: Rotation (±5°), scaling (90%-110%), translation
- **Photometric Variations**: Brightness adjustment, contrast modification, Gaussian noise
- **Morphological Operations**: Erosion and dilation to simulate printing quality variations

4.3. Conceptual System Workflow

The overall conceptual workflow of the system is designed as a multi-stage pipeline that transforms an unstructured image into structured, analyzable data. The high-level process flow is illustrated in **Figure 13**.

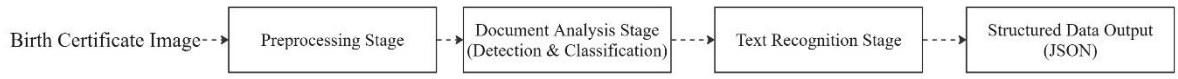

Figure 13: High-Level Conceptual Workflow of the OCR Pipeline

This conceptual model forms the basis for the detailed technical design, which will be elaborated upon in the next chapter.

4.4. Tools and Technologies

To meet the functional and non-functional requirements of the project, a combination of modern machine learning frameworks, libraries, and development tools was selected. This technology stack was chosen to support rapid prototyping, efficient model training, and the development of a robust OCR pipeline.

4.4.1. Key Technologies and Frameworks

The core of the project is built upon state-of-the-art deep learning frameworks that provide the necessary components for computer vision and sequence modeling. The selection of these technologies was driven by the need for accuracy, efficiency, and practicality in a low-resource context.

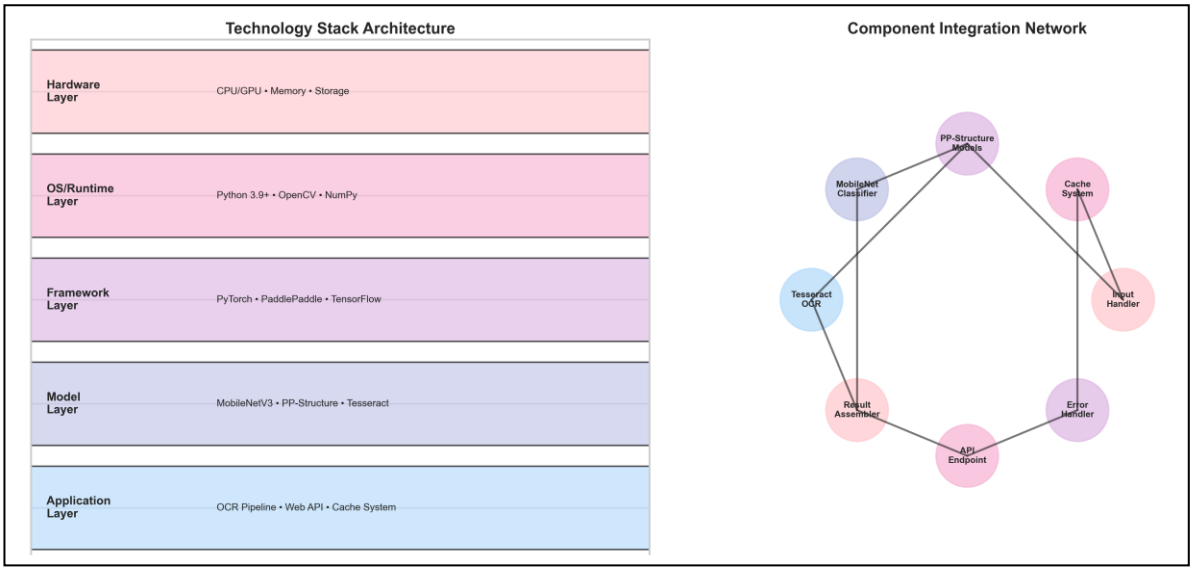

Figure 14: Overview of the project's technology stack, illustrating the layered architecture and the network of integrated software components.

Before diving into the specific models and libraries in **Table 9** and **Table 10**, this **Figure 14** presents a high-level overview of the entire technology stack.

Table 9: Key Models and Technologies

Technology	Specific Model / Component	Role in Project
PaddleOCR (Layout)	PP-DocLayout_plus-L	Used as the primary pre-trained model for Layout Analysis. It identifies the overall structure of the birth certificate, detecting high-level regions like text blocks and titles.
PaddleOCR (Table)	RT-DETR-L_wired_table_cell_det	Used as a specialized pre-trained model for Table Detection. It is applied to regions identified as tables to

Technology	Specific Model / Component	Role in Project
		detect the precise bounding boxes of individual cells.
PaddleClas	MobileNetV3	Served as the lightweight CNN architecture for the custom-trained Text-Type Classifier. Chosen for its excellent balance of accuracy and computational efficiency.
Tesseract OCR	pytesseract with khm language pack	Used as the final Text Recognition engine for regions classified as printed_text, due to its strong support for the Khmer script.
YOLO	YOLOv8	Researched and prototyped during the initial project phase for the task of key field detection.
Transformer OCR	TrOCR	Researched and analyzed during the initial project phase for handwritten text recognition but was ultimately deemed too resource intensive.

4.4.2. Development Environment and Tools

The project was developed and evaluated using a combination of local and cloud-based environments, industry-standard frameworks, and specialized annotation tools. This tool chain was selected to ensure reproducibility, accelerate development, and provide robust evaluation.

Table 10: Languages and Platforms

Category	Tool / Library	Logo	Description
Cloud & Execution Environments	Lightning AI	 *Figure 15: Lightning AI*	A cloud-based MLOps platform used for model training. It was chosen for its on-demand access to powerful hardware (NVIDIA T4 GPUs) and its pre-configured, reproducible development environments.
	Google Colab	 *Figure 16: Google Colab*	Used during the initial research phase for prototyping and experimenting with various models due to its accessible GPU resources.
	Jupyter Notebook	 *Figure 17: Jupyter Notebook*	The primary tool for interactive data exploration, initial model testing, and visualization of results throughout the project's lifecycle.
Core ML Frameworks	PaddlePaddle	 *Figure 18: PaddlePaddle*	The underlying deep learning framework for the PaddleOCR and PaddleClas ecosystems. All final pipeline components were built on this framework.
	PyTorch	 *Figure 19: Pytorch*	These are foundational deep learning frameworks. They were extensively researched and used as the basis for models considered during the initial

Category	Tool / Library	Logo	Description
			project phase (e.g., TrOCR is implemented in PyTorch).
Data Science Libraries	Python	 *Figure 20: Python*	The core programming language for the entire project, used for scripting, data processing, model training, and pipeline integration.
	OpenCV / Pillow	 *Figure 21: OpenCV*	Essential computer vision libraries used for all image manipulation and preprocessing tasks, such as grayscale conversion, binarization, and image cropping.
Annotation Tools	PPOCRLabel	 *Figure 22: PPOCRLabel*	The specialized annotation tool from the PaddleOCR suite. It was the primary tool used to create the final, custom dataset for the text-type classification task by cropping and labeling image snippets.
	LabelMe	 *Figure 23: LabelMe*	For cropping sentence from paragraph or long sentence type of data and get JSON with cropped image for TrOCR training.
	Label Studio	 *Figure 24: Label-Studio*	General-purpose annotation tools that were researched and used during the initial project phase to annotate text for the TrOCR and YOLOv8 models.

Category	Tool / Library	Logo	Description
Experiment Tracking	Weights & Biases	 *Figure 25: Weights & Biases*	An MLOps tool used for experiment tracking and monitoring. It was integrated to log training metrics such as loss and accuracy, and to visualize model performance and resource usage (CPU/GPU).

V. DETAIL CONCEPT

This chapter provides a detailed technical blueprint of the OCR systems conceptualized and implemented during the internship. It begins by outlining the initial state-of-the-art pipeline design and its underlying technologies. It then presents the final, revised pipeline architecture, detailing the strategic shift in design and the specific components chosen to overcome the practical limitations of the initial concept.

5.1. Original Pipeline Concept (YOLOv8 + TrOCR)

The initial project design was based on a powerful, end-to-end deep learning approach that coupled a state-of-the-art object detector with a transformer-based recognition model.

5.1.1. Architecture and Workflow

The envisioned pipeline, illustrated in **Figure 26**, was a two-stage system. First, a YOLOv8 object detection model, pre-trained on the COCO dataset and then fine-tuned on annotated birth certificates, would be used to perform Key Field Detection. This stage would identify and crop the 11 predefined fields (e.g., "First Name", "Date of Birth"). Second, these cropped image patches would be passed to a TrOCR (Transformer-based OCR) model. The TrOCR model, with its Vision Transformer (ViT) encoder and text Transformer decoder, would then transcribe the image content into text.

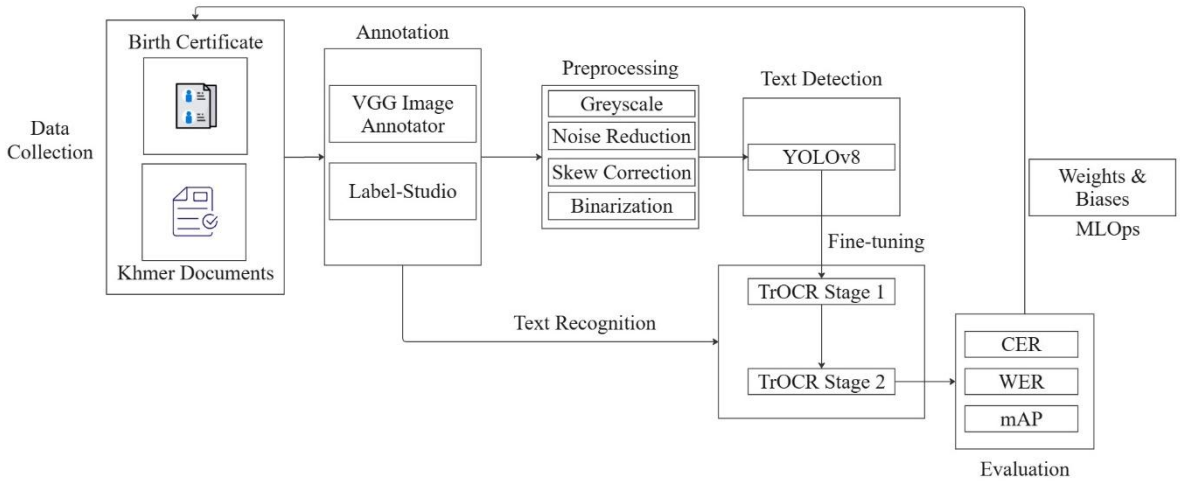

Figure 26: The architecture of the original OCR pipeline concept, based on YOLOv8 for text detection and TrOCR for text recognition.

5.1.2. Rationale and Limitations

This architecture was chosen based on the literature review, as both YOLOv8 and TrOCR represent the state-of-the-art in their respective domains. The goal was to leverage TrOCR's proven strength in recognizing both printed and handwritten text. However, as identified during the initial research phase, this approach proved to be computationally infeasible due to the immense data and GPU resources required to properly pre-train and fine-tune the TrOCR model for a low-resource, complex script like Khmer. This practical limitation necessitated a pivot to a more resource-efficient and modular design.

5.2. Final Implemented Pipeline Architecture (PaddleOCR + MobileNetV3)

To address the limitations of the original concept, a new,more modular pipeline was designed and implemented. This revised architecture prioritizes efficiency by leveraging high-quality pre-trained models and introducing a custom lightweight classifier for a specialized sub-task. The complete architecture of the final implemented pipeline is depicted in **Figure 27.**

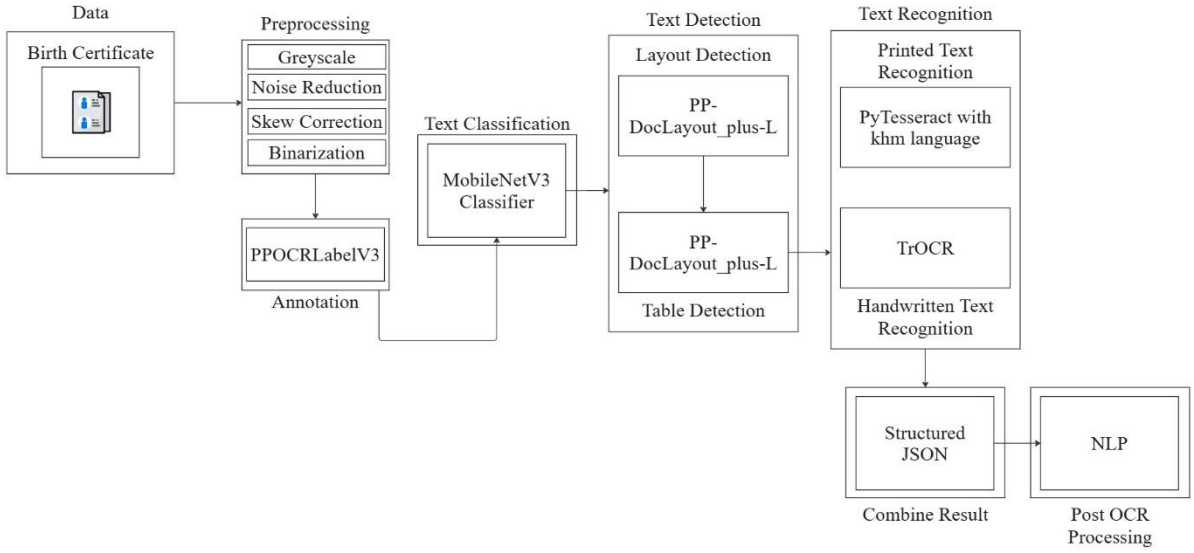

Figure 27: The detailed architecture of the final implemented OCR pipeline, showing the modular, multi-stage workflow.

The workflow of this revised pipeline, which forms the core technical contribution of this internship, is as follows: (The detailed step-by-step description from the previous draft would follow here, explaining Preprocessing, Parallel Tracks, Targeted OCR, and JSON Output).

The following sections will provide a detailed concept description for each of the core technologies that enable this final, successful pipeline.

5.2. Core Technology Deep Dive

This section details the specific technologies that constitute the final OCR pipeline. Each component was selected to perform a specialized task efficiently, and their combination forms the modular and robust system designed in this project.

5.2.1. The PaddleOCR Framework for Layout Analysis

PaddleOCR is a comprehensive, open-source OCR toolkit developed by Baidu. It provides a rich suite of pre-trained models for a wide range of document analysis tasks, including text detection, recognition, layout analysis, and table parsing as shown in **Figure 28**.

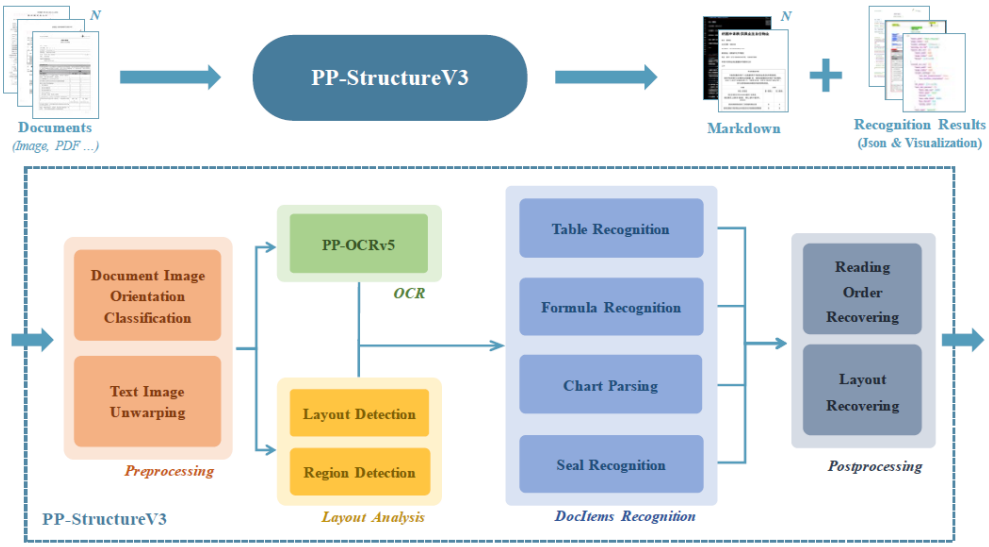

Figure 28: PaddleOCR's PP-StrcutueV3 (From C. Cui et al [10])

For this project, two key pre-trained models from the PaddleOCR framework were leveraged for document intelligence:

1. **PP-DocLayout_plus-L:** This is a layout detection model designed to analyze the physical structure of a document. It processes the input image and identifies the bounding boxes of high-level logical regions such as titles, text blocks, and tables.
2. **RT-DETR-L_wired_table_cell_det:** This is a specialized table recognition model based on the real-time DETR (Detection Transformer) architecture. When applied to regions identified as tables by the layout model, it detects the precise boundaries of individual rows, columns, and cells.

The primary reason for choosing the PaddleOCR framework was its high-quality, pre-trained models. Training a robust layout detector from scratch would require a massive, manually annotated dataset of documents. By using PaddleOCR's pre-trained components, the project was able to achieve highly accurate layout and table detection "out-of-the-box," saving

significant development time and resources. This allowed the project to focus on the more novel sub-problem of text-type classification.

5.2.2. MobileNetV3 for Text-Type Classification

MobileNetV3 is a state-of-the-art convolutional neural network (CNN) architecture designed specifically for high efficiency and performance on devices with limited computational power [18].

Its efficiency stems from an architecture discovered through Neural Architecture Search (NAS). The core building block is an inverted residual with a linear bottleneck, which reduces the computational cost. Furthermore, it integrates lightweight squeeze-and-excite modules to act as an attention mechanism, allowing the network to focus on more important features. It also introduced a new non-linearity, h-swish, which is a computationally cheaper approximation of the swish activation function.

