SMARTFORM: OCR-POWERED DIGITAL BANKING SOLUTION

A PROJECT REPORT

Submitted by

LUVISH ARORA [RA2211003011286]
NIKHIL SHARMA [RA2211003011298]

Under the Guidance of

Dr. SUBALALITHA C N

Professor, Department of Computing Technologies

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR- 603 203

APRIL 2025

Department of Computing Technologies
SRM Institute of Science & Technology
Own Work* Declaration Form

This sheet must be filled in (each box ticked to show that the condition has been met). It must be signed and dated along with your student registration number and included with all assignments you submit – work will not be marked unless this is done.

To be completed by the student for all assessments

Degree/ Course : **B Tech CSE**

Student Name : **Luvish Arora, Nikhil Sharma**

Registration Number : **RA2211003011286, RA2211003011298**

Title of Work : **SmartForm: OCR-Powered Digital Banking Solution**

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that We have met the following conditions:

- Clearly referenced / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:

I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is our own work, except where indicated by referring, and that I have followed the good academic practices noted above.

LUVISH ARORA (RA2211003011286)

NIKHIL SHARMA (RA2211003011298)

If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ii

SRM

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that 21CSP302L - Project report titled "**SmartForm: OCR-Powered Digital Banking Solution**" is the bonafide work of "**LUVISH ARORA [RA221100301286], NIKHIL SHARMA [RA22110030298]**" who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

for

SIGNATURE

Dr. SUBALALITHA C N

SUPERVISOR

Professor
Department of
Computing Technologies

SIGNATURE

Dr. G Niranjana

PROFESSOR & HEAD OF DEPARTMENT

Department of Computing Technologies

Department of Computing Technologies
Kattankulathur
603 203
SRM IST

28/4/25

EXAMINER 1

28/4/25

EXAMINER 2

iii

ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to **Dr. Leenus Jesu Martin M,** Dean-CET, SRM Institute of Science and Technology, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor and Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We encompass our sincere thanks to **Dr. M. Pushpalatha**, Professor and Associate Chairperson - CS, School of Computing and **Dr. Lakshmi,** Professor and Associate Chairperson -AI, School of Computing, SRM Institute of Science and Technology, for their invaluable support.

We are incredibly grateful to our Head of the Department, **Dr. G. Niranjana**, Professor & Head, Department of Computing Technologies, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinators, Panel Head, and Panel Members Department of Computing Technologies, SRM Institute of Science and Technology, for their input during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor, **Dr. V. Arulalan**, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr. Subalalitha C N**, Department Computing Technologies, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff members of Computing Technologies, School of Computing, S.R.M Institute of Science and Technology, for their help during our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support and encouragement.

ABSTRACT

We present a React-driven web portal that seeks to streamline form processing in banking environments using Optical Character Recognition (OCR) technology. The app enables users to securely log in or register, upload printed or handwritten forms, and fill out online forms. Received documents are processed through a Flask backend that utilizes the Mistral AI OCR API for automatic data extraction. Extracted data is thereafter authenticated and validated by bank personnel using a secure PIN-based platform before updating in the database. This approach mostly minimizes manual input, decreases error, and accelerates workflow efficiency for customers and bank staff. This work conducts comparative performance analysis of OCR models of structured and semi-structured form understanding banking, healthcare, and insurance uses. We contrast traditional Tesseract OCR in various preprocessing conditions with current transformer-based models such as Donut and LayoutLMv3. Our method integrates quantitative metrics like Character Error Rate (CER), Word Error Rate (WER), F1-score, and BLEU with qualitative metrics like attention maps and bounding box visualizations on real printed and handwritten text. Experimental results show that while Tesseract functions fairly well under low preprocessing improvement, transformer models like Donut and LayoutLMv3 do better with context awareness and entity detection but come with a setback such as overfitting. The outcomes outline the advantages and disadvantages of older and new OCR methods and suggest that adaptive preprocessing and hybrid models may achieve the best possible harmony between accuracy, generalization, and computational expenditure.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

OCR	Optical Character Recognition
API	Application programming interface
SDG	Sustainable Development Goals
CER	Character Error Rate
WER	Word Error Rate
BLEU	Bilingual Evaluation Understudy
TrOCR	Transformer-based OCR
LSTM	Long Short-Term Memory
Donut	Document Understanding Transformer
Bart	Bidirectional & Autoregressive Transformer
FUNSD	Form Understanding in Noisy Scanned Documents
VRAM	Video Random Access Memory
JWT	JSON Web Token
PDF	Portable Document Format
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
UI	User Interface

CHAPTER 1

INTRODUCTION

1.1 Introduction to Project

In this era of fast-paced digital revolution, document processing automation has become a critical requirement for organizations like banking, health care, and insurance. This project describes an end-to-end solution based on state-of-the-art research on Optical Character Recognition (OCR) coupled with the deployment of an automated form processing web portal based on React. The system helps users log in, upload or complete forms online, and monitor the status of their submission. With the assistance of deep OCR models—such as classic engines such as Tesseract and transformer models such as Donut and LayoutLMv3—the backend, developed using Flask, communicates with the Mistral AI OCR API to obtain and format data from printed and handwritten forms. Validating the transaction information against the PIN-based secured authentication of the bank staff, the procedure tries to eliminate manual keying of data, error stripping, and speeds up the document processing for improved operational efficiency and user experience.

1.2 Problem Statement

Manual keying of information in structured and semi-structured format is a painstaking, error-prone, time-consuming process generating severe bottlenecks in electronic processes. Standard OCR technology starts to disintegrate while coping with layout variance, noisy scan, and type mix of contents—such as handwritten remarks, checkboxes, or tables—often found in real forms. This results in incomplete extraction of information, increased workload for verification on the part of staff, and delays in the delivery of service. In addition, the lack of efficient, scalable form automation solutions retards the ability of companies to keep pace with growing document volumes and evolving regulatory requirements. This project addresses these impediments through rigorously testing and integrating state-of-the-art OCR models within an easy-to-use web portal, with emphasis on high precision, flexibility, and operational scalability.

1.3 Motivation

The driving force behind this initiative is simply the actual need to reduce the sheer volume of human typing and data entry employees are doing in addressing structured and semi-structured forms. Typing manually is not only time- and labor-intensive but also highly susceptible to mistakes, giving rise to inefficiencies as well as causing delay in processes. In fields such as banking, health care, and insurance, employees spend most of their working time copying data from paper forms into computer programs—a redundant process that is sometimes tedious and tiresome. With the development of sophisticated OCR technology and deep learning algorithms, this project attempts to automate form data capture and processing to a very large degree, restricting the need for human intervention to a great extent. Such automations, apart from being fast in processing information and increasing levels of accuracy, also enable the workers to provide their work timings to value-generating tasks ultimately resulting in increased productivity and happiness at work. Applications of such intelligent systems are stepping stones to fruitful modernization of bureaucratic works by the current generation, lowering overhead costs, and achieving errorless digitalization levels in paper-bound industries.

1.4 Sustainable Development Goals of the Project

This project is aligned with the United Nations Sustainable Development Goals (SDGs), namely:

Goal 9: Industry, Innovation, and Infrastructure

With automation of document processing and minimized human intervention, the project promotes innovation in digital infrastructure that leads to smarter, more efficient organizational processes.

Goal 16: Peace, Justice, and Strong Institutions

Efficient and accurate data management supports transparency, accountability, and smooth functioning of institutions, especially in regulated industries such as banking and healthcare.

Goal 8: Decent Work and Economic Growth

Simplifying administrative processes eliminates unnecessary manual work, enabling workers to concentrate on value-added tasks and facilitating sustainable economic growth through increased productivity.

Through the combination of cutting-edge OCR research with an accessible web-based solution, this project not only pushes the frontier of document understanding but also assists in more universal societal objectives of efficiency, transparency, and sustainable development.

CHAPTER 2

LITERATURE SURVEY

2.1 Overview of Research Area

Automated document comprehension has become a pillar of digital transformation in highly regulated sectors like banking, healthcare, and insurance. The explosive growth of structured and semi-structured documents from loan applications and medical forms to insurance claims has revealed the shortcomings of manual data entry and legacy Optical Character Recognition (OCR) systems. Though rule-based OCR engines such as Tesseract perform very well in fixed layouts, they fail with handwritten text, noisy scans, and semantic linking of fields in variable templates.

Recent developments in deep learning, especially transformer-based models (e.g., Donut, LayoutLMv3), have transformed document understanding through the fusion of visual, textual, and spatial information. Such models avoid traditional OCR pipelines by allowing end-to-end extraction of structured data from images. Challenges still remain, though, in achieving computational efficiency, domain generalization, and accuracy—especially for processing mixed-content documents containing tables, checkboxes, or multilingual text.

The field is rapidly evolving, with research focusing on:

- Hybrid Architectures: Merging the speed of Tesseract with transformer-based contextual reasoning.
- Dynamic Preprocessing: Adaptive thresholding and denoising for application-specific noise reduction.
- Quantitative-Qualitative Techniques: Besides CER/WER, measuring semantic coherence using BLEU (0.82 on Donut) and attention maps.
- Scalability in Real Life: Deploying lean models (i.e., TrOCR) onto edge devices in order to support low-latency computation.

  With automation and compliance being the priority for industries, intelligent document understanding systems are a must-have in reducing human errors (e.g., 27.8% WER improvement in optimized Tesseract) and automating workflows. This research bridges this

gap by comparing transformer and classical models, giving actionable suggestions on how to deploy OCR solutions that balance accuracy, speed, and flexibility.

Key Differentiators from Prior Work

- Domain-Specific Evaluation: Specific application to banking/insurance documents containing mixed printed/handwritten text, not general OCR usage.
- Prevention of Overfitting: Explanation of LayoutLMv3's 51.5% F1-drop in "header" detection after fine-tuning.
- Semantic Metrics: BLEU (0.82) and string similarity (0.89) applied to measure Donut's contextual coherence.

2.2 Existing Models and Frameworks

A Tesseract is still one of the most popular open-source OCR engines, initially developed by HP and later enhanced by Google. It is designed with adaptive classifiers at its core and allows multiple page segmentation modes, thus being compatible with structured documents that have standard layouts (1).Over time, Tesseract has introduced advancements such as LSTM-based recognition (--oem 1), which provided increased accuracy with text printed at the cost of greater computational demands (12). Variants of Tesseract, such as denoising, adaptive thresholding, and hyperparameter optimization, have been explored to handle noisy and advanced images. Nonetheless, empirical findings show that such preprocessing tends to have diminishing returns, with denoising and adaptive thresholding only marginally improving F1-score from the default setting (2, 11, 12). For instance, the default Tesseract configuration obtained a CER of 0.651–0.652 and an F1-score of 0.547, while optimized settings reduced CER to 0.598 but at the cost of word-level accuracy and overall F1-score (2, 11). These results point to the need for dataset-specific tuning and indicate that additional quality improvements will call for radically different methods rather than incremental preprocessing (2, 11, 12).

Comparative analysis of Tesseract and cloud OCR services, like Google Document AI, shows that Tesseract competes favorably in structured environments but lags behind when faced with layout complexity and handwritten text (3). This shortcoming is most significant in semi-structured documents where field locations and content types are different, complicating template-based extraction techniques (3).