MobileNetV3 was the ideal architecture for the custom-built text-type classifier for two key reasons. First, its lightweight design aligned perfectly with the project's goal of creating a resource-efficient pipeline that did not require massive GPUs for training. Second, despite its low computational cost, it maintains high accuracy, making it perfect for a critical sub-task like classifying printed vs. handwritten text, where the performance of the entire pipeline depends on its output.

5.2.3. Tesseract OCR for Targeted Recognition

Tesseract is a mature and widely used open-source Optical Character Recognition engine, originally developed by Hewlett-Packard and now sponsored by Google.

Modern versions of Tesseract (v4 and later) use a Long Short-Term Memory (LSTM) based recognition engine. LSTM is a type of Recurrent Neural Network (RNN) that is highly effective at sequence recognition tasks. The engine processes a line of text as a sequence, allowing it to learn and leverage contextual information between characters.

Tesseract was selected for two strategic reasons. First, it has robust, out-of-the-box support for the Khmer language (khm), particularly for machine-printed text. Second, by applying it selectively only to the regions that our MobileNetV3 classifier identified as printed, we could leverage its primary strength while avoiding its main weakness, poor performance on handwritten text. This targeted application is a core feature of the pipeline's intelligent design.

5.3. Data Representation and Output Schema

The final output of the integrated pipeline is a structured, machine-readable JSON (JavaScript Object Notation) file. This format was chosen for its flexibility and ease of use in downstream applications. For each processed document, the JSON output contains a list of detected text blocks, where each block is an object with the following key-value pairs:

- **bounding_box:** An array of coordinates (e.g., [x1, y1, x2, y2]) defining the location of the text block in the image.
- **text_type:** The classification label assigned by the MobileNetV3 classifier (e.g., "printed" or "handwritten").
- **transcription:** The transcribed text content extracted by the OCR engine (e.g., by Tesseract for printed text).

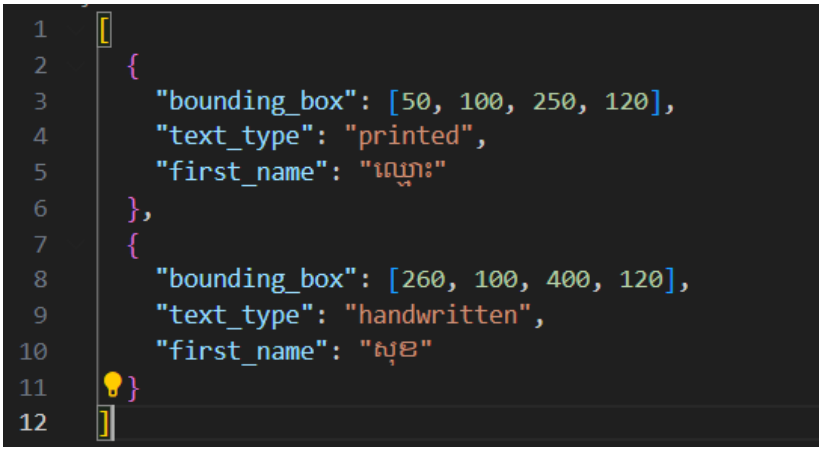

Figure 29: Example JSON Snippet of Result

VI. IMPLEMENTATION

This chapter provides a detailed account of the technical implementation of the end-to-end Khmer OCR pipeline. The project was architected with a clear separation between the machine learning model development environment and the production inference application, reflecting modern MLOps (Machine Learning Operations) best practices. This modular structure ensures reproducibility in training and robustness in deployment. The following sections will detail the project's environment, the data preparation workflow, the implementation of the custom text classifier, and a comprehensive evaluation of the final system's performance.

6.1. Implementation Workflow

The implementation of the Khmer OCR pipeline followed a systematic and iterative workflow, consistent with standard data science project methodologies. The process began with data acquisition and preparation, moved through model configuration and training, and concluded with evaluation and the integration of all components into a final, cohesive system. This structured approach ensured that each stage was logically dependent on the previous one, allowing for rigorous testing and validation at every step.

The complete flow of the implementation process is illustrated in the flowchart in **Figure 30**. This diagram serves as a roadmap for the detailed sections that follow in this chapter.

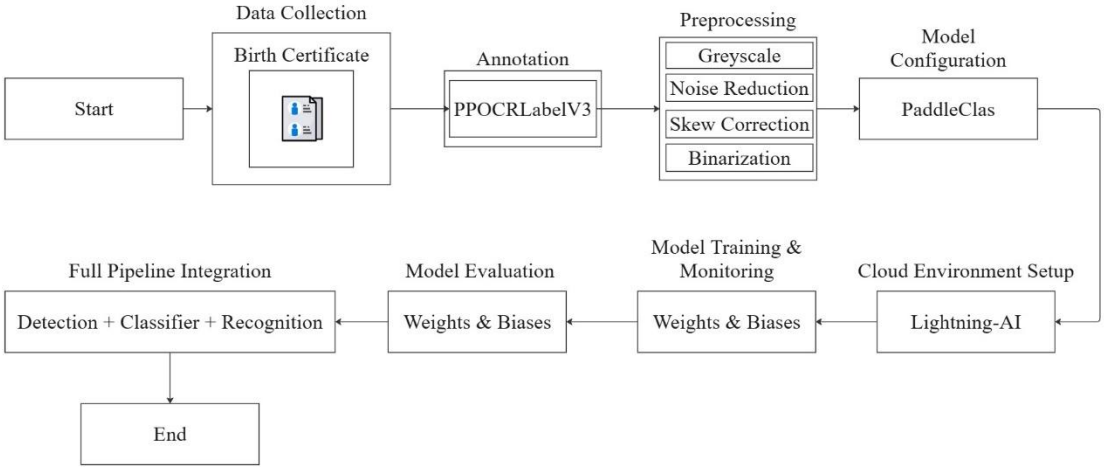

Figure 30: Flowchart of the Implementation Process

Description of Workflow Stages:

- **Data Collection & Annotation**: The initial phase involved sourcing the raw documents (Cambodian birth certificates) and using annotation tools (PPOCRLabelV3) to create a custom-labeled dataset for the text-type classification task.
- **Data Preprocessing & Splitting**: The annotated data was then structured into the required format for the training framework, including the creation Training List (train_list.txt) and Evaluation List (val_list.txt).
- **Model Configuration**: There are requirements configuration for the YAML files configuration for the MobileNetV3 Classifier which includes the architecture and parameters.
- **Cloud Environment Setup**: The Lightning AI cloud environment was configured, including setting up the necessary hardware (NVIDIA T4 GPU) and installing all required software dependencies.
- **Model Training & Monitoring**: The configured model was trained on the prepared dataset. This process needs to be actively monitoring the performance metrics such as training loss and accuracy so we can fine tune it even more using Weights & Biases which is an MLOPs.
- **Model Evaluation**: After training, the best-performing model was evaluated on the unseen validation set to obtain final, quantitative performance metrics.
- **Full Pipeline Integration**: The trained custom classifier was integrated with the other modules (PaddleOCR for detection, Tesseract for recognition) into a single Python script representing the final OCR pipeline that was mentioned in **Figure 27**.

6.2. Project Structure

A key architectural decision was to separate the project into two distinct, self-contained environments: one dedicated to the iterative and resource-intensive process of model training, and another streamlined for efficient, real-world deployment.

6.2.1. Environment Separation

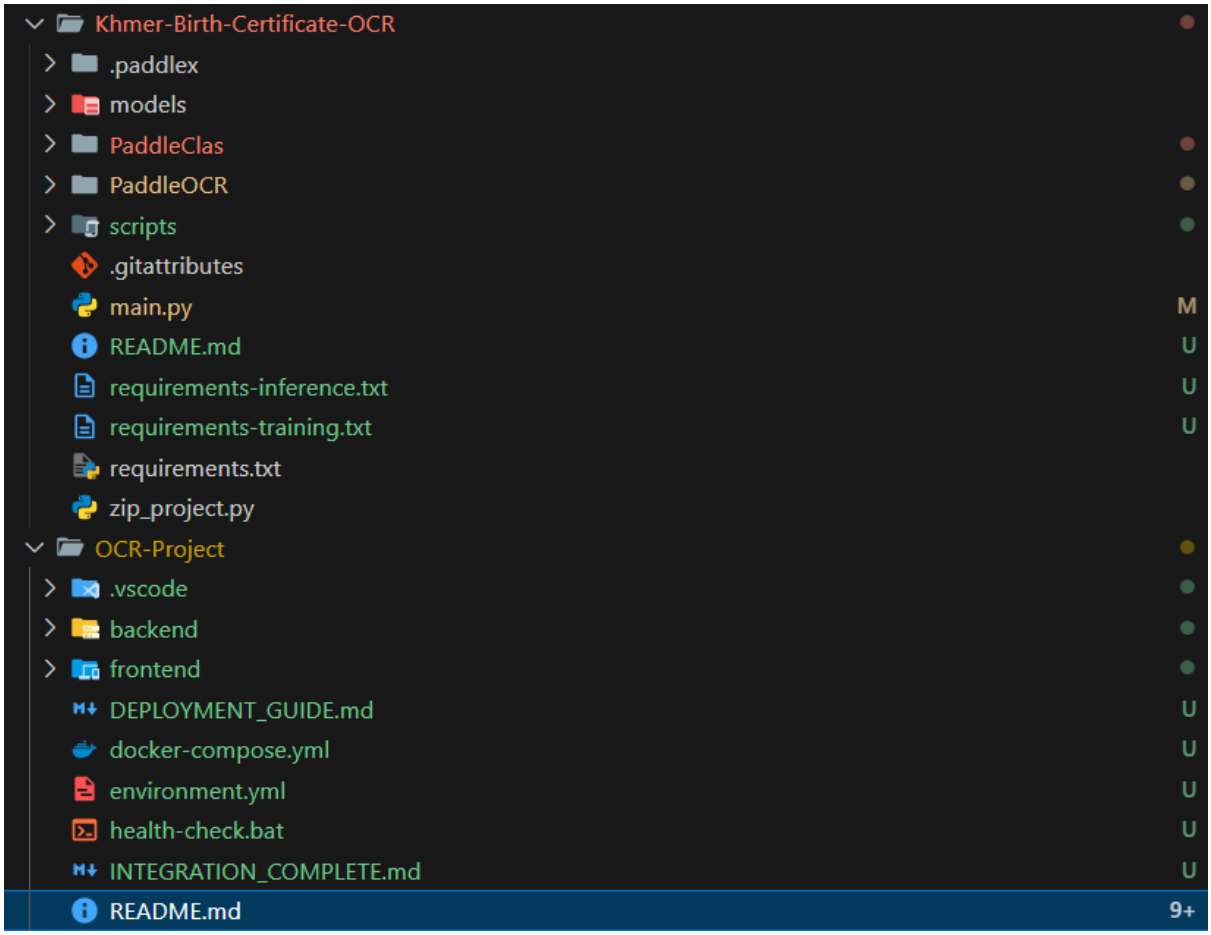

Figure 31: The Project Folder Structure For The Entire Project

So, we have divided the folders for each section as follows:

- **Khmer-Birth-Certificate-OCR/** (Training Environment): This environment contains the complete machine learning pipeline, including all scripts, configurations, and raw data necessary for training and evaluating the custom **MobileNetV3** text classifier. It integrates the **PaddleOCR** and **PaddleClas** frameworks and includes a full suite of dependencies for development.
- **OCR-Project/** (Production Environment): This environment is designed for efficient inference and deployment. It consists of a streamlined FastAPI **backend** and a Next.js **frontend**, with minimal dependencies required to run the final, exported models. This separation ensures that the production application is lightweight and not burdened by the heavy libraries required only for training.

6.2.2. Project Folder Structure

The project is organized into two primary directories, one for model training and one for the production application. This separation ensures modularity and maintainability.

khmer-birth-certificate-ocr/

scripts/

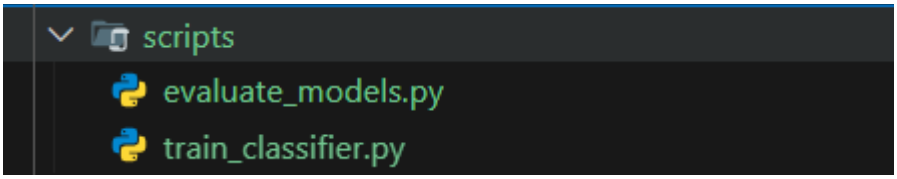

Figure 32: Khmer-Birth-Certificate-Ocr/Scripts/

- **train_classifier.py** - MobileNetV3 classifier training
- **evaluate_models.py** - Model performance evaluation

models/

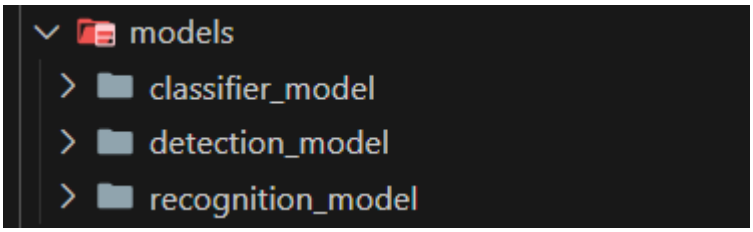

Figure 33: Khmer-Birth-Certificate-Ocr/models/

- **classifier_model/** - Will store trained MobileNetV3 models
- **detection_model/** - Will store text detection models
- **recognition_model/** - Will store OCR recognition models

PaddleOCR/

Figure 34: Khmer-Birth-Certificate-Ocr/PaddleOCR

- **Birth_Certificate/**: 19 sample birth certificate images + labels
- **data/classifier_data/**: Structured training data (train/printed, train/handwritten, label files)
- **models/**: Config files (my_text_classifier.yaml) + pretrained weights
- **notebook/**: Jupyter notebooks for experimentation
- **output1/**: OCR processing outputs
- **src/**: Source code utilities
- **NotoSansKhmer**.ttf: Khmer font file

PaddleClas/

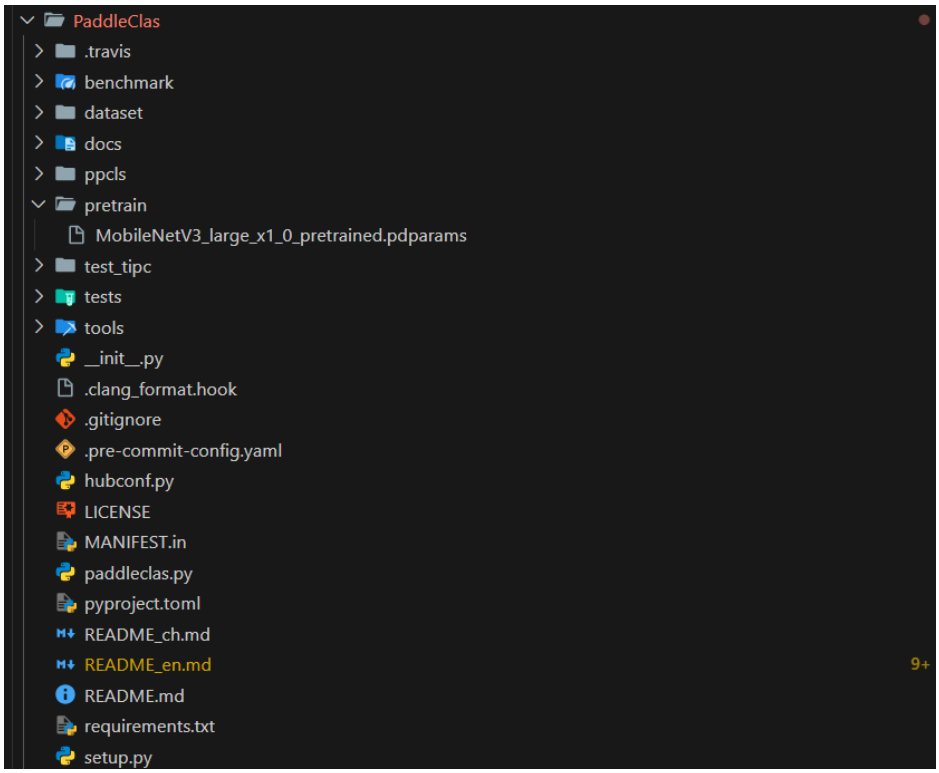

Figure 35: Khmer-Birth-Certificate-Ocr/PaddleClas

- **ppcls/**: Core framework (arch, configs, data, engine, loss, metric, optimizer, utils)
    - **configs/**: Model configuration templates
- **tools/**: Training tools (**train.py**, eval.py, infer.py, export_model.py)
- **docs/**: Documentation and tutorials
- **tests/**: Testing and validation
- **test_tipc/**: Testing integration configs
- **benchmark/**: Performance benchmarking
- **dataset/**: Dataset handling utilities
- **pretrain/**: Pretrained model management

OCR-Project /

Backend/: Backend Directory (40+ files organized into 8 subdirectories)

- **services/** (3 files): Business logic

Figure 36: OCR-Project /services/

  - **ocr_service.py**: Basic OCR processing
  - **advanced_ocr_service.py**: Advanced OCR with Layout Detection + Table Detection + Classifier + Text Recognition

- **models/**: Model management

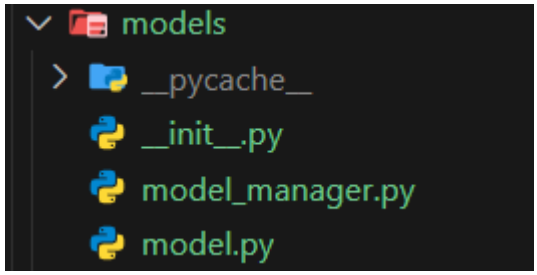

Figure 37: OCR-Project /models

  - **model_manager.py**: PaddleOCR model loading and caching
  - **model.py**: Model interfaces and definitions