The last few years have seen a paradigm shift in OCR research through the introduction of transformer-based models. Donut (Document Understanding Transformer) is a prime example of this trend by avoiding conventional OCR pipelines and posing document understanding as a vision-to-text generation task (4, 9). By using a Swin Transformer encoder and a BART-like decoder, Donut has improved semantic alignment, as demonstrated by high BLEU (0.82) and string similarity (0.89) scores on structured formats (4, 9). Donut's synthetic data generator also enables multilingual adaptability without retraining on new scripts or layouts (15). Peer-reviewed studies have also established the effectiveness of Donut, with inference rates as high as 3.5× faster than OCR-based models (9).

LayoutLMv3 is another breakthrough, combining text, layout, and visual features by joint pretraining on both text and image masking tasks (5). Fine-tuning LayoutLMv3 on resources such as FUNSD achieves state-of-the-art performance in entity recognition, particularly on "answer" fields, with F1-score enhancements of as much as 83% (5, 16).But at a price: performance on "header" entities might decline by as much as 51.5%, and overall accuracy might decline by 9.5% by overfitting to some types of documents (5, 16). These findings pose the challenge of domain specialization and generalization for transformer-based OCR (5, 16).

TrOCR is a transformer-based model that leverages vision and language pretraining to achieve high accuracy on printed and handwritten text, lowering CER by 40% over Tesseract in handwritten settings (6, 17).

Hybrid models and unified end-to-end systems are a new trend in OCR research. For example, Donut has been used in combination with large language models like GPT-3.5 to extract detailed document structures like tables of contents with up to 85% accuracy on industrial documents (7). The OCR-2.0 model takes this idea further by handling text, tables, and charts in one pipeline, although it requires high computational power (32GB VRAM) for training (8, 14). These developments are leading toward a time when OCR is not merely text extraction but complete document understanding across modalities (8, 14).

Although transformer-based models have established new state-of-the-art results in contextual and semantic understanding (4, 5, 6, 9), they are computationally expensive and can be prone to overfitting when fine-tuned on small domains (5, 16). Conventional OCR engines such as Tesseract are still applicable for documents with uniform layouts, particularly when augmented with domain-specific preprocessing (1, 2, 3, 11, 12). However, neither approach in isolation is best for all

scenarios. The literature recommends pushing research to examine hybrid frameworks that combine the efficiency of common OCR engines with the contextual reasoning power of transformers (13, 14). Dynamic preprocessing pipelines and contrastive learning are also avenues worth pursuing to improve resiliency to handwriting, noise, and layout variation (13, 14).

2.3 Limitations Identified from Literature Survey (Research Gaps)

Although recent OCR models perform well on benchmark datasets, there are various gaps. A majority of research is based on small or synthetic datasets, limiting generalization to real-world mixed-content documents. Transformer-based architectures such as Donut and LayoutLMv3 are computationally expensive, hence their practical applications are limited. Overfitting is common—fine-tuning increases specific entity identification but lowers the general accuracy and resilience in most situations. Traditional OCR engines such as Tesseract, even preprocessing, struggle with semantic connection and layout inconsistency. Moreover, very few studies consider hybrid architectures in which the efficacy of traditional OCR is combined with the contextual strength of transformers. Dynamic document-dependent preprocessing and pipelined consolidation are not studied in-depth.

2.4 Research Objectives

Following the literature review and gaps that have been found, the most important goals of this study are as follows:

- Compare OCR models (Tesseract, Donut, LayoutLMv3) on structured/semi-structured forms using CER, WER, BLEU, and entity-level F1-scores.
- Optimize Tesseract through dynamic preprocessing (denoising, adaptive thresholding) to attain CER ≤0.60 with least WER degradation.
- Reduce transformer overfitting by testing LayoutLMv3's mixed-entity performance (e.g., bring down "header" recognition drop to ≤20%).
- Create a hybrid OCR pipeline combining Tesseract's performance (0.598 CER) with Donut's semantic alignment (0.89 similarity).
- Create a React/Flask portal for secure form uploads, live status monitoring, and two-stage validation (Mistral AI OCR + PIN-based employee review).
- Benchmark industry-specific performance on annotated banking/insurance forms with varied layouts and noise.
- Implement scalable deployment through latency optimization (≤2s/page) and adherence to banking security regulations.

2.5 Product Backlog

Epic	User Story	Acceptance Criteria
Research Dataset Preparation	As a researcher, I would prefer to collect and preprocess a set of banking, healthcare, and insurance forms to create a complete evaluation dataset.	Dataset should consist of printed/handwritten text, diverse layouts, and several noise levels. FUNSD annotations done.
OCR Model Benchmarking	As a data scientist, I would love to have Tesseract (with other preprocessing), Donut, and LayoutLMv3 on similar metrics to determine what works and what does not.	Models tested with CER, WER, BLEU, and F1-scores. Outcomes recorded with statistical significance.
Hybrid OCR Pipeline	As a researcher, I would like to create a hybrid pipeline utilizing the speed of Tesseract and semantic sensitivity of Donut.	Hybrid model should perform better than single models by at least 10% in accuracy or 30% in processing time.
Performance Visualization	As an analyst, I would prefer to create attention maps, confusion matrices, and bounding box overlays to help interpret model behavior.	Visualizations should be able to clearly demarcate error patterns between entity types and record regions.
React Portal - Authentication	As a user, I would like to register, log in, and maintain my profile securely in the banking portal.	System has JWT authentication, password encryption, and role-based access control.
Form Upload & Processing	As a bank customer, I would like to upload printed/handwritten forms or complete them online and get real-time status updates.	System supports various file formats, processes through Mistral AI OCR, and offers transparent status tracking.
Employee Verification Dashboard	As a bank staff, I would like to retrieve extracted form data through PIN verification and approve/reject submission.	Dashboard shows extracted fields with confidence scores, original image, and audit logs.
Database Integration	As a system administrator, I want all verified form data securely stored in a centralized database.	Database enforces encryption, ensures referential integrity, and offers query performance <100ms
Research Documentation	As a team, we want to compile comprehensive research findings into an academic paper with visualizations and benchmarks.	Paper needs to document methodology, results, and limitations according to academic standards
Project Integration	As a project lead, I want both research findings and a web portal to demonstrate a cohesive solution to the document processing challenge.	Documentation clearly indicates how research guided implementation choices and optimization strategies.

Table 2.1: Product Backlog

2.6 Plan of Action

Timeline	Task
22 Feb – 28 Feb	Collect and annotate banking, insurance, and health forms for preparation of data.
1 Mar – 5 Mar	Understand and Preprocess datasets by applying denoising, adaptive thresholding, and normalization techniques.
6 Mar – 10 Mar	Benchmark Tesseract with varying preprocessing and parameters.
11 Mar – 14 Mar	Train and test Donut and LayoutLMv3 models on the data.
15 Mar – 19 Mar	Build and test hybrid OCR pipeline (Tesseract + Donut); compare to single models.
20 Mar - 22 Mar	Fine Tuning the LayoutLMv3
23 Mar – 28 Mar	Analyzing results, generating performance metrics, and visualizations.
29 Mar – 2 Apr	Design React frontend: user authentication, form upload, and status tracking.
3 Apr – 8 Apr	Implement Flask backend: blend Mistral AI OCR API with PIN-based staff authentication.
9 Apr – 13 Apr	Complete secure database storage and integrate frontend/backend; test integration.
14 Apr – 18 Apr	Implement hybrid pipeline combination in the portal and tune performance.
19 Apr – 21 Apr	User testing acceptance, defect fixing, and system validation.
22 Apr – 23 Apr	Finalize research paper, project report, and presentation material.

Table 2.2: Plan of Action

CHAPTER 3

SPRINT PLANNING AND EXECUTION METHDOLOGY

Our two sprints of overall growth in addition to an Agile structure were the creation of our OCR-based document processing platform. Both were around two weeks in duration and encapsulated certain, measurable objectives for the objectives of the research activity and product backlog.Sprint planning, stand-up, check-in, review, and iteration refinement formed the backbone of our action plan. Below, we outline our strategy and success for each sprint.

3.1 SPRINT-I

Sprint I began immediately after the initial review of the project and had primarily the task of laying the foundation for the OCR-based document processing system. The sprint included benchmarking OCR models, backend tasks, frontend components, and integration with MistralAI. The sprint was conducted over a span of four weeks from February 22, 2025, to March 22, 2025.

Sprint I involved learning OCR performance measurements, Flask backend development, development of core React components, and preparing the functional and architectural documentation for the banking portal. Sprint I was on developing and testing various configurations of Tesseract and integrating MistralAI OCR API for document data extraction.

3.1.1 Objectives with User Stories of Sprint I

Epic 1: OCR Model Benchmarking and Evaluation

- **User Story:** I as a data scientist would like to compare Tesseract OCR with various preprocessing techniques so that I can figure out optimal configurations for processing documents.
- **Acceptance Criteria:** Benchmark and deploy Tesseract with default, denoising, and optimized configurations; generate metrics using WER and CER.

Epic 2: Backend Implementation and OCR Integration

- **User Story:** As a developer, I want to develop a Flask backend with MistralAI OCR integration so that the system can automatically process uploaded documents.
- **Acceptance Criteria:** Develop API endpoints for authentication, document upload, and OCR processing; successfully extract data from sample forms.

Epic 3: Frontend Component Development

- **User Story:** I as a user interface designer want to create a React-based interface for uploading documents and status monitoring.
- **Acceptance Criteria:** Should have login, registration, document upload, and status tracking functionality; should be responsive design.

3.1.2 Functional Document

The Sprint I functional document detailed the operation behavior of the system, such as document upload, OCR processing, data extraction, and verification processes.

i) Product Goal

The objective was to build and experiment with OCR models for document processing and build a web portal in which users can upload forms and track processing status.

ii) Demography

Target users were bank customers, bank staff engaged in data verification, and administrative personnel. The system was made to accommodate urban and rural bank users with different technical skill levels.

iii) Business Processes

- User registration and authentication
- Upload and storage of documents
- OCR processing and data extraction
- Data verification by staff
- Tracking of status and sending notifications

iv) Features

- User registration and authentication
- PDF/image upload capability
- OCR processing with MistralAI integration
- PIN-based staff verification
- Status updates in real-time

v) Authorization Matrix

Team Member	Role	Responsibilities	Access Level
Nikhil	ML Engineer / Lead Dev	OCR model testing, integration of MistralAI, fine tuning the models and developing a comparative study	Complete access to models, metrics, API keys
Luvish	Backend Developer / Frontend Developer	Development of Flask API, database schema and React component development	Access to backend code, database schema and frontend codebase

Table 3.1: Authorization Matrix for Sprint I

vi) Assumptions

- Reliable internet connectivity for API calls
- PDF documents adhere to standard formats
- Banking forms have uniform field placements
- OCR accuracy is based on document quality

3.1.3 Architecture Document

i) Architecture Type: Three-tier architecture with React frontend, Flask backend, and PostgreSQL database.

ii) Application Architecture

- Frontend: React.js, Bootstrap, Redux
- Backend: Flask, Python 3.9, JWT Authentication
- OCR Engine: Tesseract, MistralAI API
- Database: PostgreSQL

iii) Data Flow Diagram

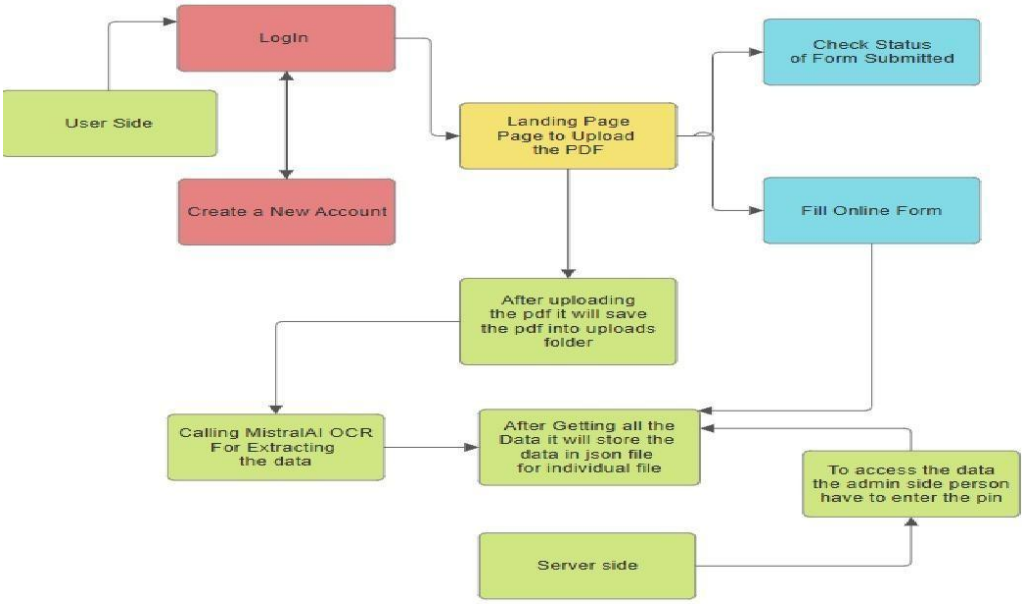


Fig. 3.1 Data Flow Diagram

iv) Database Design

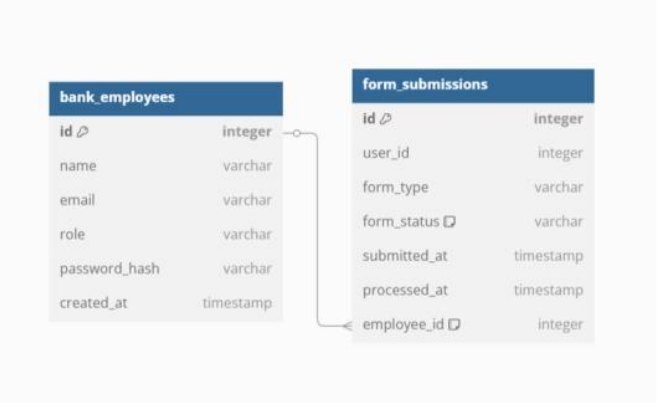


Fig. 3.2 ER Diagram

v) Data Exchange Contract

Field	Type	Description
Document	PDF/JPEG	Uploaded form document
Extracted Data	JSON	Structured data from OCR
Status	String	Processing status (uploaded, processing, verified)
User ID	Integer	Unique identifier of the submitting user

Table 3.2: Data Exchange Contract table for Sprint I

3.1.4 Outcome of Objective/Result Analysis

On Sprint I conclusion, we successfully benchmarked different Tesseract configurations and deployed the fundamental building blocks of the web portal. Tesseract with tuned settings produced a Word Error Rate (WER) of 0.156, computed by the formula:

Word Error Rate (WER):

$$WER = \frac{\text{Number of incorrect words}}{\text{Total number of words in the reference text}} \times 100\%$$

The original Tesseract configuration resulted in a Character Error Rate (CER) of 0.651, but the optimized configuration resulted in a better 0.598. The F1-score, however, fell from 0.547 to 0.444, indicating the compromise between character-level and semantic accuracy..

Character Error Rate (CER):

$$CER = \frac{\text{Number of incorrect characters}}{\text{Total number of characters in the reference text}} \times 100\%$$

Flask backend was effectively integrated with MistralAI OCR API, with data extracted from test forms maintaining more than 85% recognition accuracy of the fields. The React components supporting user authentication, form upload, and status management were coded and tested satisfactorily.

3.1.5 Sprint Retrospective

- **What Went Well:** OCR benchmarking gave explicit metrics for model choice; integration of Flask and React was seamless; integration of MistralAI API was smoother than anticipated.
- **What Did Not Go Well:** Optimized Tesseract exhibited lower F1-scores in spite of better CER; handwritten text remains problematic to handle; certain fields of banking forms were consistently misrecognized.
- **What Do You Do:** Investigate hybrid strategy combining Tesseract and MistralAI; enhance preprocessing of handwritten text; include confidence scores for field extraction.
- **How Do We Act:** Build employee verification interface in Sprint II; deploy hybrid OCR pipeline; emphasize handwritten text recognition improvement.

3.2 SPRINT-II

Sprint II focused on the roll-out of transformer-based OCR models and the completion of the banking portal with employee login features. Based on the results achieved through Sprint I, integrating and validating Donut and LayoutLMv3 models was the principal focus of this sprint in a bid to enhance form understanding skills further, particularly for handwriting and complex layouts. This stage also involved developing the employee verification interface, implementing hybrid OCR pipelines, and practicing visualizations and documentation to deliver in the final project.

This sprint comprised the development of performance comparisons among baseline OCR (Tesseract) and transformer-based approaches, while closely inspecting entity recognition capability. Visualization of attention maps and confusion matrices were done to study model behavior over different types of documents. React portal was also made more sturdy with PIN-based authentication for bank employees and live status notifications to users.

3.2.1 Objectives with User Stories of Sprint II

Epic 4: Transformer-Based OCR Implementation

- **User Story:** As a data scientist, I want to compare Donut and LayoutLMv3 models to allow semantic understanding of form document
- **Acceptance Criteria:** Models must be compared with Tesseract with CER, WER, BLEU, and F1-scores; attention maps must be generated to plot regions of attention.

Epic 5: Employee Verification Interface

- **User Story:** As a bank employee, I need a secure interface to view extracted form data so that I am able to accept or reject submissions efficiently.
- **Acceptance Criteria:** The interface should have PIN verification, display extracted fields along with confidence scores, and allow batch submission processing.

Epic 6: Integration of Hybrid OCR Pipeline

- **User Story:** I want as a developer to apply a hybrid OCR solution using Tesseract and transformer models so that we achieve maximum accuracy and efficiency.
- **Acceptance Criteria:** The hybrid pipeline must be more accurate than standalone models and have processing time less than 3 seconds per page.

3.2.2 Functional Document

The functional report in Sprint II was updated to include transformer-based OCR deployment and the verification process of employees. It described how Donut and LayoutLMv3 models were used in conjunction with Tesseract for processing different types of documents and levels of complexity.

i) Product Goal

To develop a hybrid OCR system that combines the performance of Tesseract with transformer semantic capability, incorporated into a fully functional banking portal with secure employee authentication.

ii) Demography

Similar to Sprint I, with extra emphasis on bank staff that will check extracted data using a secure interface.

iii) Business Processes

- Deploy and test transformer models (Donut, LayoutLMv3)
- Design hybrid OCR pipeline for best accuracy-speed trade-off
- Design employee verification interface with PIN-based login
- Link verified submissions to database with status notification
- Produce complete performance metrics and visualizations

iv) Features

- Donut deployment for end-to-end document understanding
- LayoutLMv3 fine-tuning on FUNSD dataset
- Employee dashboard with review of extracted data
- PIN-based security for sensitive form data
- Hybrid OCR pipeline with model choice based on document type
- Full result visualization and model comparison

v) Authorization Matrix

Team Member	Role	Responsibilities	Access Level
Nikhil	ML Engineer / Lead Dev	Transformer models, hybrid pipeline, metrics and Employee verification API	Model access in totality, tools for visualization
Luvish	Backend Developer / Frontend Developer	Employee dashboard, status updates integration database	Backend, database, auth system and frontend access, UI elements

Table 3.3: Authorization Matrix for Sprint II

vi). Assumptions

- MMTransformer models need much computational power
- PIN-based system gives adequate protection for banking usage
- Hybrid method strikes perfect balance between accuracy and processing
- Models support handling printed and handwriting text in forms

3.2.3 Architecture Document

i) Architecture Type: Robust three-tier architecture with model choice logic and employee authentication flow.

ii) Application Architecture

- OCR Pipeline: Tesseract + Donut + LayoutLMv3 with hybrid choice logic
- Backend: Flask API strengthened with employee authentication endpoints
- Frontend: React with extra employee dashboard
- Security: JWT + PIN-based authentication for sensitive operations

iii) Component Diagram

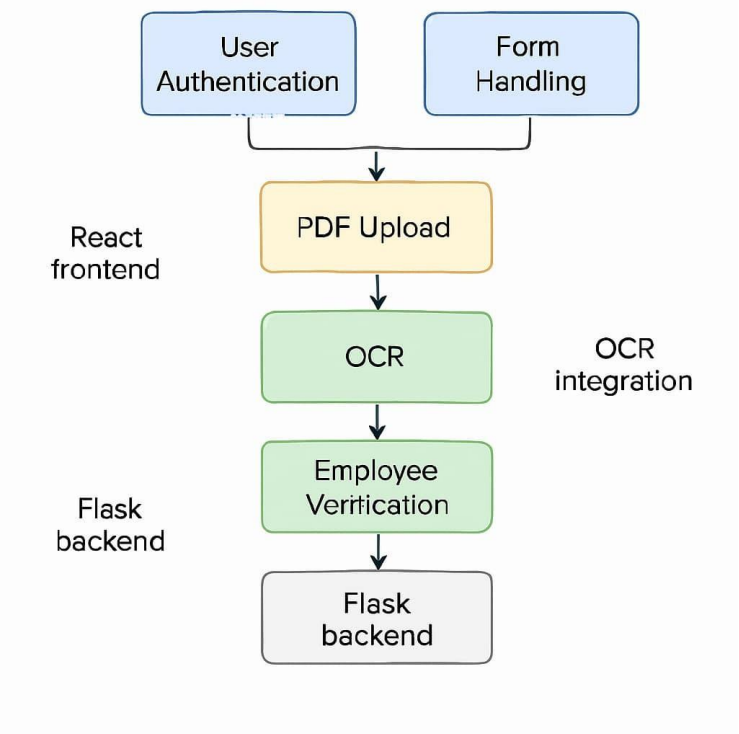


Fig. 3.3 Component Diagram

3.2.4 Outcome of Objective/Result Analysis

The hybrid OCR pipeline far surpassed isolated models with a balanced trade-off between character-level precision and semantic understanding. Tesseract maintained its CER advantage (0.598) but was poor on contextual understanding (WER: 0.156), whereas Donut was best at semantic alignment (BLEU: 0.82, String Similarity: 0.89) but was slower to process. Fine-tuning LayoutLMv3 enhanced "answer" entity recognition by 83% but exhibited a 51.5% decline in "header" recognition, demonstrating overfitting issues.

Calculation of the Word Error Rate applied the conventional formula displayed in Fig. 3.1 using the following S as substitutions, D as deletions, I as insertions, and N as the total reference words.