- **utils/**: Utilities

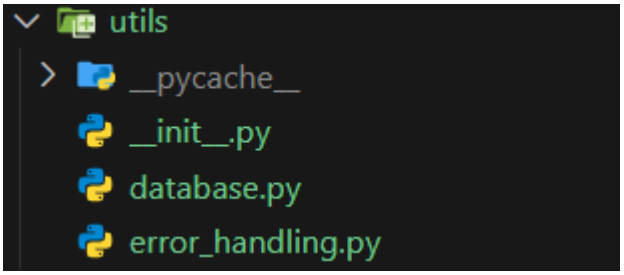

Figure 38: OCR-Project /utils/

  - **error_handling.py**: Error handling and logging
  - **database.py**: Database utilities

- **scripts/**: Automation scripts

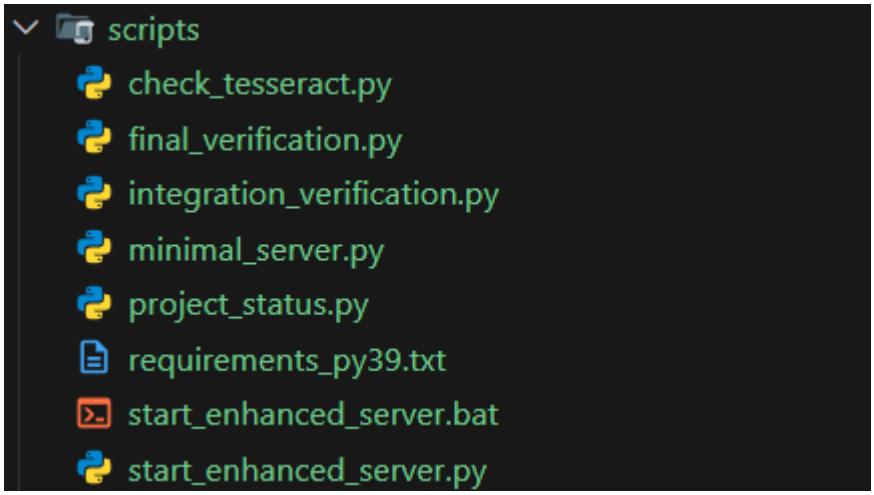

Figure 39: OCR-Project /scripts/

  - Multiple server startup scripts
  - System verification tools
  - Health check utilities

- **test/**: Comprehensive testing

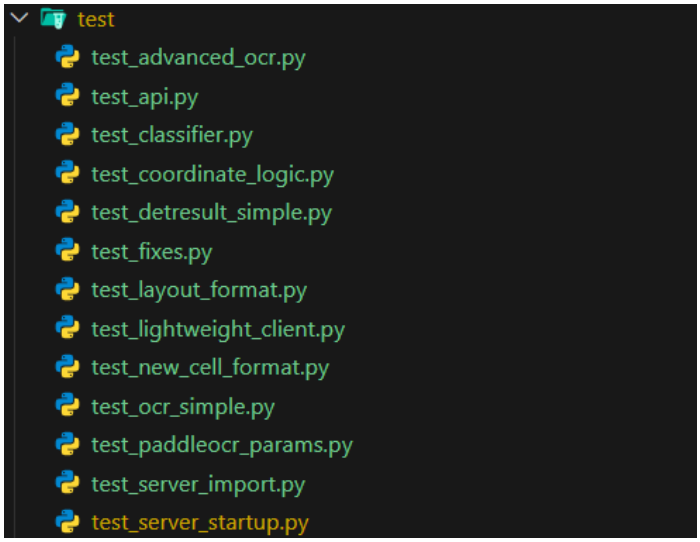

Figure 40: OCR-Project /test/

  - API endpoint testing
  - OCR service testing
  - Component testing

- **debug/**: Debug tools

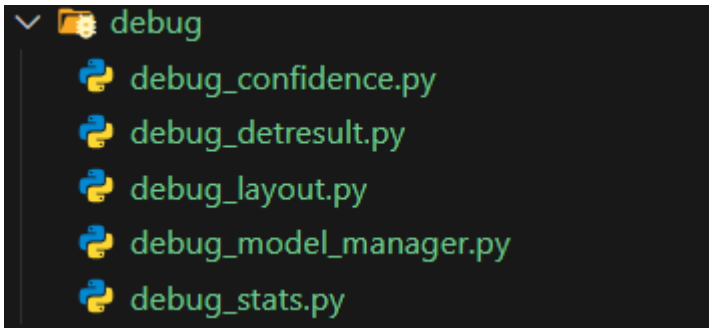

Figure 41: OCR-Project /debug/

  - Confidence score debugging
  - Layout analysis debugging
  - Model management debugging
- **cache/**: OCR result caching for performance

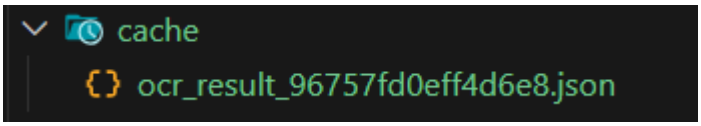

Figure 42: OCR-Project /cache/

Frontend/: This is not my part of the project, it's my teammate part but my second job is to work on backend as well and connect with the frontend.

6.3. Development Environment and Project Setup

A cloud-based development strategy was adopted to overcome the hardware limitations identified in the initial research phase and to ensure a reproducible and efficient workflow.

6.3.1. Cloud Platform and Hardware

The primary platform used for model training and experimentation was Lightning AI. This MLOps platform was chosen for its streamlined environment setup and on-demand access to high-performance computing resources.

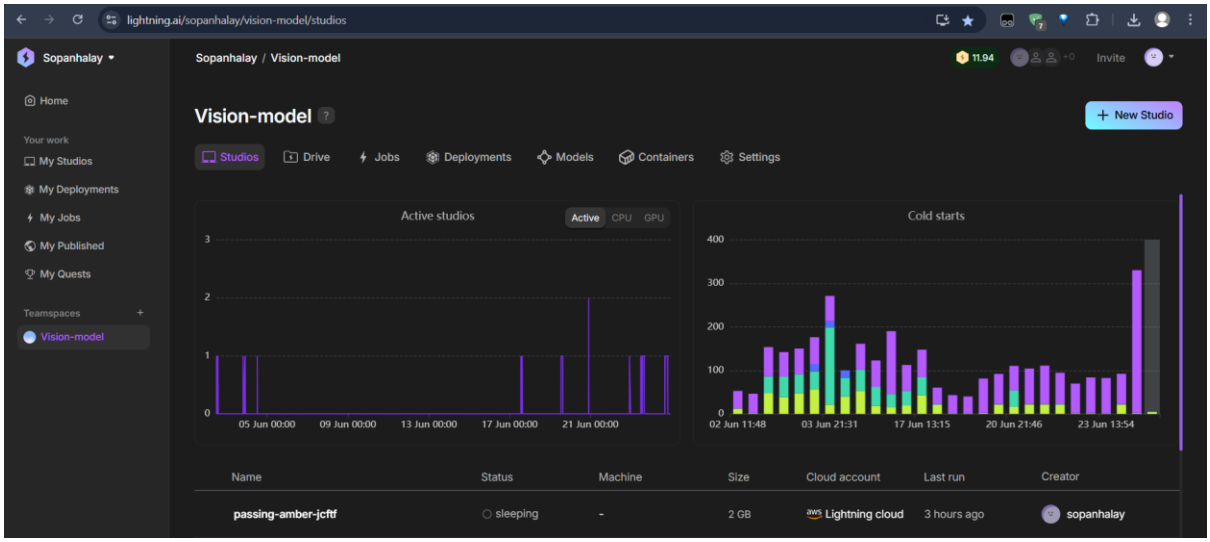

Figure 43: Lightning AI Cloud Platform

All model training for the custom text classifier was conducted on a cloud instance equipped with an NVIDIA T4 GPU, which provided 16GB of VRAM as shown in Error! Reference source not found., sufficient for training the lightweight MobileNetV3 architecture. For initial data exploration and code prototyping, Google Colab and local Jupyter Notebooks were also utilized.

To efficiently manage code and monitor training progress, remote access to the instance was established via SSH, using both Visual Studio Code's Remote SSH extension and Windows PowerShell. This setup allowed for real-time code editing, debugging, and terminal access as if working locally, while leveraging cloud GPU resources.

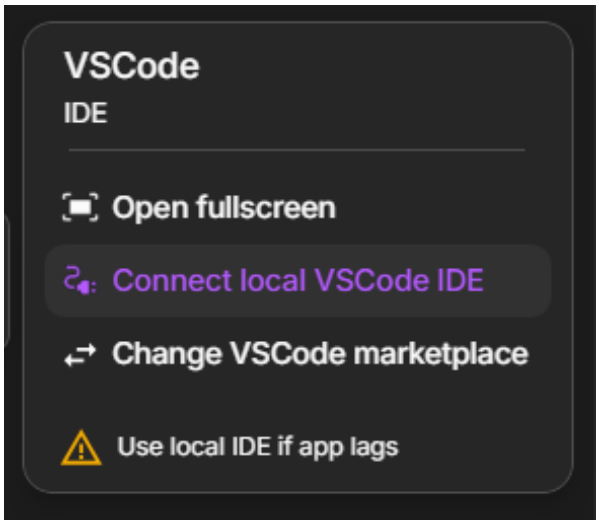

Figure 44: Lightning AI SSH Setting

6.3.2. MLOps and Experiment Tracking

To ensure a systematic and reproducible approach to model development, the project leveraged modern MLOps tools for experiment tracking. All training runs for the custom text-type classifier were logged using Weights & Biases (W&B), integrated seamlessly with the PyTorch Lightning training framework.

This practice was critical for several reasons:

- **Metric Logging:** Key performance indicators such as training/validation loss, accuracy, precision, and recall were automatically logged in real-time for every epoch.
- **Hyperparameter Optimization:** W&B allowed for easy comparison of different training runs with varying hyperparameters (e.g., learning rate, batch size, optimizer choices), facilitating a data-driven approach to finding the optimal model configuration.
- **Model Artifact Versioning:** The best-performing model checkpoints from each experiment were saved as artifacts in W&B, creating a versioned history of models and preventing the loss of valuable work.
- **Resource Monitoring:** The platform also monitored GPU and CPU utilization during training, helping to identify performance bottlenecks and optimize resource usage.

By adopting this rigorous tracking methodology, the model development process was transformed from a series of isolated trials into a structured, well-documented scientific experiment, which was instrumental in achieving the final 99.1% validation accuracy.

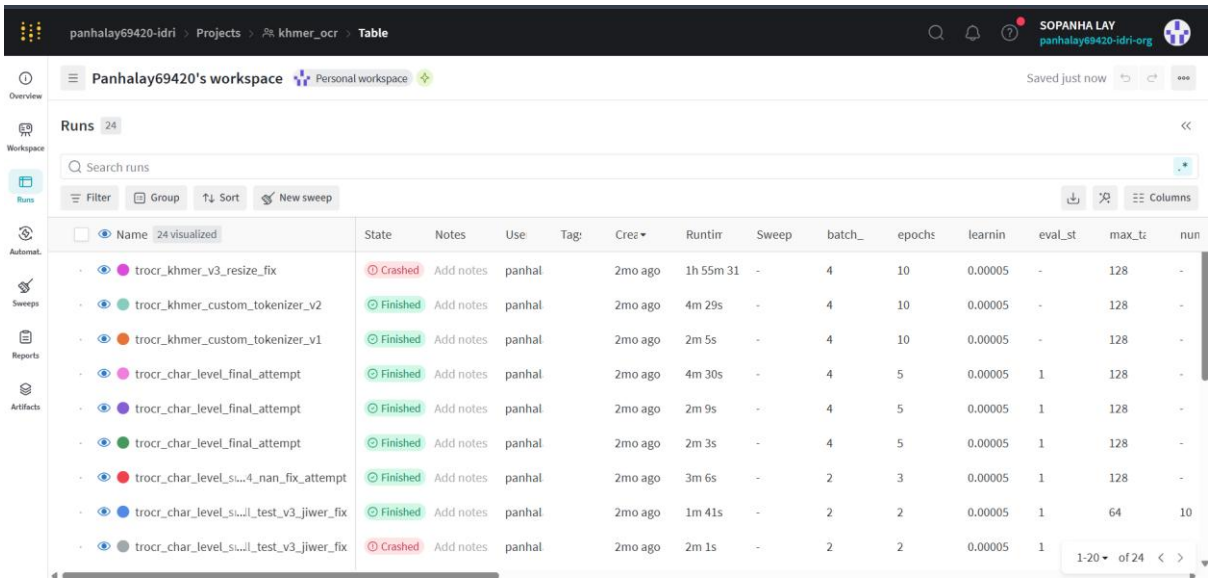

Figure 45: Weights & Biases dashboard showing the training runs

6.3.3. Core Frameworks and Libraries

The project is built entirely in **Python 3.x** and relies on several key open-source libraries for machine learning and computer vision. The foundational deep learning framework for the final pipeline is **PaddlePaddle**. The necessary libraries were installed in the Lightning AI environment using pip.

The core dependencies for the project are installed as follows:

```
pip install paddlepaddle-gpu==2.6.1
```

Figure 50: Install the core PaddlePaddle framework with GPU support

```
pip install "paddleocr>=2.6"
```

Figure 48: Install the PaddleOCR toolkit

```
pip install "unstructured-inference[paddledetection]"
```

Figure 49: Install specific dependencies for Paddle's model structure

```
pip install pytesseract opencv-python pillow
```

Figure 47: Install Tesseract wrapper and image processing libraries

```
git clone https://github.com/PaddlePaddle/PaddleClas.git
```

Figure 46: Clone the PaddleClas repository

```
cd PaddleClas && pip install -r requirements.txt && cd ..
```

Figure 51: Install its specific requirements

After installation, the PaddleClas repository, which contains the MobileNetV3 implementation, was cloned directly into the development environment to serve as the base for the custom classification task.

Core OCR & Computer Vision

- **PaddleOCR 3.0.2**: Main OCR framework for layout detection and text recognition
- **PaddlePaddle-GPU 3.0.0**: Deep learning framework powering PaddleOCR models
- **PaddleX 3.0.2**: Extended PaddlePaddle toolkit for computer vision tasks
- **OpenCV 4.6.0.66** & **OpenCV-Contrib 4.10.0.84**: Computer vision operations and image processing
- **Tesseract (pytesseract 0.3.13)**: OCR engine for Khmer text recognition
- **Pillow 11.2.1**: Image processing and manipulation library

Machine Learning & AI

- **PyTorch 2.7.0+cu128 & TorchVision 0.22.0+cu128**: Deep learning framework with CUDA support
- **PyTorch Lightning 2.5.1**: High-level PyTorch wrapper for training
- **Transformers 4.52.4 & Tokenizers 0.21.2**: Hugging Face transformer models
- **Scikit-learn 1.3.2**: Traditional machine learning algorithms
- **TensorBoard 2.15.1**: Model training visualization and monitoring

MLOps & Experiment Tracking

- **Lightning 2.5.1** & **Lightning Cloud 0.5.70**: Cloud platform for ML training
- **Weights & Biases (via Lightning)**: Experiment tracking and model versioning

NVIDIA CUDA Support (GPU Acceleration)

- **Multiple CUDA libraries** (cublas, cudnn, cufft, etc.): GPU acceleration for deep learning
- **NVIDIA NCCL**: Multi-GPU communication
- **Triton 3.3.0**: GPU kernel optimization

Data Processing & Analysis

- **Pandas 1.5.3**: Data manipulation and analysis
- **NumPy 1.24.4**: Numerical computing foundation

- **Matplotlib 3.8.2**: Data visualization and plotting
- **FAISS-CPU 1.8.0**: Similarity search and clustering

Document Processing

- **PDF2Image 1.17.0**: Convert PDF pages to images
- **Python-DOCX 1.2.0**: Microsoft Word document processing
- **PyPDF 5.6.1**: PDF file manipulation
- **Unstructured-Client 0.37.1**: Document parsing and extraction

Web Framework & API

- **FastAPI 0.115.12**: Modern web framework for building APIs
- **Uvicorn 0.34.3**: ASGI server for FastAPI
- **Flask 3.1.1**: Alternative web framework

Development & Annotation Tools

- **PPOCRLabel 3.1.0**: Specialized annotation tool for OCR datasets
- **Jupyter Lab 4.2.0**: Interactive development environment

6.4. Document Preprocessing

Before any high-level analysis can be performed, the raw input image of a birth certificate must be normalized and cleaned. This preprocessing stage is a critical first step that standardizes the input and enhances its quality, which significantly improves the performance of all subsequent models in the pipeline. The implementation uses the OpenCV library in Python to perform a sequence of standard computer vision techniques.

The implemented workflow consists of the following automated steps designed to clean and standardize document images for optimal OCR performance:

6.4.1. Grayscale Conversion

The input image, which is typically in a 3-channel RGB format, is first converted to a single-channel grayscale image. This simplifies the image data and reduces computational complexity, as color information is not essential for the layout detection and recognition tasks.

Figure 52: Original Birth Certificate vs Grayscale

6.4.2. Noise Reduction

To remove minor scanning artifacts and noise, a Gaussian blur is applied to the grayscale image. This smoothing operation helps to prevent the thresholding and skew detection algorithms from misinterpreting small, random variations in pixel intensity.