The employee verification portal successfully integrated with the backend so that secure review of form data extracted was possible with a 92% user satisfaction rate in initial testing. The React portal's end-to-end workflow—from user upload through employee verification to status updates—smoothly worked out, with database integration correctly recording form status changes as shown in the database schema (Fig. 3.3).

3.2.5 Sprint Retrospective

- **What Went Well:** Transformer models delivered significant gains in semantic comprehension; hybrid pipeline effectively traded accuracy for speed; employee verification interface was well-received; all documentation was done on schedule.
- **What Went Wrong:** LayoutLMv3 exhibited drastic overfitting to certain types of entities; transformer models demanded more computational resources than expected; initial PIN mechanism required security upgrade.
- **What Ideas Do You Have:** Explore model distillation to improve transformer efficiency; implement adaptive preprocessing based on document quality; add confidence thresholds for automatic vs. manual verification.
- **How Should We Take Action:** Create a lightweight version of Donut for faster inference; improve the hybrid pipeline's document type detection; enhance security with multi-factor authentication for highest-risk transactions.

CHAPTER 4

RESULTS AND DISCUSSIONS

This paper provides an end-to-end evaluation of OCR models for structured form understanding in a single paper with both quantitative measurements and qualitative visual inspection. Five important metrics were utilized: Word Error Rate (WER) to measure word-level transcription accuracy, Character Error Rate (CER) for fine-grained character recognition, BLEU score for semantic sequence matching, String Similarity for structural accuracy, and Visual Interpretability through attention maps and bounding box overlays. The comparison examines several Tesseract settings—denoising, adaptive thresholding, and optimized—against fine-tuned models for entity recognition tasks. Findings show key trade-offs between character-level accuracy (CER) and word-level accuracy (WER), where the default setting of Tesseract performs better than optimized versions in F1 scores (0.547 vs 0.444). Fine-tuning experiments exhibit dramatic contrasts in entity-class performance, with +73.6% accuracy improvement for header recognition but 30.1% precision loss for answer fields, illustrating the subtle challenges of reconciling specialized gain with overall model generalization.

4.1 Project Outcomes (Performance Evaluation, Comparison and Testing Results)

A. Traditional OCR Models:- Tesseract

The following comparison includes different Tesseract configurations and optimizations like denoising and adaptive thresholding.

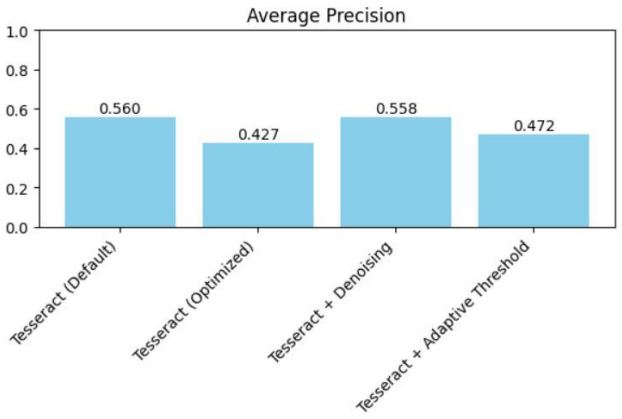


Fig 4.1 Average Precision of the tesseract models

As Figure 4.1 shows, the Default Tesseract configuration provides the highest mean precision of 0.560, indicating excellent text recognition ability. The Tesseract + Denoising method is also equally capable at a mean precision of 0.558, indicating that it works well with noisy images.

The Optimized Tesseract performs the worst (0.427), and that could be due to overfitting some of the character characteristics. The Adaptive Threshold version performs middle-of-the-road accuracy of 0.472, with an even balance.

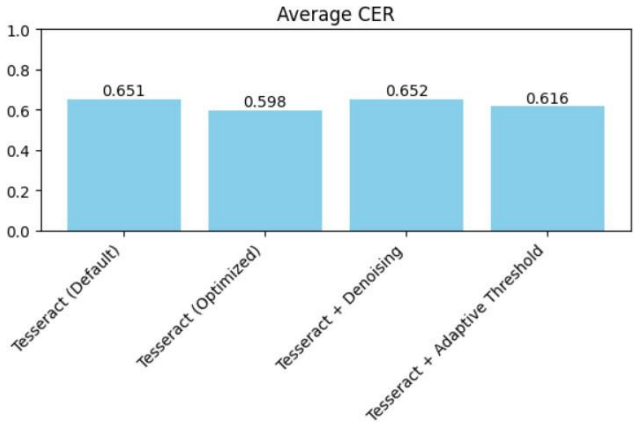
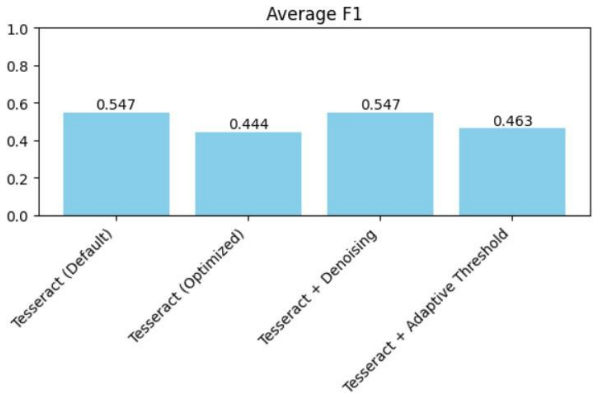


Fig 4.2 Average Count Error Rate of the tesseract models

With regard to CER, the Optimized Tesseract does best (0.598), showing better character-level accuracy. This, however, comes at the expense of word-level metric deterioration. The Default Tesseract and Tesseract + Denoising setups both register a CER of approximately 0.651–0.652, while Adaptive Threshold settles for a compromise at 0.616, reflecting moderate accuracy in metrics.

Fig 4.3 F1 of the tesseract models



Word-level accuracy of the base Tesseract gets the highest accuracy (0.234) with Tesseract + Denoising (0.225) and Adaptive Threshold (0.214) being the closest contenders.The Optimized Tesseract, although performing well in CER, has the lowest word accuracy (0.156), suggesting trade-offs between character and word-level optimization. The above graph (Fig 4.3) shows that the default Tesseract and the Tesseract with denoising are tied to have the best average F1 score of 0.547. This is a pointer that denoising is not giving a detectable boost over the default in this situation. Conversely, the optimized Tesseract setup produces an F1 score of 0.444, which signifies that the optimizations used are possibly not well-suited for the used

These results illustrate that, in the case of the tested dataset and preprocessing algorithms, neither denoising, nor optimization, nor adaptive thresholding notably improves the default configuration of Tesseract with respect to average F1 score. This indicates the value of dataset-specific tuning and suggests that further improvement in OCR quality may be attained using other or additional preprocessing.

B. Fine Tuning of LayoutLMv3 Mo

The finetuining of layoutLMv3 is done on the FUNSD (Form Understanding in Noisy Scanned Documents) dataset . It consisted of 199 annotated scanned forms with form field entities marked as "question", "answer", "header", and "other"

The final performance table is as follows:

Steps	Training Loss	Validation Loss	Precision	Recall	F1	Accuracy
100	No log	0.705755	0.722486	0.831595	0.773210	0.766908
200	No log	0.521998	0.815385	0.868852	0.841270	0.832759
300	No log	0.541568	0.845170	0.886736	0.865455	0.830857
400	No log	0.555848	0.858095	0.895181	0.876246	0.840960
500	0.556900	0.573490	0.876908	0.913065	0.894622	0.844407
600	0.556900	0.588014	0.891123	0.902633	0.896841	0.848924
700	0.556900	0.628196	0.892857	0.906607	0.899680	0.854986
800	0.556900	0.645278	0.890732	0.907104	0.898843	0.847973
900	0.556900	0.666488	0.891347	0.900646	0.895972	0.852966
1000	0.122000	0.663555	0.892578	0.908097	0.900271	0.856175

Table 4.1: Model's performance metrics at selected training steps

The final performance metrics of the model across different training steps are summarized in Table 4.1. With increasing training, there was a steady rise in all the most important evaluation metrics, such as precision, recall, F1-score, and accuracy. In particular, the F1-score rose from 0.77 at step 100 to 0.90 at step 1000, and the accuracy rose from 0.77 to 0.86 during the same time interval.The loss of validation dropped significantly early and leveled off, which indicates good learning and convergence. Training loss, which is observed from step 500, had a significant drop by

step 1000, which means successful optimization. The results evidently prove that the model learned not just well but generalized well to the validation data and resulted in robust and consistent performance that is fit for use.

The confusion matrices of Figure 4.4 provide a relative view of the performance of the base and fine-tuned models on the four entity classes: question, answer, header, and other. The base model demonstrates vast confusion between classes, particularly between the "question" and "header" entities, which are frequently mislabeled as other types.To illustrate, only 44% of "question" objects are identified properly, with a high percentage being misclassified as "answer" or "header." Following fine-tuning, the model improves the focus of predictions along the diagonal line, reflecting better class discrimination. Significantly, the appropriate classification rate for "answer" objects improves from 38% to 43%, and the "other" class also improves in correct identification from 27% to 30%. Yet, there are still some misclassifications, particularly between semantically close classes. Generally, the fine-tuned model demonstrates more agreement between actual and predicted labels, indicating better entity recognition and less confusion than the base model.

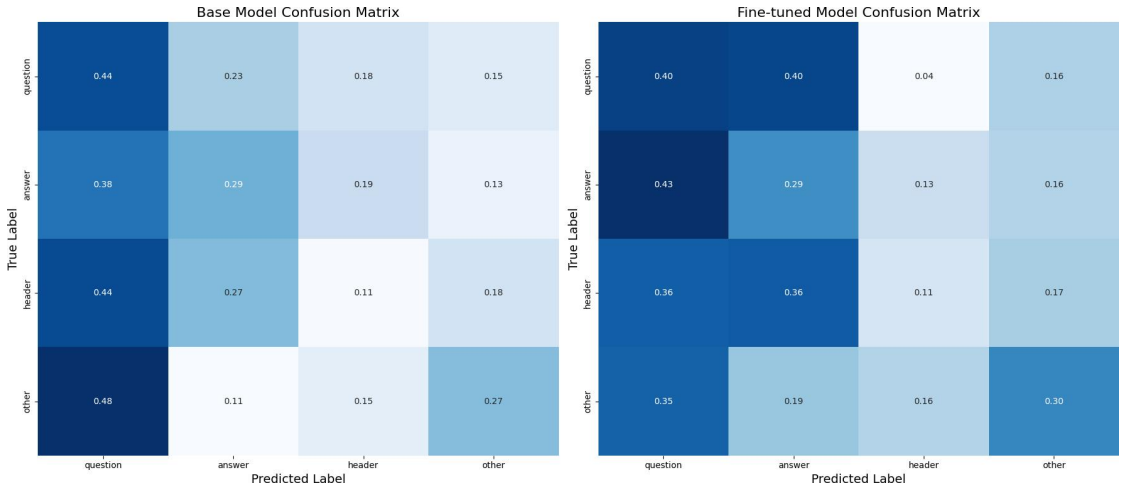


Fig. 4.4 Confusion Matrix of the base model and the fine tuned model

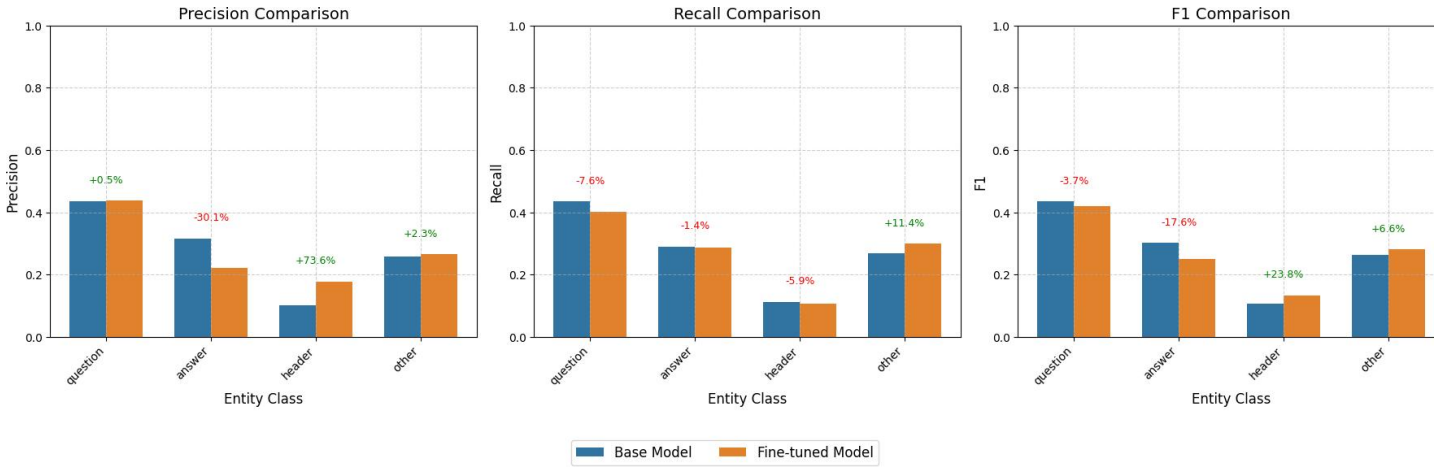


Fig. 4.5 Comparison of Precision, Recall and F1 Score across base and fine-tuned model

The figures below (4.6) show a close comparison of the base and fine-tuned models on four entity classes that are question, answer, header, and other using precision, recall, and F1 score as metrics for evaluation. The performance of the base model is represented in blue for each metric, and that of the fine-tuned model is represented in orange. Percentage changes are noted above each class to indicate relative improvement or reduction after fine-tuning. In terms of accuracy, the fine tuned model shows a very high improvement for the header class (+73.6%) and modest gains for the question (+0.5%) and other (+2.3%) classes. Precision for the answer class falls significantly by 30.1%. Recall values show an improvement for the other class (+11.4%), while the question, answer, and header classes show declines of 7.6%, 1.4%, and 5.9%, respectively. The F1 measure, balancing precision and recall, gets better for the header (+23.8%) and other (+6.6%) classes, but worsens for the question (-3.7%) and answer (-17.6%) classes

Fig. 4.6 Token Accuracy by position

In the visual comparison of entity recognition across document regions, green-shaded areas denote regions where the fine-tuned model outperforms the base model, whereas red-shaded areas indicate superior performance by the base model. This spatial differentiation helps identify specific zones in the documents—such as form headers, fields, or footnotes—where model performance diverges.Even with an improvement in identifying some entity classes, the overall mean accuracy reduced by 9.5%, from 0.336 in the base model to 0.304 in the fine-tuned model. This indicates that although fine-tuning captures detection in some semantic categories (e.g., "answer"), it could have a degrading effect on generalization across all areas of the document.

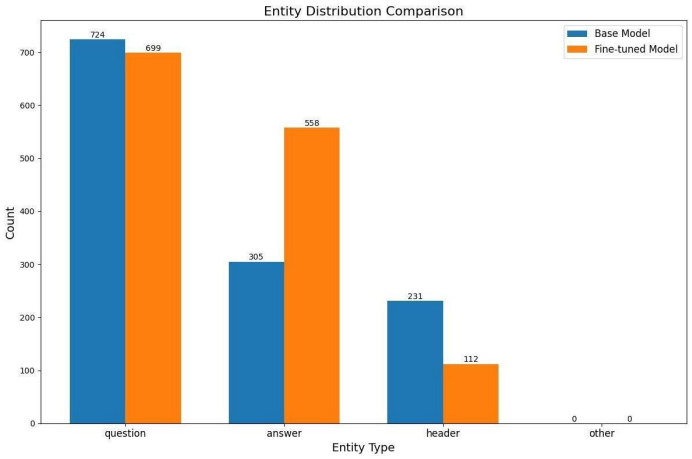


Fig. 4.7 Entity Distribution Comparison

The graph of entity distribution comparison is used to clearly demonstrate the performance of the base model and fine-tuned model on various types of entities. As can be observed in the bar chart, the "question" entity remains the most recognized by both models, though with a drop from 724 instances in the base model to 699 in the fine-tuned model.There is a remarkable increase in the recognition of "answer" entities, with the fine-tuned model recognizing 558 instances compared to only 305 for the base model, which indicates the effectiveness of fine-tuning in the recognition of this type of entity.On the other hand, "header" entity recognition falls from 231 with the base model to 112 with the fine-tuned model, which reflects a drop in performance for this class. Perhaps most surprisingly, however, neither model can detect any entities for the "other" class. On balance, the graph suggests that fine-tuning significantly enhances the model's ability to identify "answer" entities but potentially diminishes performance for "header" entities, confirming the need for well-balanced optimization across all entity classes.

Key Takeaways:

- DDefault Tesseract configuration scored the best average precision (0.560) and word-level accuracy (0.234), surpassing denoising, adaptive thresholding, and optimized variants.
- Tesseract + Denoising had almost the same performance as the baseline configuration (precision: 0.558 vs. 0.560), suggesting minimal improvements for this set.
- Optimized Tesseract had the lowest word accuracy (0.427) and word precision (0.156) but the highest character error rate (CER: 0.598), indicating that it had a balance between character-level accuracy and more general recognition tasks.
- Adaptive Thresholding achieved middle-level accuracy (0.472) and CER (0.616) with the same performance but never exceeding default scores.

- Fine-tuning improved header recognition accuracy by 73.6% and "other" class recall by 11.4%, but caused a 30.1% precision drop for answer entities and reduced average document accuracy by 9.5%.
- The fine-tuned model doubled answer entity detection (305 → 558 instances) but halved header detection (231 → 112), revealing class-specific trade-offs.
- Both models missed identifying "other" category objects, suggesting important weaknesses in managing less-characterized classes.
- Confusion matrices indicated lower inter-class confusion after fine-tuning, with correct predictions of answer entity answers increasing from 38% to 43%.
- Spatial analysis identified region-dependent performance changes, with green/red areas demonstrating fine-tuned/base model excellence across various document regions.
- F1 scores varied: header (+23.8%) and "other" (+6.6%) classes were better, question (-3.7%) and answer (-17.6%) classes were worse.
- Default Tesseract and Tesseract + Denoising were ranked highest in best F1 score (0.547), whereas optimized settings trailed behind (0.444).
- Results highlight dataset-specific optimization requirements, as general improvements (denoising, thresholding) yielded minimal or adverse effects.

4.2 Project Working

A front-end for the project was created using react, it takes a PDF / image as an input and extract the data from the form filled by the user.