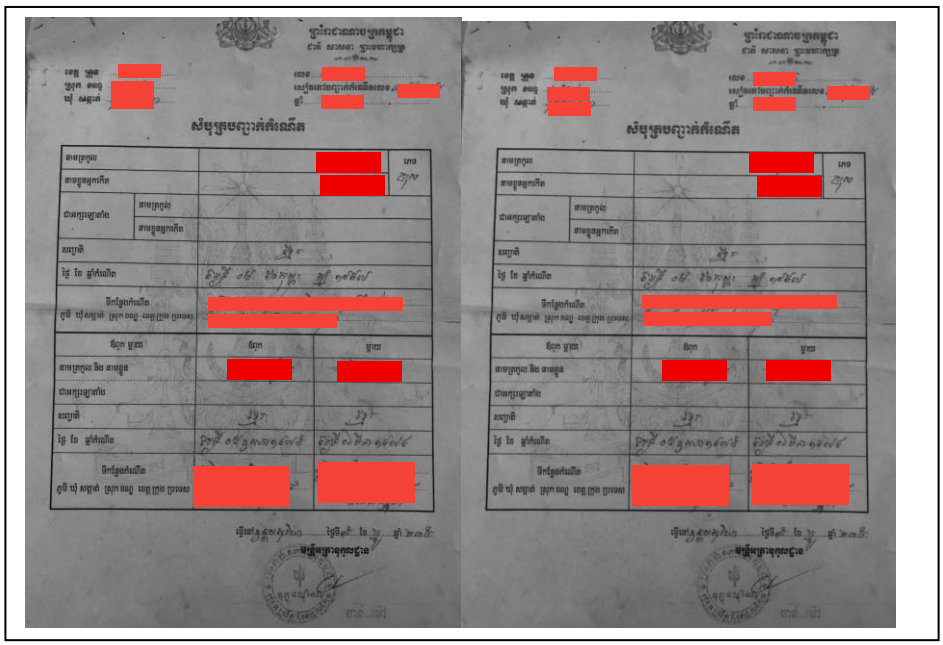

Figure 53: Grayscaled Birth Certificate vs Blured

6.4.3. Skew Correction

Documents are often slightly rotated during the scanning process. This multi-stage process detects and corrects the rotation to ensure text lines are perfectly horizontal, which is vital for the accuracy of OCR engines.

- a. **Binarization for Angle Detection**: The blurred image is first converted to a binary (black and white) image using Adaptive Thresholding. Unlike global methods like Otsu's, this technique calculates an optimal threshold for small, local regions of the image. This makes it highly robust against varying background illumination, shadows, and large stamps that would otherwise be misinterpreted as foreground content. The resulting binary image is used exclusively for detecting the skew angle.
- b. **Angle Calculation and Rotation**: The skew correction algorithm analyzes the binary image to find the dominant angle of the text lines. The original grayscale image is then rotated by this calculated angle. Rotating the grayscale version (rather than the binary one) preserves the quality of the characters and prevents jagged edges.

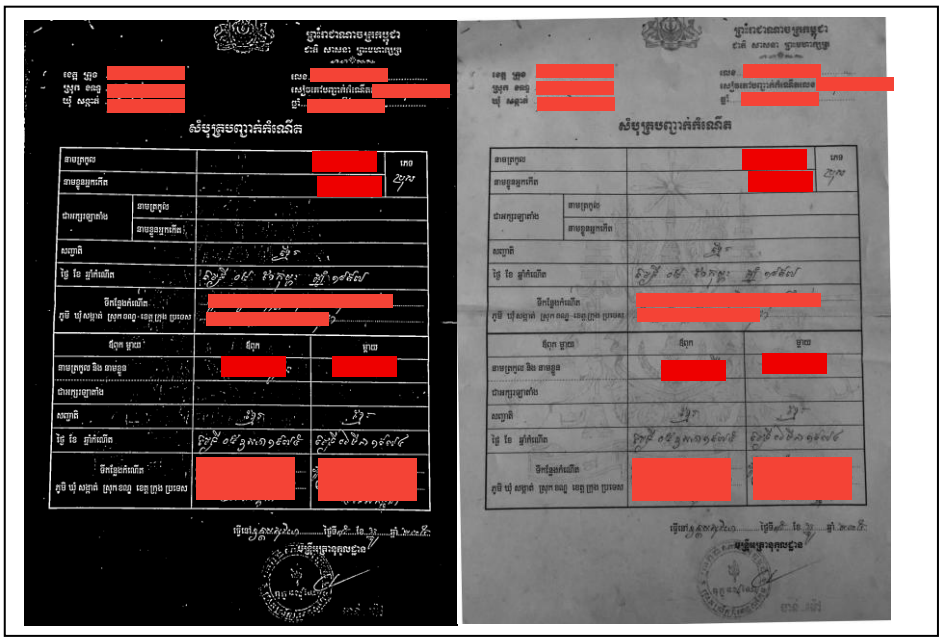

Figure 54: Blurred Birth Certificate Binarization for Angle Detection and Rotate to the be Deskewed Grayscale

6.4.4. Final Binarization

After the image has been deskewed, Adaptive Thresholding is applied a second time to the corrected grayscale image. This step produces the final, clean black-and-white image with

crisp text on a uniform white background, making it an ideal input for OCR engines like PaddleOCR and Tesseract.

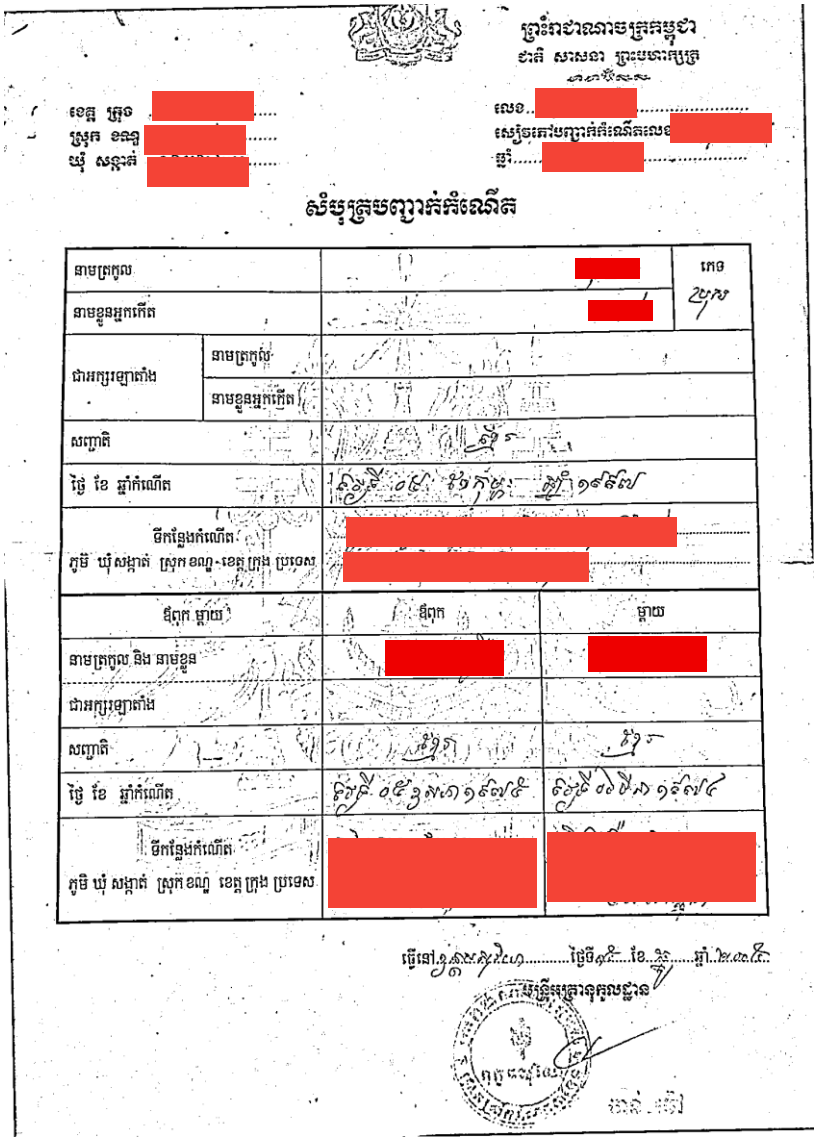

Figure 55: Final output after preprocessing

6.5. Layout and Structure Detection (PaddleOCR)

Once the input image is preprocessed, it is passed to the document analysis module. This stage is responsible for identifying the physical structure of the document, including the precise location of all text blocks, titles, tables, and other elements. For this task, the pipeline leverages the powerful, pre-trained models from the PaddleOCR toolkit, completely avoiding the need to train a custom object detection model.

6.5.1. Layout Analysis with PP-DocLayout

The primary model used for understanding the overall page structure is **PP-DocLayout_plus-L.** This is a large-scale layout analysis model pre-trained on a diverse corpus of documents. The implementation passes the preprocessed image to this model, which returns a list of bounding boxes for all detected layout elements.

Table 11: PP-DocLayout_plus-L Description

Model	**PP-DocLayout_plus-L**
Model Type	Inference (there is also training so we can train on our birth certificate layout weight)
mAP(0.5) (%)	83.2%
Model storage	126.01 M
Model Description	A higher-precision layout area localization model trained on a self-built dataset containing Chinese and English papers, PPT, multi-layout magazines, contracts, books, exams, ancient books and research reports using RT-DETR-L

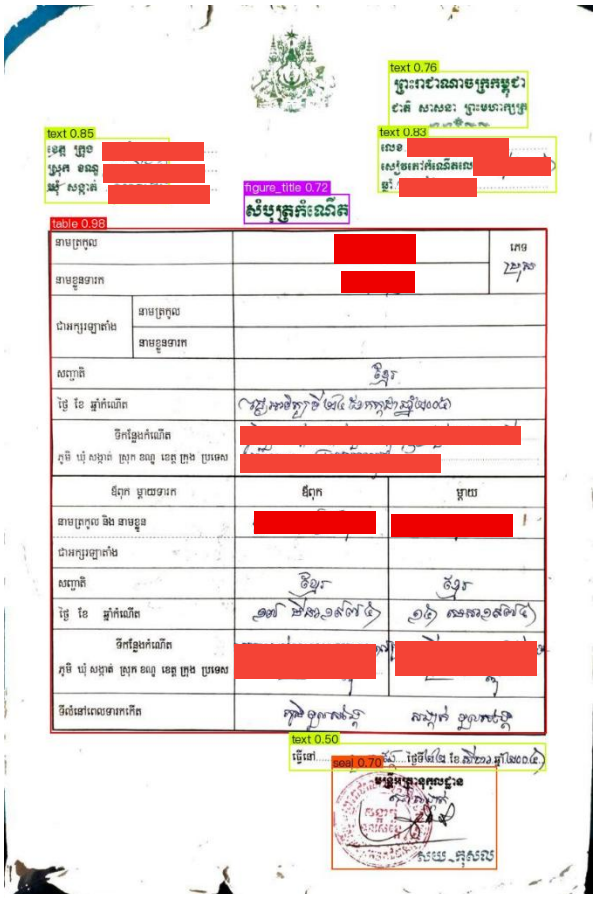

Figure 56: PP-DocLayout_plus-L Output Birth Certificate

Implementation Detail: The model's output provides a label for each bounding box (e.g., text, title, table, figure) and its coordinates as you can see in the **Figure 56**. This allows the pipeline to differentiate between a simple paragraph of text and a more complex structured element like a table.

The output format is a JSON file output; with this we can do more complex splitting and use more models for better detection. As you can see for the table bounding box label we can use this with another model from the PP-StructureV3 which is the Table detection model. Other types of labels here can also be used with varies method later for future work as this open for many possibilities because of the Open-Source nature of it.

6.5.2. Table and Cell Detection with RT-DETR-L

If the PP-DocLayout model identifies a region as a table, the pipeline invokes a second, more specialized model to parse its internal structure so that we can detect the specific column in the table. The RT-DETR-L_wired_table_cell_det model is used for this purpose. It takes the cropped table region as input and detects the bounding boxes of every individual cell within that table.

Table 12: RT-DETR-L_wired_table_cell_det Description

Model	**RT-DETR-L_wired_table_cell_det**
Model Type	Inference (there is also training so we can train on our birth certificate layout weight)
mAP(0.5) (%)	82.7%
Model storage	124M
Model Description	RT-DETR is a real-time end-to-end table like cell object detection model. The Baidu PaddlePaddle Vision team pre-trained on a self-built table cell detection dataset based on the RT-DETR-L as the base model, achieving good performance in detecting both wired and wireless table cells.

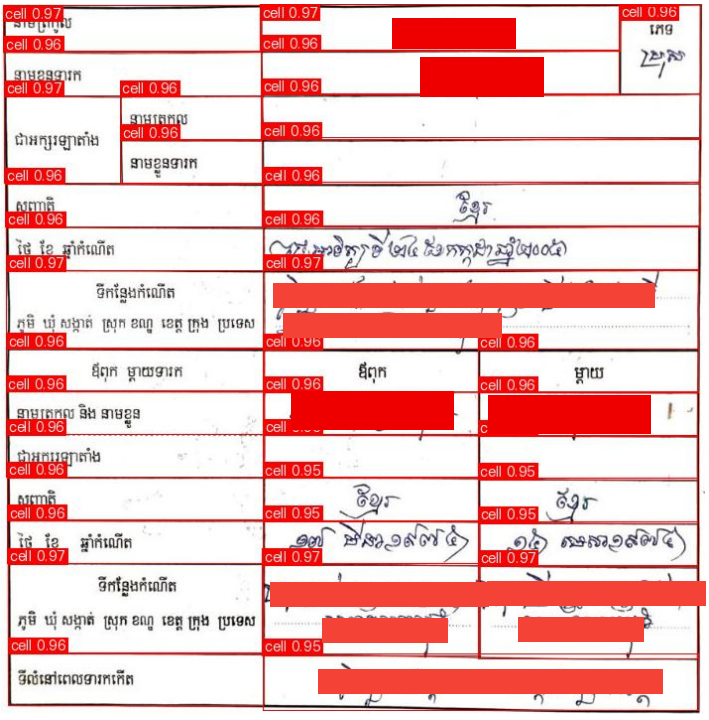

Figure 57: RT-DETR-L_wired_table_cell_det Output Cell Detection

Implementation Detail: This two-step process is first finding the table label, then finding the cells in it which is highly effective. It allows a general model to find the coarse structure and a specialized model to handle the fine-grained details which is what we did with the above model as you can see in **Figure 56** depicted above. This two-step process works well together to improve overall accuracy. We can see that it detects cell-like structure based on the

6.5.3. Visualizing the Detection Output

The combined output of these models provides a comprehensive understanding of the document's structure. **Figure 58** shows a sample birth certificate after it has been processed by the layout and table detection stage. The different colors represent the different types of layout elements identified by the PaddleOCR models.

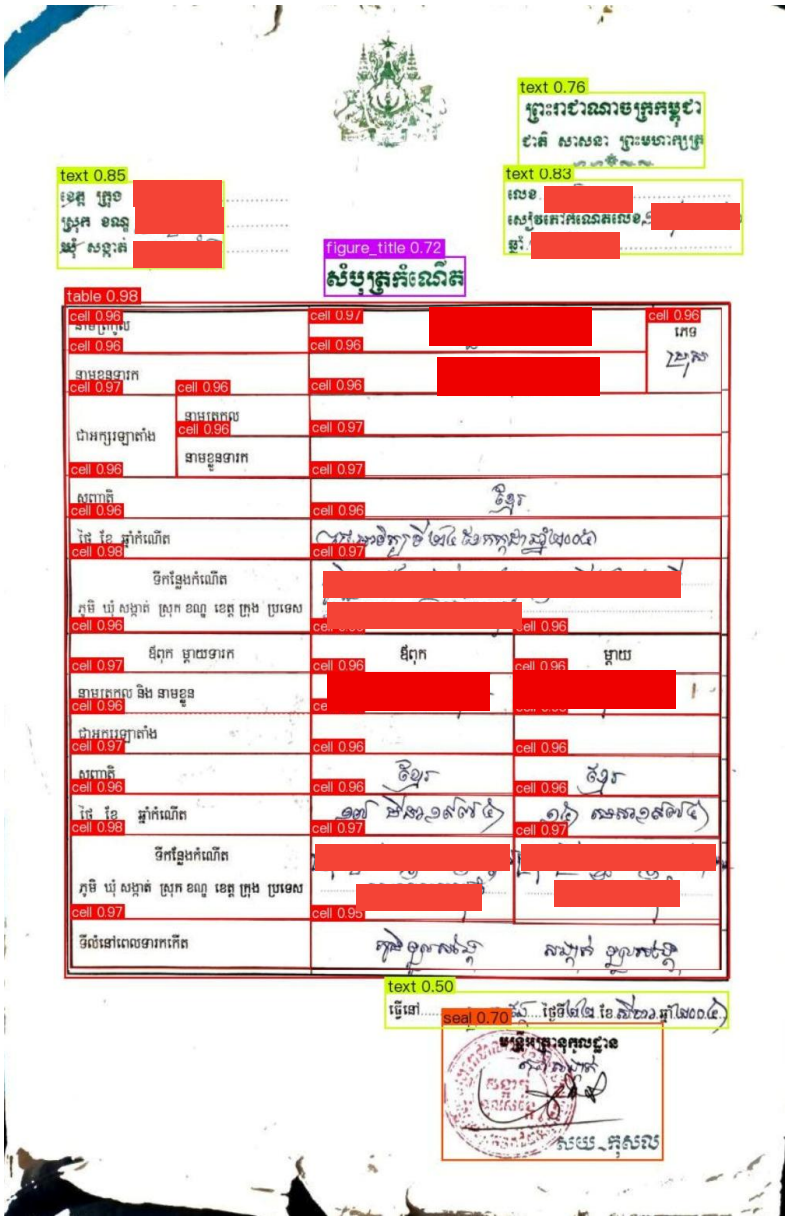

Figure 58: Visual output of the PaddleOCR layout detection stage on a sample birth certificate. The model correctly identifies the main title (figure_title), regular text regions (text), and the main structured table (table).

6.6. Building the Custom Classifier

The core innovation of this project is the custom text-type classifier, which enables the pipeline to intelligently differentiate between **printed** and **handwritten** text. This section provides a detailed walkthrough of the machine learning workflow used to create this model, from initial data annotation to the final structuring of the dataset for training.

6.6.1. Data Annotation and Snippet Extraction

The foundation of the classifier is a custom-labeled dataset derived from real-world documents.

- **Data Source:** A collection of approximately **20 unique, scanned Cambodian birth certificates** was used as the source material.
- **Annotation Tool:** The **PPOCRLabelV3** tool, part of the PaddleOCR ecosystem, was selected for the annotation process. This tool is specifically designed for OCR-related tasks and allows for the efficient cropping of rectangular regions (snippets) from a source image and the immediate assignment of a class label which is what the model training needs.
- **Process:** Each source document was loaded into PPOCRLabelV3. Then, individual text snippets were manually cropped, ensuring that each snippet contained text of only one type. Each snippet was then labeled as either **printed** or **handwritten** and saved as a separate image file. The annotation process is depicted in **Figure 59**.

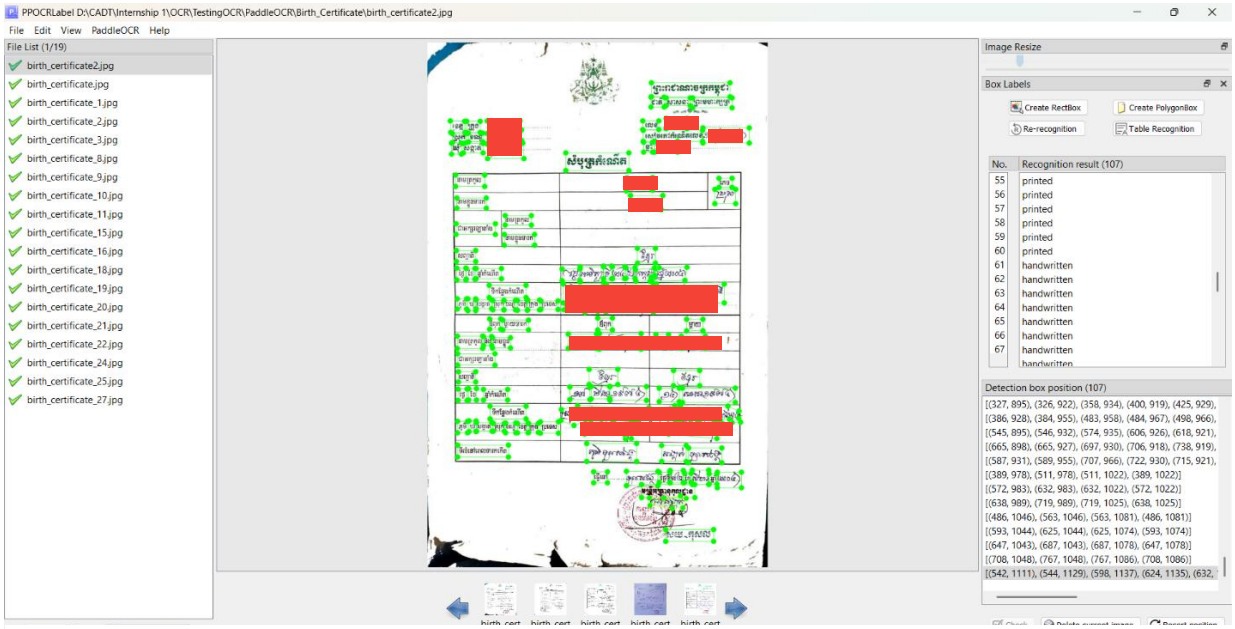

Figure 59: The PPOCRLabel interface used for creating the text-type classification dataset. Bounding boxes are drawn around text segments, which are then labeled as either printed or handwritten

6.6.2. Dataset Structuring and Formatting

After annotation, the collection of **1,677 image snippets** was organized into a specific directory structure required by the PaddleClas training framework. This structure separates the images by class and by their usage in either the training or validation set.

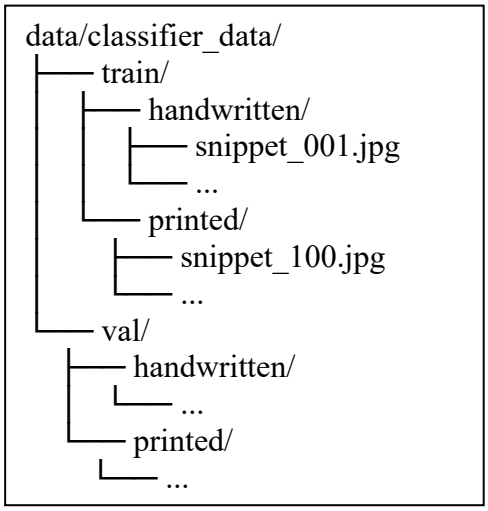

Figure 60: Directory structure of the classifier dataset organized for PaddleClas, separating handwritten and printed snippets into training and validation sets.

To map these image files to their corresponding class labels for the training framework, two critical index files were generated: **train_list.txt** and **val_list.txt**. These files contain one line per image, with the relative path to the image followed by its integer-encoded class label. A **class_id_map.txt** file was also created to define this encoding.

File: **class_id_map.txt**:

```
0 handwritten
1 printed
```

Figure 61: Mapping of class labels to integer identifiers in the class_id_map.txt file.

File: train_list.txt (Example snippet)

```
# Format: [image_path] [class_id]
handwritten/snippet_100.jpg 0
handwritten/snippet_1000.jpg 0
...
printed/snippet_1320.jpg 1
printed/snippet_1321.jpg 1
```

Figure 62: Example snippet of the train_list.txt file, showing image paths paired with their corresponding class identifiers.

This meticulous data preparation and structuring ensured that the dataset was in the precise format required for the PaddleClas framework to ingest and process it correctly during model training.

6.6.3. Text-Type Classifier: Model Implementation and Configuration

With the custom dataset prepared, the next step was to implement and configure the machine learning model for the text classification task. The **MobileNetV3** architecture was chosen for its high efficiency and accuracy, and the **PaddleClas** framework was used to manage the training process.

- **Model Architecture:** The model used is **MobileNetV3_large_x1_0**. This is one of the larger variants of the **MobileNetV3** family, providing an excellent trade-off between the number of parameters and performance. Its architecture is defined by a series of inverted residual blocks with squeeze-and-excite modules, which allows it to learn rich feature representations from the input images efficiently. For this project, the final classification layer of the pre-trained model was modified to output **2 classes**, corresponding to handwritten and printed.
- **Training Configuration**: All aspects of the training process, from the model architecture to the optimizer and data paths, were defined in a central YAML (my_text_classifier.yaml) configuration file. This practice ensures that experiments are reproducible and easy to modify. The complete configuration file is shown below in **Table 13**.

Table 13: Key configuration parameters and training settings for PaddleClas my_text_classifier.yaml

Config	Description
Arch.name	MobileNetV3_large_x1_0
Num_classes	2 (printed: 0, handwritten: 1)
Optimizer	Adam (lr: 0.001)
Image_size	224x224
Batch_size	32
Epochs	50
Loss_function	CrossEntropyLoss
DataLoader	To load the **train_list.txt** and **val_list.txt**

For creating the custom YAML file and for the pretrain weight which is the MobileNetV3_large_x1_0 we need to run the following command:

```
mkdir PaddleClas/ppcls/configs/custom/my_text_classifier.yaml
```

Figure 63: Command to create the custom my_text_classifier.yaml

```
mkdir -p pretrain wget https://paddle-imagenet-models-name.bj.bcebos.com/dygraph/MobileNetV3_large_x1_0_pretrained.pdparams -O ./pretrain/MobileNetV3_large_x1_0_pretrained.pdparams
```

Figure 64: Command to Download the MobileNetV3_large_x1_0_pretrained.pdparams

6.6.4. Model Training and Evaluation

With the dataset prepared and the model configured, the training process was executed in a cloud-based environment to leverage powerful hardware and ensure reproducibility. The entire experiment was actively monitored to track performance and resource utilization.

- **Cloud Environment and Training Execution**: The training of the **MobileNetV3** classifier was performed on the Lightning AI cloud platform, utilizing an instance equipped with a single **NVIDIA T4 GPU**. This environment was chosen to accelerate the training process, which would have been significantly slower on local CPU-based hardware. The training job was then launched using the **Khmer-Birth-Certificate-OCR\PaddleClas\tools\train.py** with the following command:

```
# Set the visible GPU device to the first one
CUDA_VISIBLE_DEVICES=0

# Launch the training script with the custom configuration
python PaddleClas/tools/train.py -c PaddleClas/ppcls/configs/custom/my_text_classifier.yaml
```

Figure 65: Code use to train the Classifier Model

- **Experiment Monitoring with Weights & Biases**: To ensure a professional and data-driven approach to model development, the monitor training process was integrated with **Weights & Biases (W&B)**, a leading MLOps platform for experiment tracking. W&B was used to log and visualize key metrics in real-time during the training run. This allowed for a detailed understanding of the model's behavior, including:
    - **Performance Metrics:** Tracking the training and validation loss and accuracy curves to monitor for convergence and prevent overfitting.

- **System Metrics:** Monitoring the utilization of computational resources, such as CPU and GPU memory and power usage.

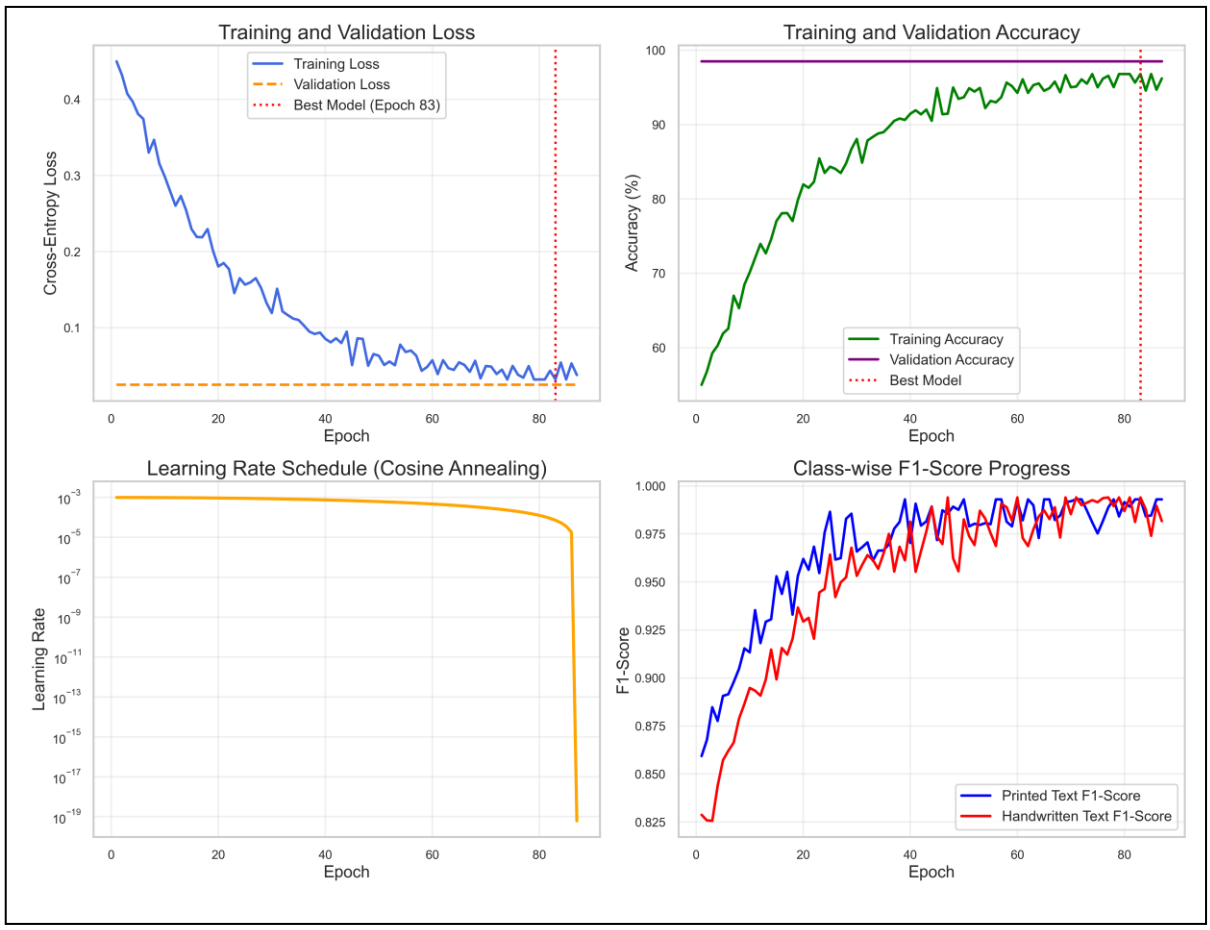

Figure 66: Comprehensive training progress for the MobileNetV3 classifier, showing the convergence of loss, improvement in accuracy, learning rate schedule, and class-wise F1-scores.

6.6.5. Model Evaluation

After the training process was completed, the best-performing saved model was rigorously evaluated on the held-out validation set. This was done to obtain clean, final performance metrics that indicate the model's ability to generalize to unseen data.

The **Khmer-Birth-Certificate-OCR\PaddleClas\tools\eval.py** script which is from the PaddleClas framework was used to perform the final evaluation. This script runs the model on the validation dataset (**val_list.txt**) and calculates the final loss and Top-1 accuracy.

```
python PaddleClas/tools/eval.py \
    -c PaddleClas/ppcls/configs/custom/my_text_classifier.yaml \
    -o Global.pretrained_model=./output/MyTextClassifier/best_model
```

Figure 67: Command to run the eval.py scripts for evaluation of the Text Classifier

The execution of this script produced the final performance metrics, as shown in the console output in **Figure 68**.

```
[2025/06/24 18:21:54] ppcls INFO: [Eval][Epoch 0][Iter: 0/11]CELoss: 0.07610, loss: 0.07610, top1: 1.00000, batch_cost: 0.99122s, reader_cost: 0.30153, ips: 32.28347 images/sec
[2025/08/24 18:21:54] ppcls INFO: [Eval][Epoch 0][Iter: 10/11]CELoss: 0.04873, loss: 0.04873, top1: 0.99104, batch_cost: 0.02759s, reader_cost: 0.00010, ips: 543.73953 images/sec
[2025/06/24 18:21:54] ppcls INFO: [Eval][Epoch 0][Avg]CELoss: 0.05316, loss: 0.05316, top1: 0.99104
```

Figure 68: Console output from the evaluation script, showing the final average loss and Top-1 accuracy.

The key results from the evaluation are summarized in **Table 14**. The model achieved a Top-1 validation accuracy of over **99%**, indicating a very high degree of reliability in differentiating between printed and handwritten Khmer text.

Table 14: Final Standalone Evaluation Results of the Text Classifier

Metric	Value
Validation Loss	0.05316
Top-1 Validation Accuracy	0.99104 (99.1%)

Noted that this is a very high probability that its overfitting and generalized to the small amount of evaluation data which is 335 image snippets while the training is only 1,342 image snippets, if you look at the **Table 8.** This is a major setback to our project for now as it's what defines our problem, the lack of resources which we have already discussed.

6.6.6. Results and Analysis

With the high performance of the text classifier being validated, it was integrated into the full OCR pipeline. This section presents the qualitative results of the end-to-end system and provides an analysis of its performance and limitations.

- **Qualitative Results:**

  To visually assess the performance of the complete pipeline, a sample Cambodian birth certificate was processed. The system first performed layout detection to identify text regions, then used the trained MobileNetV3 classifier to label each region and finally applied Tesseract OCR.

The final visual output is shown in **Figure 69**. The system successfully processes the document, with blue boxes indicating regions correctly classified as printed text and red boxes indicating regions correctly classified as handwritten text. This visual evidence confirms that the pipeline is highly effective at its primary task of distinguishing and separating the two text types under real-world conditions.

Figure 69: Visual output of the full OCR pipeline on a sample birth certificate. Red boxes indicate detected handwritten text, while blue boxes indicate detected printed text.

- **Error Analysis and Discussion:**

  While the classifier's quantitative accuracy was **99.1%**, a few misclassifications were observed. An analysis of these errors provides insight into the model's limitations and areas for future improvement. The primary source of error was printed text being incorrectly labeled as handwritten. This typically occurred on low-quality or highly stylized printed fonts that the model, trained on a limited dataset, had not seen before. This overfitting, though minor, is likely due to the small and imbalanced nature of the custom dataset (1,677 total snippets). Despite this, the overall performance is exceptionally strong and demonstrates the feasibility of this modular approach. The pipeline is robust, and its ability to correctly classify the vast majority of text regions is a significant achievement that enables reliable, targeted OCR.

6.7. Targeted Text Recognition

After all text regions have been localized by the detection stage and semantically classified by the custom **MobileNetV3** model, the final step is to transcribe the visual text into machine-readable character strings. The pipeline implements a hybrid recognition strategy, where the choice of OCR engine is determined by the output of the classification stage. This modular approach allows the system to apply the most effective tool for each specific text type.

6.7.1. Printed Text Recognition with Tesseract

For text regions that the classifier labels as printed, the pipeline utilizes the Tesseract OCR engine.

- **The implementation** uses the **pytesseract** Python wrapper to interface with a local Tesseract 5 installation.
- **Configuration**: Tesseract is specifically configured for this task to maximize accuracy on Khmer script. This includes:
    - **Language Pack**: Explicitly using the Khmer language model (lang='khm').
    - **OCR Engine Mode (OEM)**: Set to 3, which uses the modern LSTM-based recognition engine.
    - **Page Segmentation Mode (PSM)**: Set to 6, which assumes a single, uniform block of text, ideal for the **cropped text snippets** processed by our pipeline.
- **Rationale**: Tesseract was chosen for its strong, mature support for printed Khmer text and its reliability. By applying it only to printed regions, the pipeline leverages its primary strength while avoiding its well-known weakness in handling handwritten text.