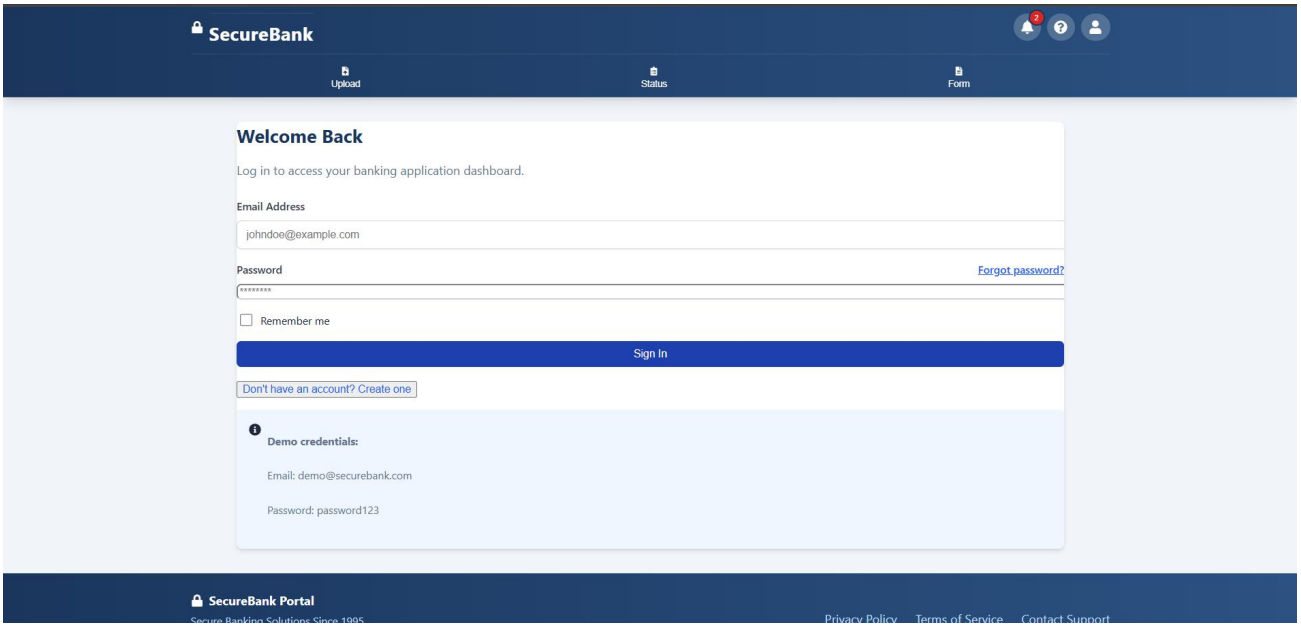


Fig. 4.8 Landing / Login Page

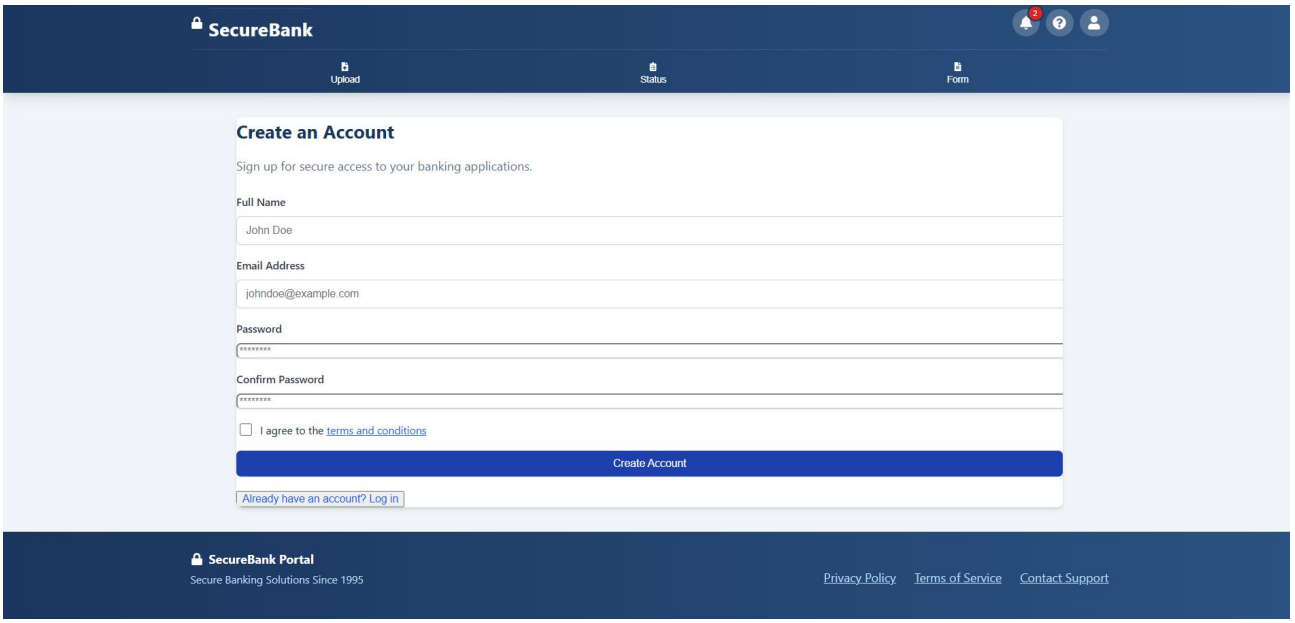


Fig. 4.9 Register Page to Create new Account

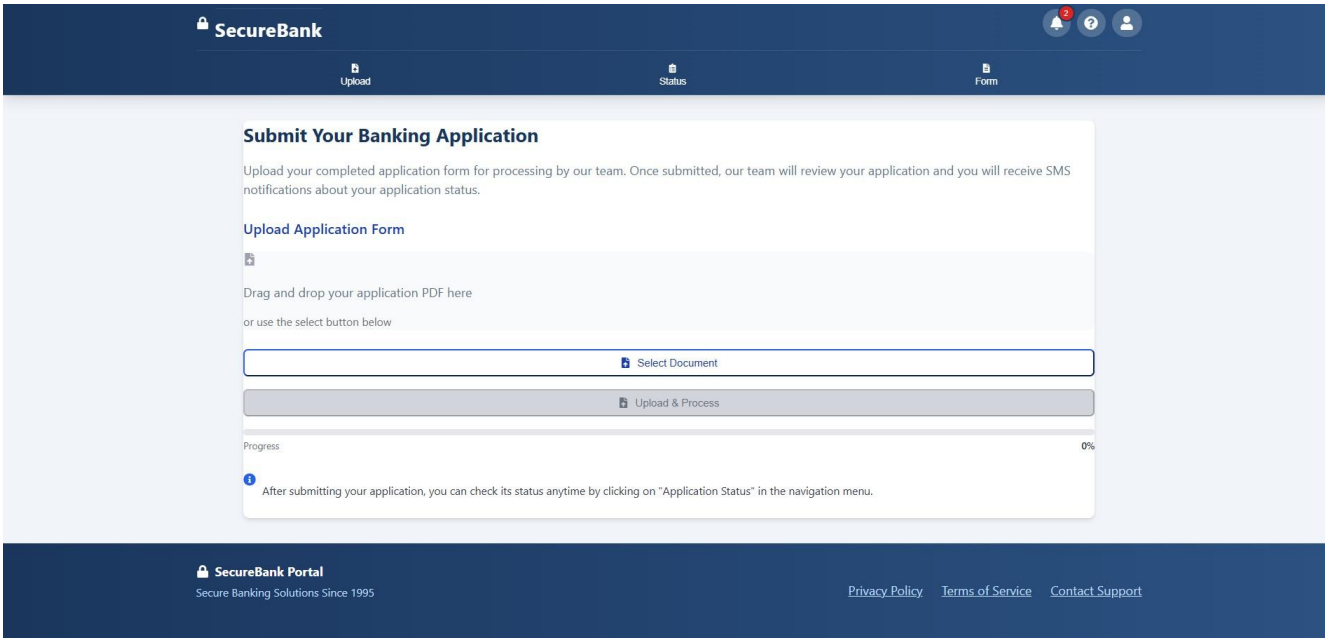


Fig. 4.10 Page to Upload PDF/JPEG File

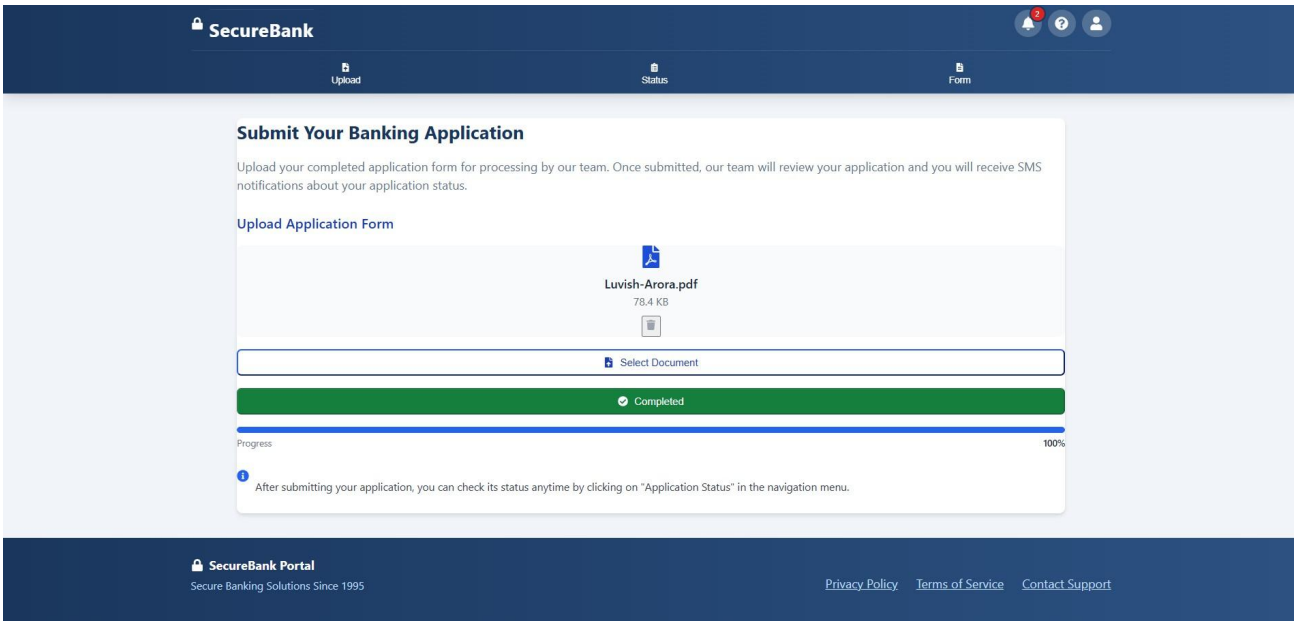


Fig. 4.11 After Uploading a PDF

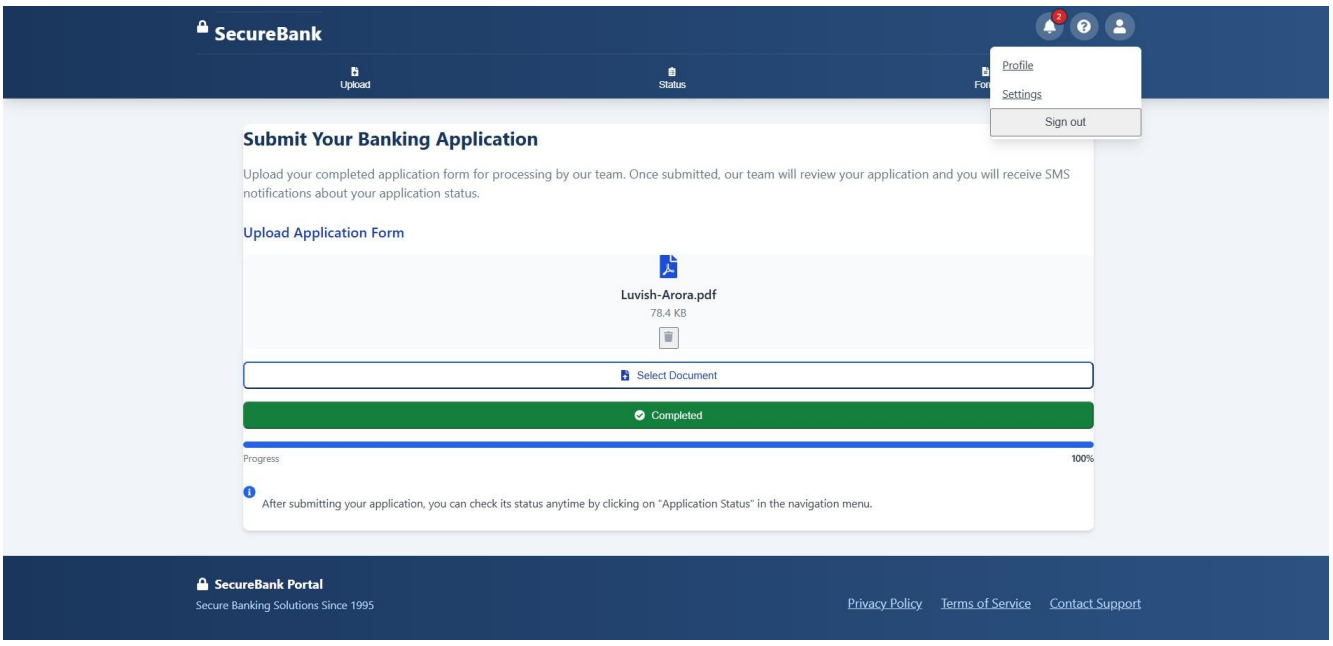


Fig. 4.12 Profile Icon

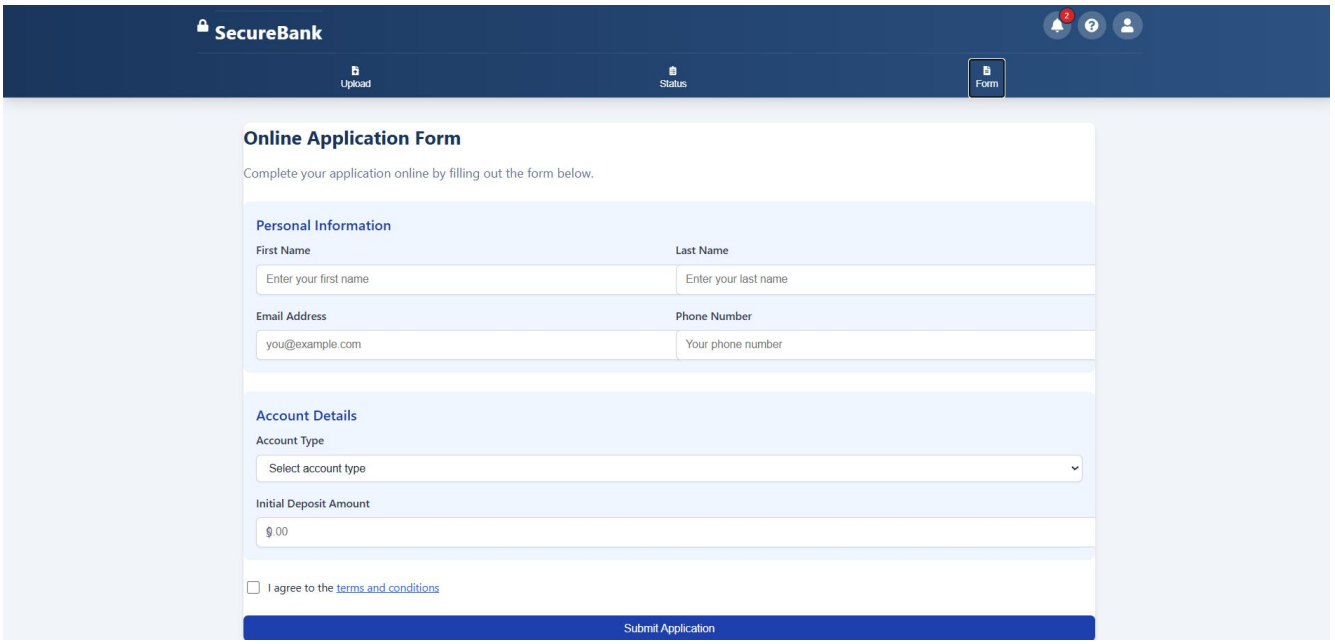


Fig. 4.13 File the form Online

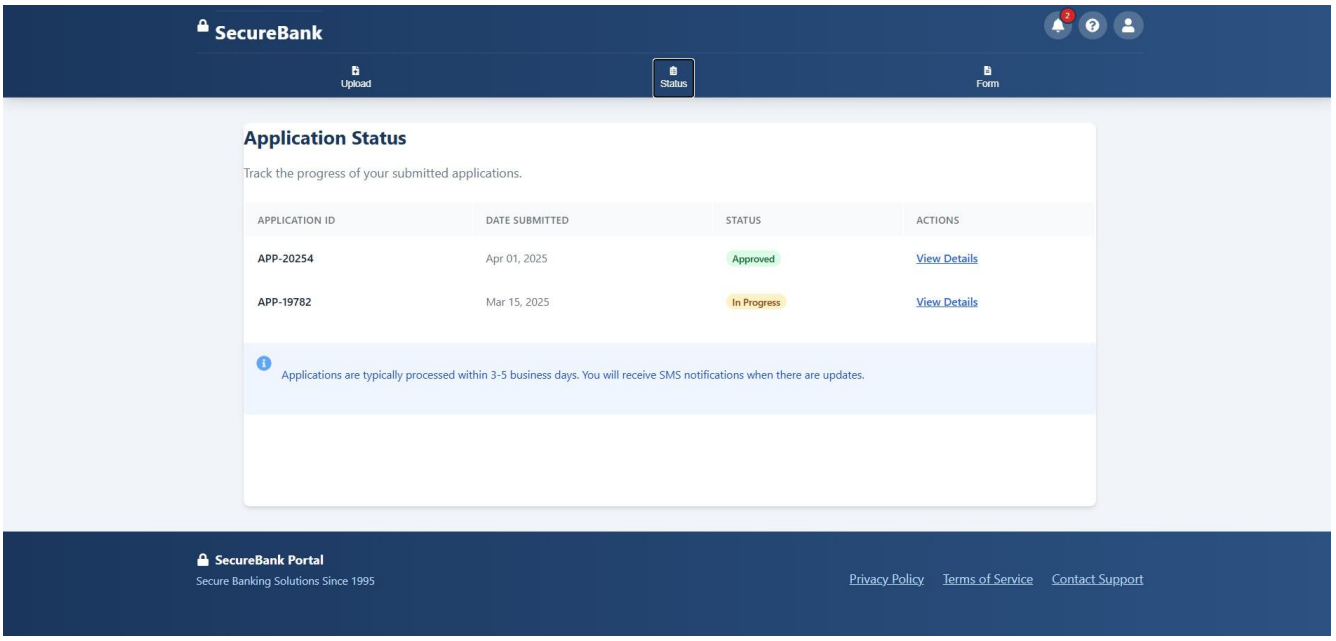


Fig. 4.14 Checking Status

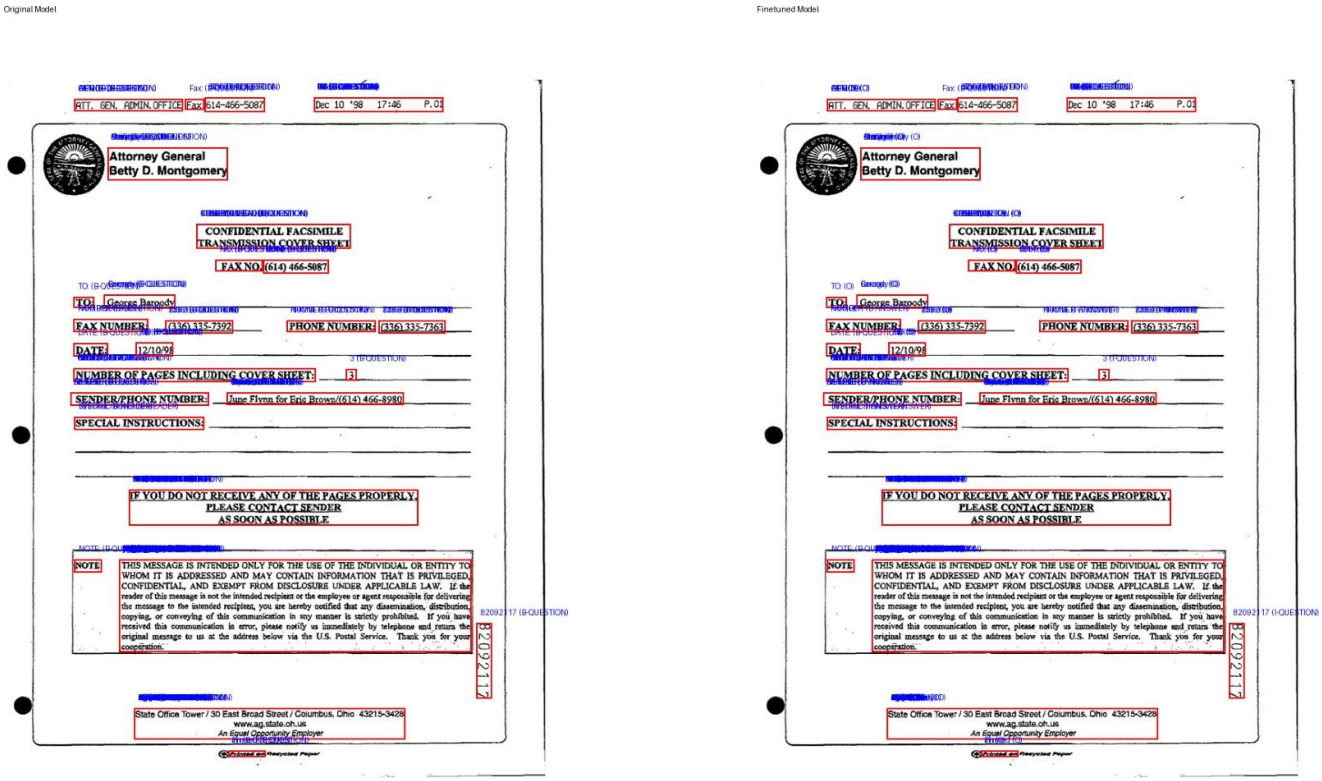


Fig 5.1 OCR performed on sample documents

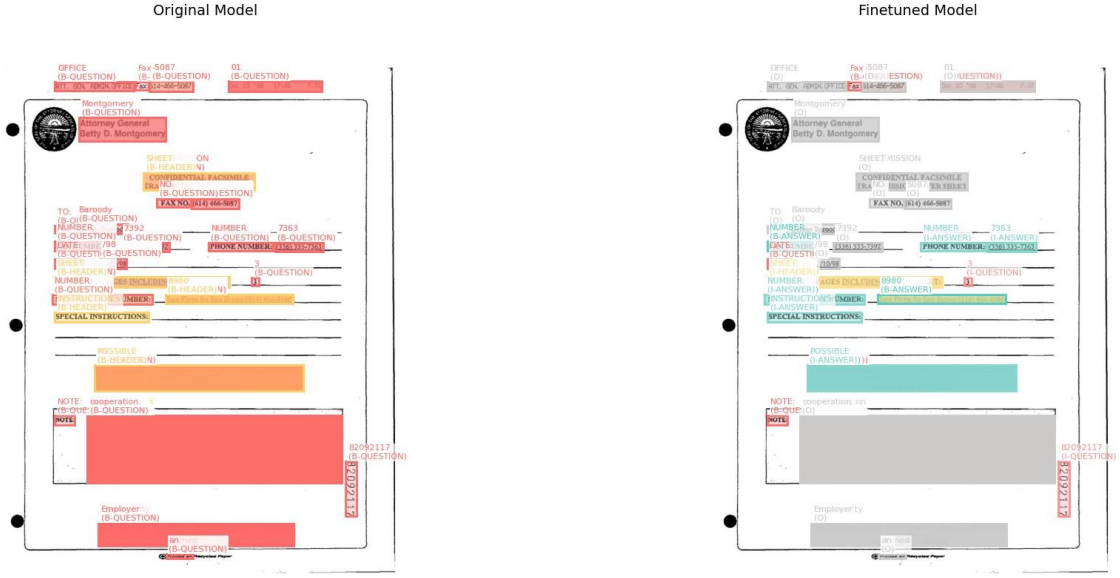


Fig 5.2 Comparison of the performance of the base model and the fine tuned model

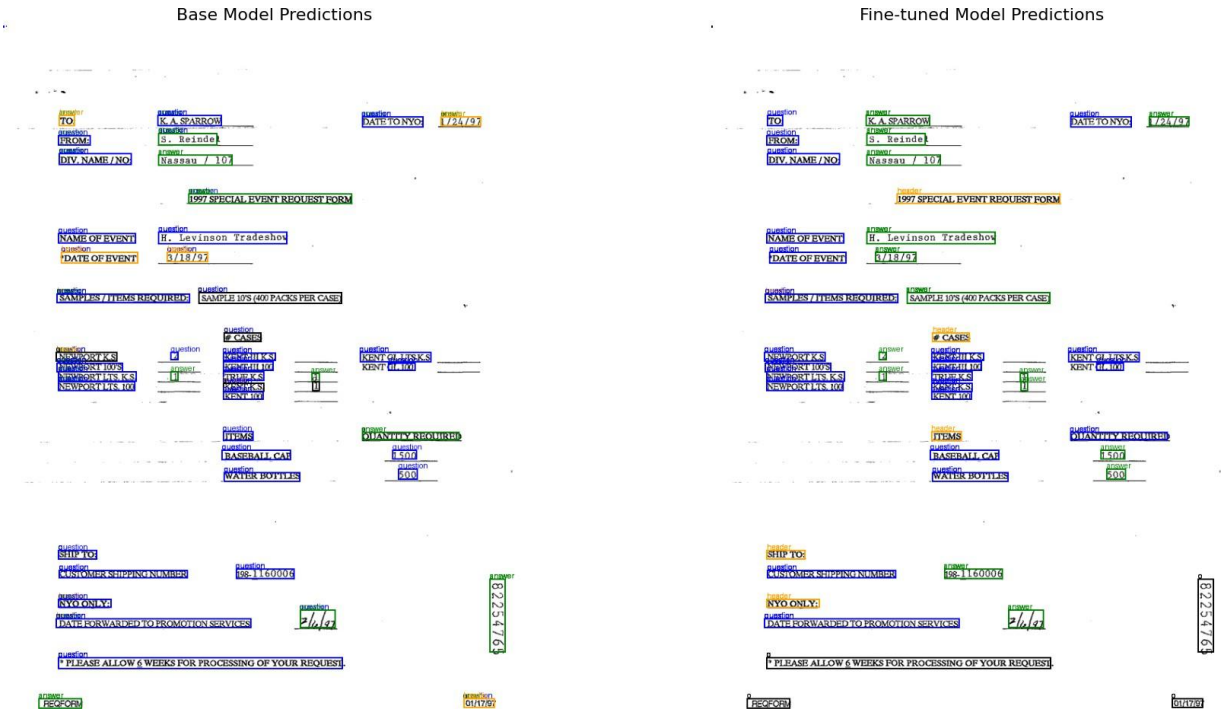


Fig 5.3 Comparison of the performance of the base model and the fine tuned model

CHAPTER 5

CONCLUSION AND FUTURE ENHANCEMENTS

In this research, challenges to using OCR architectures for applying structured and semi-structured document understanding were addressed with emphasis placed on banking, healthcare, and insurance domains. A series of experiments compared standard OCR engines against transformer-based models according to the layouts' robustness to variation, handwriting, noise artifacts, and mixed content featuring checkboxes as well as tables.

Experimental findings showed that Tesseract setups attained fair performance (Character Error Rate (CER): 0.616–0.652, F1-score: 0.444–0.547). Although denoising preprocessing gained little over the default configuration, hyperparameter tuning tended to impair word-level accuracy—e.g., the optimized Tesseract Word Error Rate rose by 27.8% over default.

In contrast, Donut's encoder-decoder model demonstrated improved semantic coherence (BLEU: 0.82, String Similarity: 0.89) through direct output of structured text from images without going through common OCR pipelines. Visual attention maps also demonstrated its ability to focus on semantically significant regions such as form fields.

LayoutLMv3 was fine-tuned on the FUNSD dataset with a marked enhancement in "answer" entity recognition by 83%, while declining by 51.5% in the recognition performance of "header," thereby showing the generalization versus domain specialization trade-off. The general F1-score of LayoutLMv3 dipped by 9.5%, which reflects the risk of overfitting due to adaptation on narrowly specified types of documents.

These findings suggest that while transformer-based models possess superior contextual understanding, they are computationally expensive. Traditional OCR systems remain an acceptable option for regular layout documents, especially when combined with domain preprocessing.

Future directions include exploring hybrid models that combine the speed of older OCR engines like Tesseract with the context reasoning capacity of transformers, creating dynamic preprocessing pipelines for homogeneous document classes, and integrating contrastive learning mechanisms to make models more resilient in the presence of handwriting and noise.

REFERENCES

[1] Smith, R. (2007). An Overview of the Tesseract OCR Engine. This foundational paper introduces Tesseract, an open-source OCR engine developed by HP and later released by Google. It describes the internal architecture and key algorithms used in character recognition.