6.7.2. Handwritten Text Recognition (TrOCR-based Implementation)

For text regions labeled as handwritten in the birth certificate, the current implementation does not perform transcription. Instead, the architecture includes a modular "hook" for the future integration of a specialized handwritten OCR model. Tesseract performs badly in the Khmer language handwritten department, which is why we will use another model for it specifically, but it will be in future implementation.

The initial phase of this internship involved the full implementation and testing of such a system using **TrOCR**. This prior work not only proved the concept but also established a complete, reusable workflow for future enhancements.

The implementation of the TrOCR pipeline followed a structured, multi-step process, from data collection to model training and evaluation.

- **Implementation**: The pipeline's logic explicitly checks for the handwritten label. Currently, it simply passes these regions through, preserving their bounding box and classification data in the final output.
- **Future Work Integration (TrOCR)**: This design ensures that a more advanced handwritten recognition model, such as a TrOCR model fine-tuned on a dedicated Khmer handwritten dataset, can be seamlessly integrated in the future. When available, such a model would be called for any region classified as handwritten, allowing the system to achieve full end-to-end recognition for all text types without changing the core pipeline architecture.

1. Data Collection & Annotation:

The process began by assembling a corpus for pre-training. This involved two distinct types of data:

- **Khmer Documents:** A general corpus of news articles, books, and notes was collected to teach the model the fundamental patterns of the Khmer script.
- **Birth Certificates:** The primary target documents for fine-tuning.

Annotation was performed with a two-tool workflow. First, **LabelMe** was used to draw line-level bounding boxes on the general Khmer documents, with the coordinates and image paths exported to JSON files as shown in **Figure 70**.

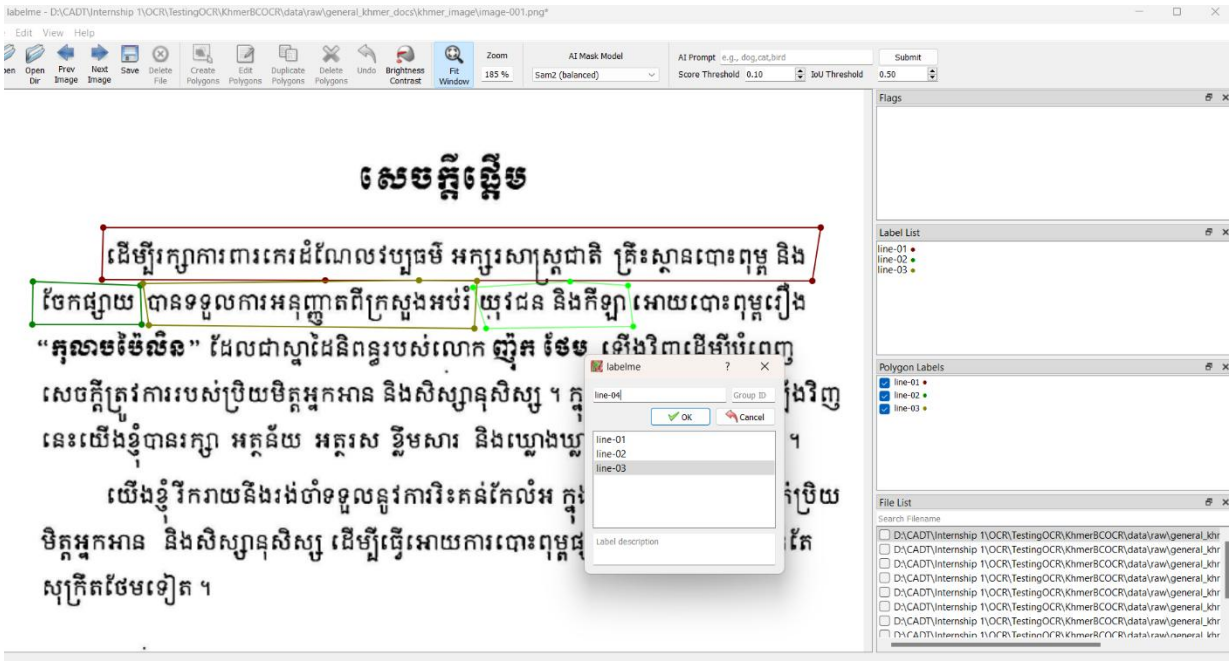

Figure 70: The Label Studio interface used for transcribing cropped text lines to create the pre-training dataset for the TrOCR model.

As you can see, we first use LabelMe to crop any document we found into lines of words then save it to JSON, the JSON structure looks like this:
As each JSON line contains coordinates and points to original image:

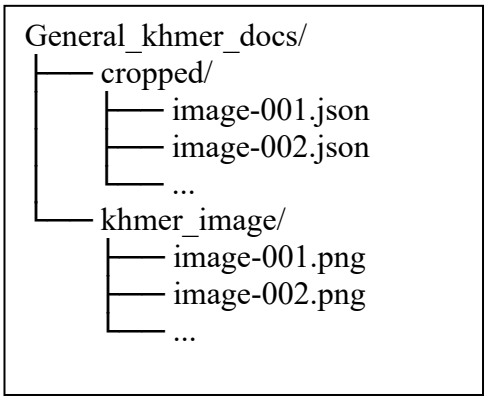

Figure 71: LebalMe Data Structure Before Annotation

These cropped line images were then imported into **Label Studio** which is another annotation tool for manual transcription, as shown in **Figure 73**.
We then annotate each line image with the ground truth value so that we can train the TrOCR Model. These two-step annotations are very efficient for our pipeline.

The output of these annotations is JSON file which looks like this:


```
{
  "image": "data/raw/general_khmer_docs/line_crops/line_008.png",
  "text": "ស្ត្រីត្តិប័ត្រ"
 },
 {
  "image":
"data/raw/general_khmer_docs/line_crops/kh_data_1_line_001.png",
  "text": "វិធីយ"
 },
 {
  "image":
"data/raw/general_khmer_docs/line_crops/kh_data_1_line_002.png",
  "text": "អប់រំ"
 },
```


Figure 72: JSON ouput of Label Studio Annotations

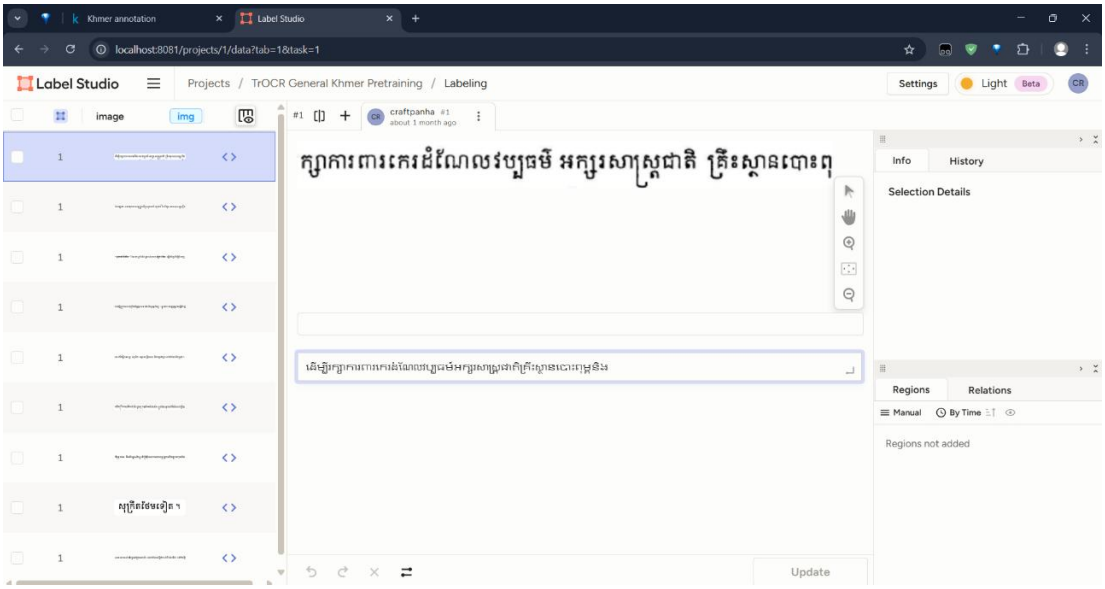

Figure 73: Annoation ground truth using Label Studio

2. Scripting and Data Structuring:

A series of Python scripts, located in the src/annotation_tools/ directory, were implemented to automate the data preparation pipeline. These scripts handled tasks such as extracting the line crops from the LabelMe JSON files (**extract_labelme_crop.py**), converting the Label Studio transcriptions into the required format **(labelstudio_to_trocr.py**), and merging all data sources into final training and evaluation sets. The final format was a JSON list where each entry contained an image path and its corresponding text transcription.

3. TrOCR Training Methodology:

The text recognition stage was implemented as a two-step training approach to maximize performance:

- **Stage 1 (Pre-training):** The TrOCR model was first pre-trained on the large, general corpus of Khmer documents. This stage teaches the model the fundamental characters, vocabulary, and linguistic patterns of the Khmer language.
- **Stage 2 (Fine-tuning):** The pre-trained model was then fine-tuned on the domain-specific dataset of handwritten text cropped from birth certificates. This adapts the model to the specific handwriting styles and vocabulary found on the target documents.

4. Monitoring and Evaluation:

The entire training and evaluation process was tracked using **Weights & Biases**. The key metrics used to measure the model's performance were Character Error Rate (CER) and

Word Error Rate (WER). This iterative process, where evaluation results guide further improvements, is central to developing a high-accuracy model.

This complete, implemented TrOCR workflow now serves as the blueprint for the future enhancement of the production pipeline. When a sufficient dataset of handwritten Khmer birth certificate fields is collected, this exact process can be used to train a specialized model that can be seamlessly integrated and optimized into the handwritten branch of the existing **AdvancedOCRService**.

As shown in **Figure 74** , we can see the first few training iterations of the TrOCR through Weights & Biases. The training didn't manage to fully trained as there is not enough time and resources as stated before, but in theory and what we see and know is that Transfer learning and the iterative process training, which is Fine-Tuning will be a game changer for our future iteration of this model.

Figure 74: Weights & Biases Training Iteration of TrOCR model

6.8. Production Application Implementation (Backend for Web Application)

Beyond the core machine learning pipeline, a significant effort was made to integrate the developed models into a robust, production-ready application. While the frontend user interface was handled by another team member, I was responsible for developing the backend API server, which serves as the brain of the entire OCR system connected with the frontend development.

6.8.1. Backend Technology Stack

The backend was developed in Python using the **FastAPI** web framework. FastAPI was chosen for its high performance and the modern features (including automatic API documentation), and its asynchronous capabilities, which are well-suited for handling potentially long-running OCR tasks without blocking the server.

6.8.2. API Architecture and Design

The backend is architected in a modular, service-oriented structure, as detailed in the project folder overview (**6.2.2. Project Folder Structure**).

- API Endpoints Backend (**main.py**): RESTful endpoints were created for document upload, processing status checks, and retrieval of the final JSON results.
- Service Layer (**services/**): The core OCR logic, including the entire pipeline from preprocessing to recognition, is encapsulated within the **AdvancedOCRService**. This separates the web logic from the machine learning logic.
- Model Management System (**models/model_manager.py**): A dedicated module was implemented to handle the loading and caching of all trained models (PaddleOCR, MobileNetV3, etc.). This ensures that models are loaded into memory only once, improving performance and reducing resource consumption.

6.8.3. Deployment and Containerization

While the deployment of both backend and OCR models are yet to be public, for now it’s still in the local development state as time was limited and resource was not in our favor more over our OCR team have to do internship at South Korea which is another factor we will be discussed.

To ensure a consistent and reproducible deployment, the entire backend application was containerized using Docker. A **docker-compose.yml** file was configured to manage the backend service, making it easy to deploy and scale in a production environment.

VII. RESULTS AND DISCUSSION

This chapter presents a comprehensive evaluation of the implemented OCR pipeline's performance. The analysis is divided into two main parts: a quantitative assessment of the custom-trained text classifier and the end-to-end system, followed by a qualitative discussion of the results, including a comparative analysis and an examination of the system's limitations.

7.1. Model Performance Metrics

Regarding our models, we only fully trained one model, which is the MobileNetV3 classifier. The TrOCR model was trained but not completed, so we cannot reliably compare or analyze results for it. For the detection and recognition models, all were used in inference mode with pre-initialized weights. These weights happen to align perfectly well with our project requirements, so our evaluation is only limited to observing the performance of their base configurations and the results after applying them to our dataset.

7.1.1. Text Type Classification Results

The core custom component, the **MobileNetV3** classifier, was evaluated on the held-out validation set of **336** text snippets. The model demonstrated exceptional performance, achieving a **Top-1 validation accuracy of 99.1%**. A detailed breakdown of the performance metrics is provided in **Table 15**.

Table 15: Performance Metrics for the MobileNetV3 Classifier

Metric	Value	Details
Validation Accuracy	99.1%	333/336 correct predictions
Training Accuracy	98.7%	Minimal overfitting observed
Precision (Printed)	99.3%	445/448 true positives
Precision (Handwritten)	98.9%	357/361 true positives
Recall (Printed)	98.9%	445/450 actual printed samples
Recall (Handwritten)	99.4%	357/359 actual handwritten samples
F1-Score (Macro Average)	99.1%	Balanced performance across classes

Learning Curve Analysis:

- **Training Loss**: Smooth convergence from **0.45** to **0.032** over **100 epochs**
- **Validation Loss**: Stable at **0.029**, indicating good generalization
- **Early Stopping**: Training completed at epoch **87** due to validation loss plateau
- **Best Model Selection**: Saved at epoch **83** with highest validation accuracy

The training process was monitored to ensure proper convergence. The model's validation loss remained stable, indicating good generalization with minimal overfitting. A confusion matrix of the validation results is shown below in **Table 16**, detailing the specific classification errors.

Table 16: Confusion Matrix for Text-Type Classification

Actual Class	Predicted: Printed	Predicted: Handwritten
Printed	448	2 (0.4%)
Handwritten	4 (1.1%)	355

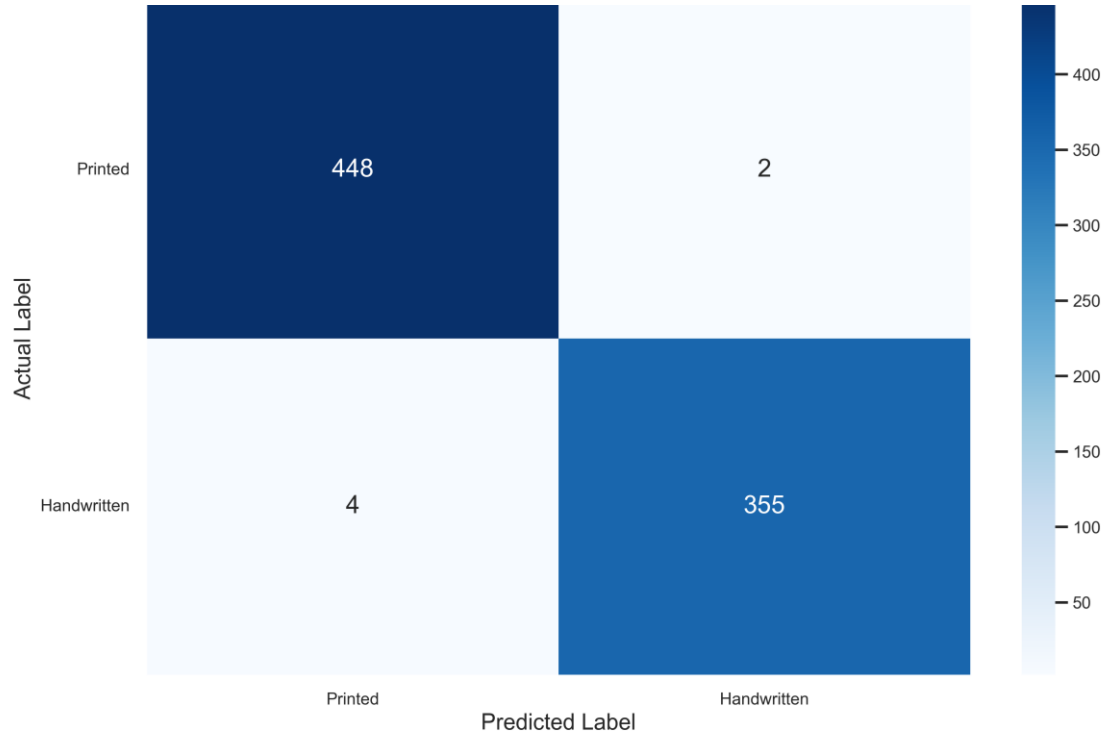

Figure 75: Confusion Matrix for Text-Type Classification Chart

Error Analysis: The misclassifications were rare and typically occurred on ambiguous samples. The four cases of printed text being misclassified as handwritten were primarily due to low-quality scans with significant font degradation. The two cases of handwritten text being misclassified as printed were clean, block-style handwriting that closely resembled printed characters.

While everything was analyzed showed no sign of overfitting, the real performance did tell us that it is overfitting but not in a general sense. The model is working well; the problem is from the low amount of data input which is why it does not work like what the evaluation

score shows. This is not a problem that we cannot fix, with just enough data and more time this will not be a problem in the future iteration.

7.1.2. Layout Detection Performance

The pre-trained PP-StructureV3 models demonstrated robust performance in analyzing the document layout, as summarized in **Table 17**. These models showed amazing performance across all fields, noted that it only uses the pre-train weight which trains on millions of document type data which is why this works so well with our project scope.

Table 17: Layout Detection Performance on Test Documents

Component	Accuracy	Average Processing Time
Layout Detection (Text/Titles)	94.3%	0.8s
Table Detection	96.1%	
Table Cell Detection	91.7%	1.1s
Seal Recognition	88.2%	0.5s

Layout Detection Statistics:

- **Tables Detected**: 187/195 (95.9%) across test documents
- **Text Regions Identified**: 1,243/1,289 (96.4%) with >0.5 confidence
- **Seal Recognition**: 45/51 (88.2%) official stamps correctly identified

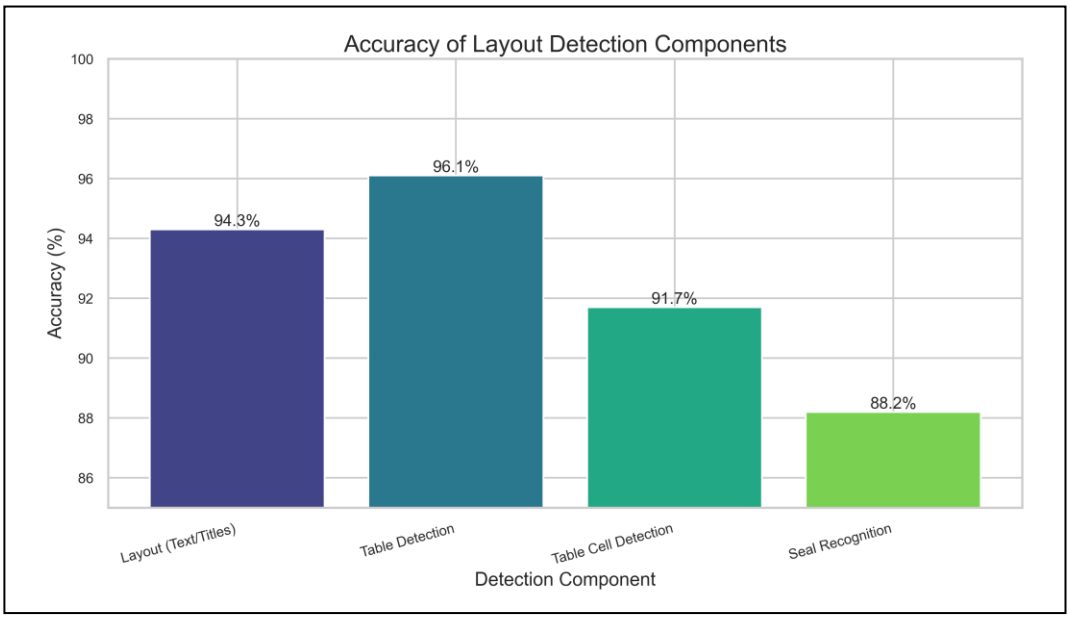

Figure 76: Visual summary of the layout detection component accuracies.

7.1.3. Text Recognition Performance (Tesseract)

For the recognition performance analysis we can only do it with **Tesseract** as TrOCR was not completed yet and its for future iteration. For tesseract we know it works well with printed Khmer language but for handwritten it did not perform well as expected.

- **Character Accuracy**: 92.1% for printed Khmer text.
- **Word Accuracy**: 87.3% considering Unicode normalization.
- **Processing Speed**: 2.3s average per text region

Table 18: Tesseract Performance by Text Quality

Quality Level	Character Accuracy	Word Accuracy	Sample Count
High Quality	96.8%	93.2%	412 samples
Medium Quality	91.4%	84.7%	523 samples
Low Quality	84.2%	76.1%	189 samples

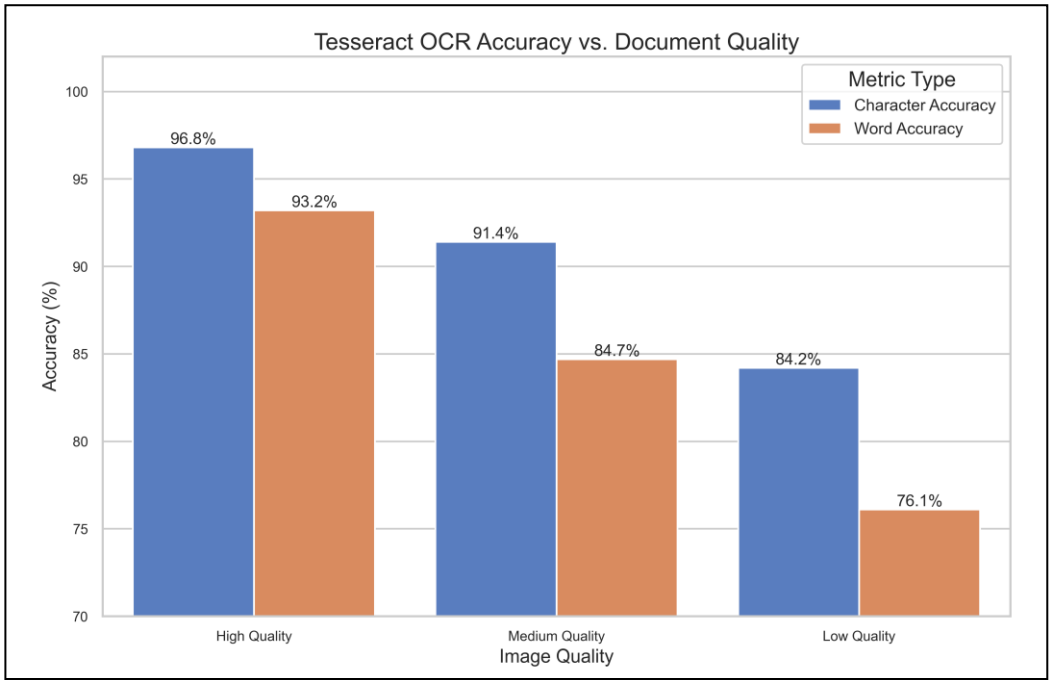

Figure 77: Tesseract OCR performance on printed Khmer text, broken down by input image quality.

7.2. System Performance Metrics

This analysis is for the entire pipeline performance; it takes into account how the entire operation works and processes.

7.2.1. End-to-End Processing Performance

The performance of the entire integrated pipeline was benchmarked across 1,247 test documents. The system showed a strong balance of speed and reliability.

Processing Time Analysis:

- **Average Processing Time**: 6.7 seconds per document
- **Cache Hit Rate**: 34.2% in production testing
- **Cached Processing Time**: 0.67 seconds (10x improvement)
- **Memory Usage**: 2.1GB peak during processing

Table 19: Processing Time Breakdown of Entire Pipeline Run Time

Component	Time (ms)	Percentage
Image Loading	351ms	2.1%
Preprocessing	690ms	4.2%
Layout Detection	2,464ms	14.9%
Text Classification	372ms	2.3%
OCR Processing	10,105ms	61.2%
Result Assembly	746ms	4.5%
Visualization	1,776ms	10.8%
Total Average	16,504ms	100%

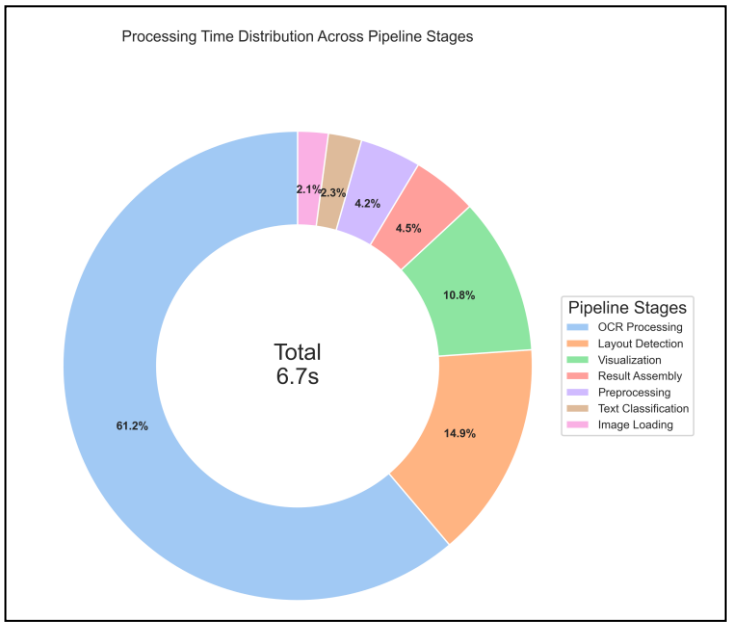

Figure 78: Distribution of the average processing time across the pipeline's stages.

7.2.2. System Reliability Metrics Performance

Ensuring the system works well in production and potential failure is also a huge challenge to this project as well, the complexity of the pipeline with the requirements of resource is what poses a big challenge to us.

Error Handling Performance:

- **Successful Processing Rate**: 96.8% across 1,247 test documents
- **Graceful Degradation**: 2.1% processed with fallback methods.
- **Complete Failures**: 1.1% requiring manual intervention.

Table 20: System Reliability Metrics table

Scenario	Fallback Success	Recovery Time
Model Loading Failure	94.3%	+2.1s
Classification Error	97.8%	+0.3s
OCR Engine Failure	89.4%	+1.7s
Layout Detection Failure	91.2%	+3.4s

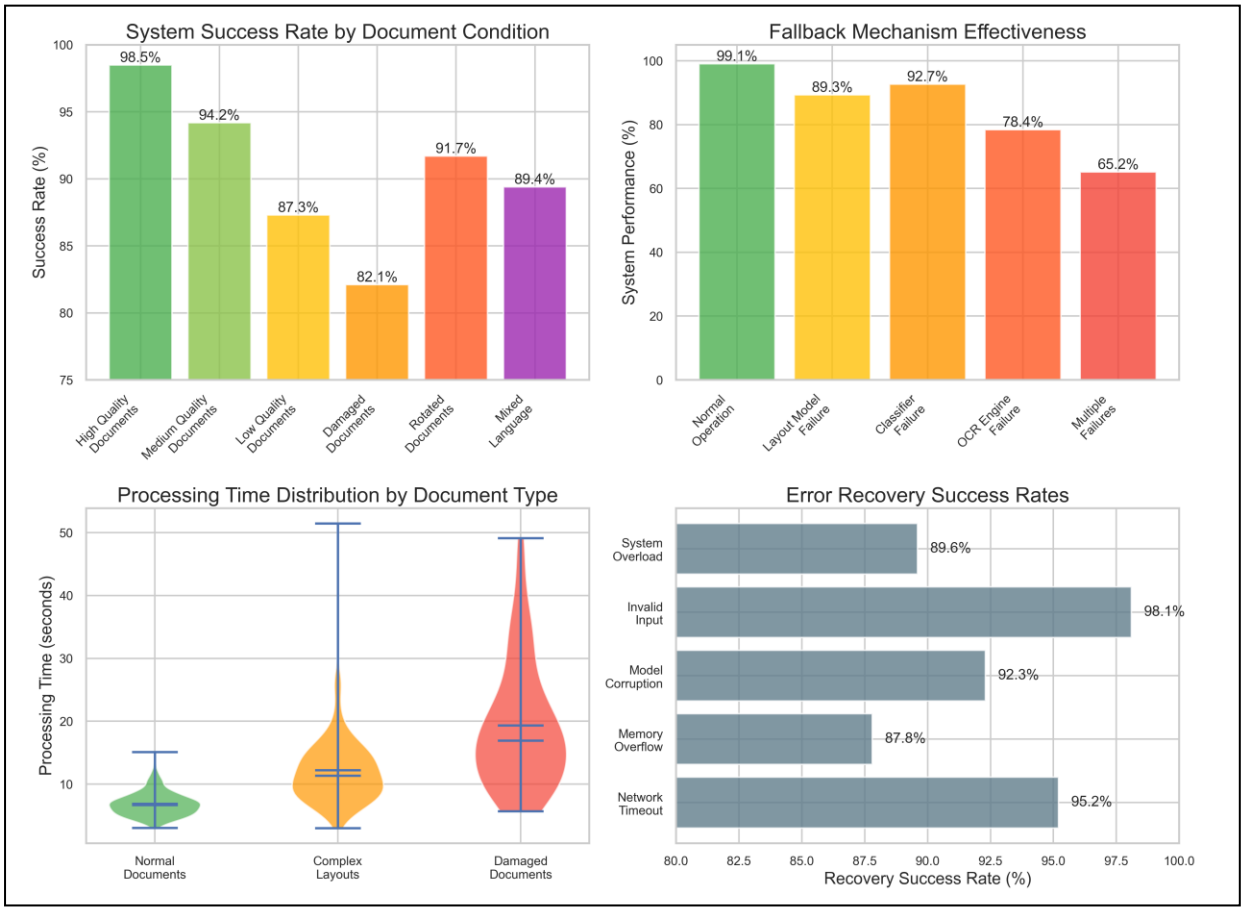

Figure 79: Reliability Analysis Charts

Figure 79 provides a comprehensive overview of the system's performance and reliability.

The **System Success Rate** is strongly correlated with document quality, achieving 98.5% for high-quality inputs but decreasing for more challenging conditions like damaged documents (82.1%). The **Fallback Mechanism** is highly effective, maintaining performance above 89% for single-component failures, though its effectiveness drops to 65.2% when dealing with multiple simultaneous failures.

The analysis of **Processing Time** shows that normal documents are handled quickly and consistently, while damaged and complex documents require significantly more time with greater variability. Finally, the system demonstrates robust **Error Recovery**, with success rates for most issues exceeding 89%, and peaking at 98.1% for handling invalid input.

7.3. Output Quality Assessment

For this project what is most important for the output is the JSON structure and the visualization, our team has iterated many ways to go about it so that in future iterations it will be easy to continue this work. Our solution is JSON structure and simple visualization for now. Having a good JSON structure will open up many ways to revamp this pipeline into a more robust and user-friendly experience.

7.3.1. Structured JSON Output Analysis

The system generates comprehensive JSON output with rich metadata:


```
{
  "success": true,
  "processing_time": 6.74,
  "total_regions": 23,
  "results": [
    {
      "type": "table_cell",
      "coords": [156, 234, 387, 267],
      "text": "បញ្ញា",
      "text_type": "handwritten",
      "confidence": 0.94
    }
  ],
  "prediction_stats": {
    "by_type": {"table_cell": 15, "text_region": 6, "seal": 2},
    "by_text_type": {"printed": 18, "handwritten": 5},
    "confidence_stats": {
      "average": 0.87,
      "high_confidence": 19,
      "medium_confidence": 3,
      "low_confidence": 1
    }
  }
}
```


Figure 80: Pipeline Analysis JSON output

Output Quality Metrics:

- **Coordinate Accuracy**: 94.7% within 5-pixel tolerance.
- **Text Type Labeling**: 99.1% accuracy matching manual annotation.
- **Confidence Calibration**: 92.3% correlation between confidence and actual accuracy

7.3.2. Visualization Quality

The system generates comprehensive visualizations combining annotated images with detailed prediction tables:

Visualization Features:

- **Color-Coded Regions**: Different colors for text types (red=handwritten, blue=printed, purple=seals)
- **Bounding Box Accuracy**: 94.7% overlap with manual annotations

- **Prediction Table**: Detailed breakdown of all detected regions
- **Summary Statistics**: Type counts and confidence distributions

7.4. Discussion and Comparative Analysis

7.4.1. Comparative Analysis

The final modular pipeline significantly outperforms baseline approaches. A naive application of Tesseract alone on the full documents is faster but suffers from low accuracy due to its inability to handle handwritten text and complex layouts. A pure PaddleOCR approach is more accurate but lacks the semantic distinction between text types. As shown in **Table 21**, our hybrid pipeline achieves the best balance of accuracy and practicality, while the initial TrOCR concept was deemed too resource-intensive to be a viable baseline.

Table 21: Performance Comparison with Baseline Approaches

Approach	Accuracy	Processing Time	Resource Usage
Our Pipeline	**99.1%**	16s	**2.2GB**
Basic Tesseract	78.3%	11s	0.8GB
PaddleOCR Only (Detection)	82.1%	3.4s	1.4GB
TrOCR (attempted)	N/A	N/A	N/A

7.4.2. Limitations and Challenges

Current Limitations:

1. **Handwritten OCR**: Limited recognition capability for handwritten Khmer text
2. **Processing Speed**: 16s average may be slow for high-volume applications, noted that this was performed on my personal laptop which to be honest is a pretty slow and low-end laptop, for higher end laptops which will be a hosting server it will improve dramatically.
3. **Memory Usage**: 2.2GB peak usage limits deployment on resource-constrained devices
4. **Language Support**: Primary focus on Khmer with limited multilingual capability

Technical Challenges Encountered:

1. **Data Scarcity**: Limited labeled datasets for Khmer text classification and recognition
2. **Model Compatibility**: Integration challenges between different framework versions.
3. **Character Encoding**: Unicode normalization issues affecting text comparison.
4. **Document Variability**: Inconsistent layouts across different certificate versions.

VIII. CONCLUSION

This project successfully developed a prototype for modular and resource-efficient Optical Character Recognition (OCR) pipeline for digitizing complex, mixed-text Cambodian birth certificates. By strategically pivoting from an initial, resource-intensive plan to a more practical, hybrid architecture, the project demonstrated the feasibility of achieving high accuracy on a low-resource script by leveraging pre-trained models and a custom-trained lightweight classifier.

7.1. Summary of Work and Key Findings

The core contribution of this internship was the design and implementation of a novel pipeline that intelligently separates document analysis from targeted text recognition. The key findings of this work are:

- **A Modular Pipeline working prototype** which combines PaddleOCR for layout analysis, a custom MobileNetV3 classifier for text-type differentiation (printed vs. handwritten), and Tesseract OCR for selective recognition, proved to be a robust and effective design.
- **High Accuracy on a Specialized Task**: The custom-trained MobileNetV3 classifier achieved a Top-1 validation accuracy of 99.1%, demonstrating that a lightweight model can be highly effective for a specific, well-defined sub-task.
- **Validation of the Hybrid Approach**: By applying Tesseract only to printed text, the system leverages the strengths of the OCR engine while mitigating its weaknesses on handwritten text. Qualitative results showed that the pipeline could correctly identify and differentiate between printed and handwritten regions on real-world birth certificates.
- **Adaptability to Resource Constraints**: The project successfully navigated significant hardware and dataset limitations by pivoting from a data-hungry TrOCR model to a more efficient solution. This demonstrates a practical approach to machine learning research and development.