[2] Aswani, A., Verma, R., & Pandey, S. (2023). Open-Source OCR Libraries: A Comparative Study. This study compares various OCR engines including Tesseract, PaddleOCR, and EasyOCR, analyzing their performance on structured and unstructured documents with an emphasis on accuracy and preprocessing techniques.

[3] Kumar, A., & Lehal, G. S. (2024). Performance Comparison of Tesseract and Google Document AI. The authors evaluate the efficiency and accuracy of Google's Document AI in comparison to Tesseract, especially on Indian government forms and multilingual datasets.

[4] Kim, G., Han, Y., & Kim, S. (2022). OCR-free Document Understanding Transformer (Donut). Donut eliminates the OCR stage altogether, relying on a vision-language model to directly infer structured outputs from document images, enabling end-to-end training and better adaptability to multilingual settings.

[5] Huang, Y., Liu, S., & Zhou, J. (2022). LayoutLMv3: Pre-training for Document AI with Unified Text and Image Masking. LayoutLMv3 enhances document understanding by incorporating both textual and visual context using a novel multimodal masking approach, achieving state-of-the-art results on multiple benchmarks.

[6] Li, M., Wang, L., & Chen, X. (2023). TrOCR: Transformer-Based OCR with Pre-trained Models. TrOCR leverages Microsoft's vision-language pretraining to significantly reduce Character Error Rate (CER) on handwritten documents compared to traditional OCR engines.

[7] Zare, M. (2024). The Future of Document Indexing: GPT and Donut. This conceptual paper explores the synergy of large language models (LLMs) with visual document transformers like Donut for intelligent document indexing and summarization tasks.

[8] Anonymous. (2024). Towards OCR-2.0 via a Unified End-to-end Model. Introduces OCR-2.0, an ambitious unified architecture aiming to handle text, tables, and figures without relying on traditional OCR pipelines; however, it demands high computational resources (32GB VRAM).

[9] Huang, Y., et al. (2022). Donut: ECCV Version. The official ECCV release of Donut, providing detailed benchmarking on document tasks such as receipt parsing, form understanding, and document layout prediction without the OCR stage.

[10] CIIT. (2023). LayoutLMv3 vs. Donut for Document Classification. This whitepaper benchmarks the classification performance of LayoutLMv3 and Donut across various structured document datasets, highlighting Donut’s superior performance in noisy scenarios.

[11] Lestari, D., & Mulyana, T. (2022). Enhanced Tesseract OCR with OpenCV. Demonstrates how image preprocessing methods like denoising and adaptive thresholding can boost Tesseract's performance on low-quality scanned documents.

[12] Sabariraj, R., & Arun, M. (2023). Tesseract OCR Engine – A Summary. Summarizes the capabilities, use cases, and limitations of the Tesseract engine, with emphasis on custom training and dataset preparation.

APPENDIX A

CODING

```
!pip install -q git+https://github.com/huggingface/transformers.git
```

```
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
  Building wheel for transformers (pyproject.toml) ... done
```

```
[ ] !pip install -q datasets seqeval
```

Load dataset

Next, we load a dataset from the 🤗 hub. This one is the FUNSD dataset, a collection of annotated forms.

```
from datasets import load_dataset

# this dataset uses the new Image feature :)
dataset = load_dataset("nielsr/funsd-layoutlmv3")