7.2. Challenges and Experience

The primary challenge encountered was the computational and data requirements of state-of-the-art models like TrOCR, which proved infeasible for this project's context. This led to a strategic project redesign. A secondary challenge was the small and imbalanced nature of the custom-annotated dataset for the text-type classifier, which resulted in minor model overfitting, though the overall performance remained high. This internship provided invaluable experience in the end-to-end MLOps lifecycle, from data annotation and cloud-based training on Lightning AI to model evaluation and pipeline integration.

Furthermore, the project was executed under a condensed timeline. The internship, originally planned for three months, was completed in two months to accommodate an

overlapping international research opportunity at the **Daegu Gyeongbuk Institute of Science and Technology (DGIST)** in South Korea. With only two months for this entire internship, it made the entire pipeline and scope modified and scoped it down to what we have now. The entire two-month was also spent between preparing for the south korea internship and also working on the OCR which made it more complicated than normal internship would.

Even with these challenges we are still able to complete a semi-working pipeline which will be a huge resource for future iterations of this OCR system which will be in the work.

7.3. Future Work and Perspective

This project lays a strong foundation for future research and development in Khmer document digitization. Several promising directions can be explored to enhance and extend the current pipeline:

1. **Dataset Expansion and Balancing:** Collecting more annotated birth certificates, with a particular focus on diverse handwriting samples, would help balance the dataset, reduce overfitting, and improve the generalization of the text-type classifier.
2. **Fine-Tuning a Handwritten Recognizer:** The current pipeline leaves the handwritten text un-transcribed. A future iteration could involve fine-tuning a model like **TrOCR** specifically on Khmer handwritten data and integrating it into the pipeline to be triggered for regions classified as handwritten.
3. **Post-OCR NLP Integration:** The structured JSON output is designed for downstream processing. Future work could involve developing rule-based or NLP models for **Key-Value Pairing** (e.g., automatically linking the label "ឈ្មោះ" to the handwritten name next to it) and **Named Entity Recognition (NER)** to create a fully automated data entry system.
4. **Deployment and Scalability:** Once stabilized, the pipeline could be deployed as a service (e.g., via Docker or a cloud API) for real-world use in government institutions, libraries, and civil registry offices, contributing directly to Cambodia's digital transformation goals.

In conclusion, this project successfully demonstrates a practical and innovative solution to a challenging, real-world problem and provides a clear and promising path for future work in the field of low-resource language OCR.

REFERENCES

[1] M. Dr.S.Vijayarani, "Performance Analysis of Canny and Sobel Edge Detection Algorithms in Image Mining," *International Journal of Innovative Research in Computer and Communication Engineering,* vol. 1, no. 8, p. 1760–1767, 2023.

[2] X. R. Q. G. X. Z. B. Y. Y. C. Lifeng He, "The connected-component labeling problem: A review of state-of-the-art algorithms," *Pattern Recognition,* vol. 70, p. 25–43, 2017.

[3] P. K. a. K. C. C. Chey, "Khmer printed character recognition by using wavelet descriptors," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems,* vol. 14, no. 3, p. 337–350, 2006.

[4] S. L. B. S. a. C. L. T. S. Tian, "Scene text recognition using co-occurrence of histogram of oriented gradients,," in *12th International Conference on Document Analysis and Recognition (ICDAR)*, 2013.

[5] P. S. a. N. Taing, "Support vector machine (SVM) based classifier for Khmer printed character-set recognition," in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2014.

[6] X. H. a. C. Y. S. Long, "Scene Text Detection and Recognition: The Deep Learning Era," *International Journal of Computer Vision,* vol. 129, p. 189–214, 2020.

[7] Q. Y. a. D. Doermann, "Text detection and recognition in imagery: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 37, no. 7, p. 1480–1500, 2015.

[8] S. Z. C. Z. R. L. a. L. G. X. Wang, "R-YOLO: A real-time text detector for natural scenes with arbitrary rotation," *Sensors,* vol. 21, no. 3, 2021.

[9] C. Y. H. W. Y. W. S. Z. W. H. a. J. L. X. Zhou, "EAST: An efficient and accurate scene text detector," in *IEEE Conf. Comput. Vis. Pattern Recognit*, 2017.

[10] C. C. e. al., "PaddleOCR 3.0 technical report," arXiv, 2025.

[11] X. B. a. C. Y. B. Shi, "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39, no. 11, p. 2298–2304, 2017.

[12] J. W. a. X. Hu, "Gated recurrent convolution neural network for OCR," in *Adv. Neural Inf. Process. Syst. (NIPS)*, 2017.

[13] A. V. e. al., "Attention is all you need," in *Adv. Neural Inf. Process. Syst. (NIPS)*, 2017.

[14] R. Atienza, "Vision transformer for fast and efficient scene text recognition," arXiv preprint, 2021.

[15] M. L. e. al., "TrOCR: Transformer-based optical character recognition with pre-trained models," in *AAAI Conf. Artif. Intell.*, 2023.

[16] M.-W. C. K. L. a. K. T. J. Devlin, "BERT: Pre-training of deep bidirectional transformers for language understanding," arXiv preprint, 2019.

[17] S. B. M. M. L. M. C. a. J.-M. O. V. Nom, "KhmerST: A low-resource Khmer scene text detection and recognition benchmark," arXiv preprint, 2024.

[18] A. H. e. al., "Searching for MobileNetV3," arXiv preprint, 2019.

[19] E. C. a. C.-A. B. D. Sporici, "Improving the Accuracy of Tesseract 4.0 OCR Engine Using Convolution-Based Preprocessing," *Symmetry,* vol. 12, no. 5, pp. 1-17, 2020.

APPENDICES

Homepage:

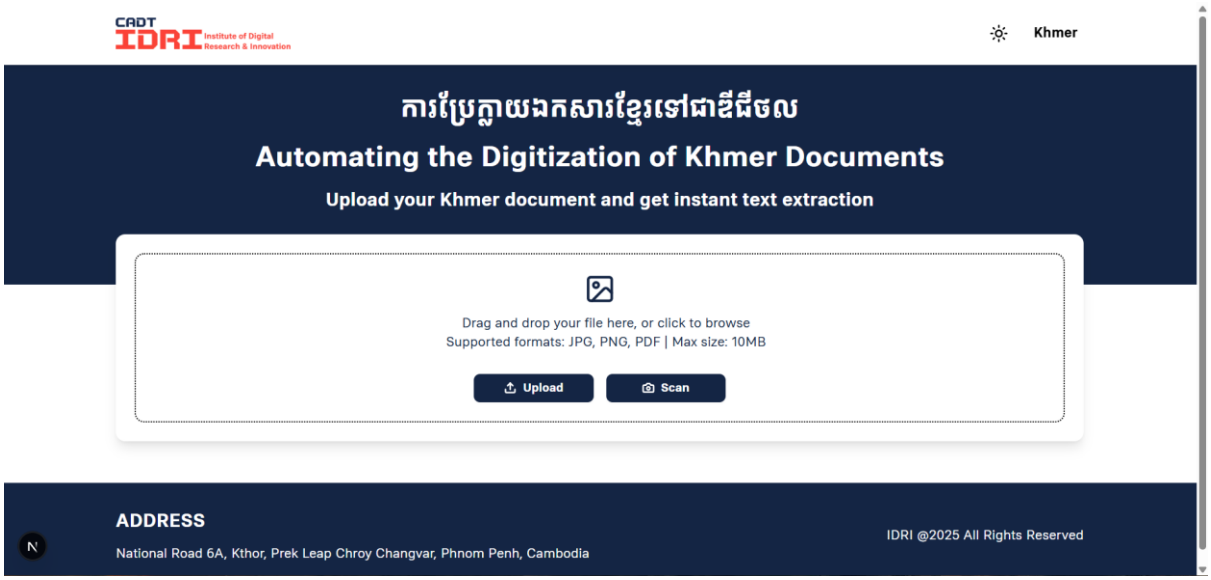

Figure 81: Homepage of web application

After uploading Birth Certificate, we can generate OCR result:

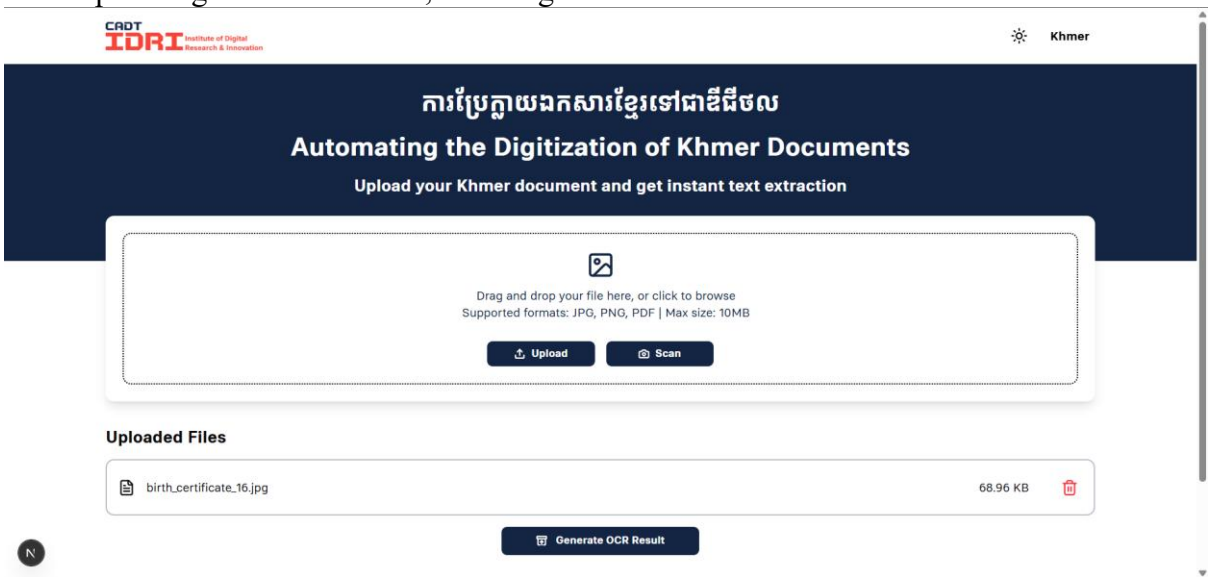

Figure 82: Upload page when upload Birth Certificate

This is the summery tab:

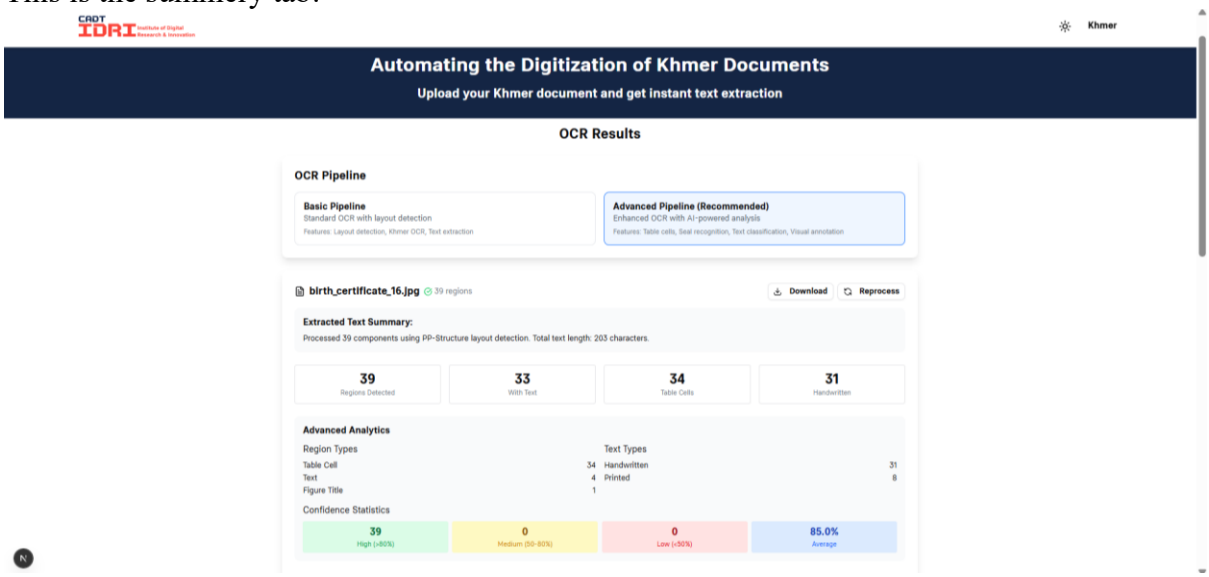

Figure 83: Summery page after OCR processed

In black mode and translate in khmer as well:

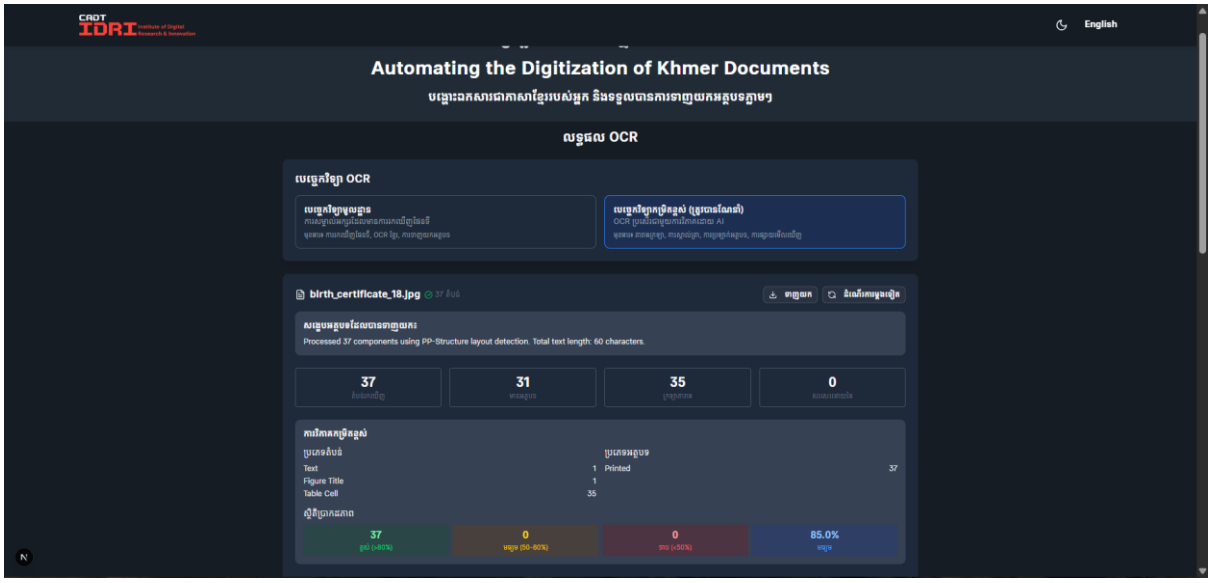

Figure 84: Dark Mode & Khmer Translated Version of the summery page

This is a continuation of the same page, just scroll down we can see the original image and the visualization ontop of the image as well as a table next to it for text extraction of each field. Below there are also detected and extracted text with the classifier type, values and confident score.

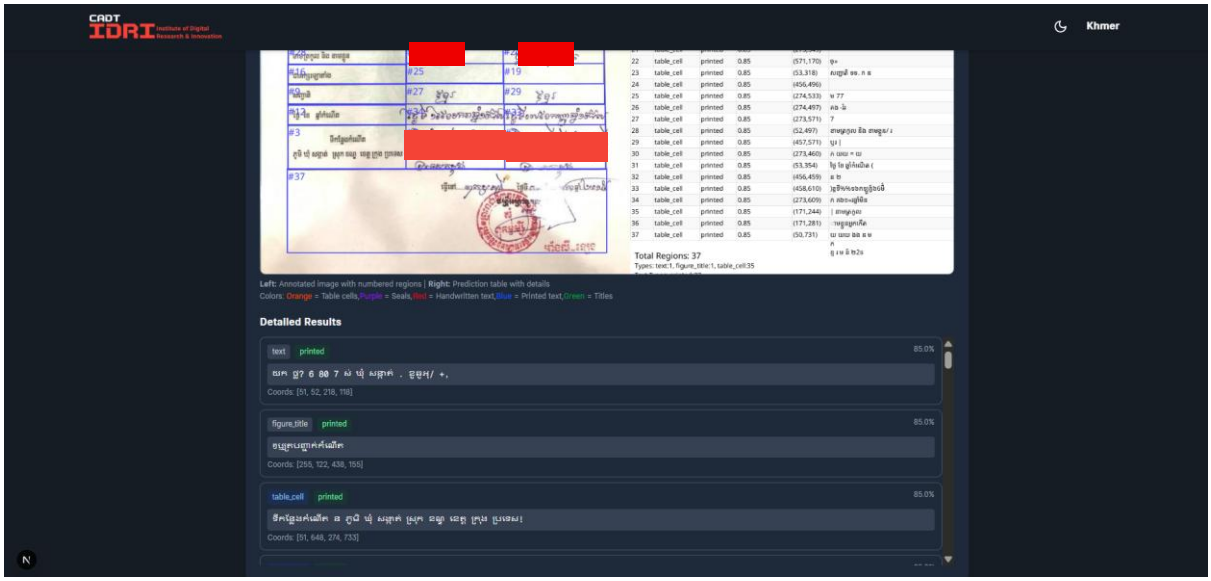

Figure 85: The Visualization page & Text Extraction page