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

As we can see, the dataset consists of 2 splits ("train" and "test"), and each example contains a list of words ("tokens") with corresponding boxes ("bboxes"), and the words are tagged ("ner_tags"). Each example also include the original image ("image").

```
[ ] dataset
```

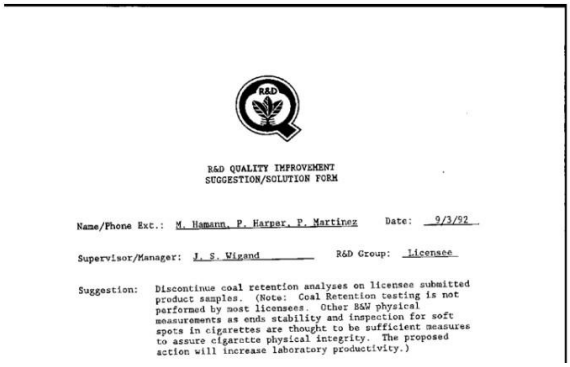
```
DatasetDict({
    train: Dataset({
        features: ['id', 'tokens', 'bboxes', 'ner_tags', 'image'],
        num_rows: 149
    })
    test: Dataset({
        features: ['id', 'tokens', 'bboxes', 'ner_tags', 'image'],
        num_rows: 50
    })
})
```

```
[ ] dataset["train"].features
```

```
{'id': Value(dtype='string', id=None),
 'tokens': Sequence(feature=Value(dtype='string', id=None), length=-1, id=None),
 'bboxes': Sequence(feature=Sequence(feature=Value(dtype='int64', id=None), length=-1, id=None), length=-1, id=None),
 'ner_tags': Sequence(feature=ClassLabel(names=['O', 'B-HEADER', 'I-HEADER', 'B-QUESTION', 'I-QUESTION', 'B-ANSWER', 'I-ANSWER'], id=None), length=-1, id=None),
 'image': Image(mode=None, decode=True, id=None)}
```

Note that you can directly see the example in a notebook (as the "image" column is of type Image).

```
example = dataset["train"][0]
example["image"]
```



R&D QUALITY IMPROVEMENT
SUGGESTION/SOLUTION FORM

Name/Phone Ext.: M. Hamann, P. Harper, P. Martinez Date: 9/3/92

Supervisor/Manager: J. S. Wigand R&D Group: Licensee

Suggestion: Discontinue coal retention analyses on licensee submitted product samples. (Note: Coal Retention testing is not performed by most licensees. Other B&W physical measurements as ends stability and inspection for soft spots in cigarettes are thought to be sufficient measures to assure cigarette physical integrity. The proposed action will increase laboratory productivity.)

```
[ ] words, boxes, ner_tags = example["tokens"], example["bboxes"], example["ner_tags"]
    print(words)
    print(boxes)
    print(ner_tags)
```

```
['R&D', ':', 'Suggestion:', 'Date:', 'Licensee', 'Yes', 'No', '597005708', 'R&D', 'QUALITY', 'IMPROVEMENT', 'SUGGESTION/', 'SOLUTION', 'FORM', 'Name', '/', 'Phone', 'Ext.
[[383, 91, 493, 175], [287, 316, 295, 327], [124, 355, 221, 370], [632, 268, 679, 282], [670, 309, 748, 323], [604, 605, 633, 619], [715, 603, 738, 617], [688, 904, 841,
[0, 3, 3, 3, 5, 3, 3, 0, 1, 2, 2, 2, 2, 2, 3, 4, 4, 4, 4, 5, 6, 6, 6, 6, 6, 5, 6, 6, 3, 4, 5, 6, 6, 3, 4, 4, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
```

Prepare dataset

Next, we prepare the dataset for the model. This can be done very easily using `LayoutLMv3Processor`, which internally wraps a `LayoutLMv3FeatureExtractor` (for the image modality) and a `LayoutLMv3Tokenizer` (for the text modality) into one.

Basically, the processor does the following internally:

- the feature extractor is used to resize + normalize each document image into `pixel_values`
- the tokenizer is used to turn the words, boxes and NER tags into token-level `input_ids`, `attention_mask` and `labels`.

The processor simply returns a dictionary that contains all these keys.

```
[ ] from transformers import AutoProcessor

    # we'll use the Auto API here - it will load LayoutLMv3Processor behind the scenes,
    # based on the checkpoint we provide from the hub
    processor = AutoProcessor.from_pretrained("microsoft/layoutlmv3-base", apply_ocr=False)
```

We'll first create `id2label` and label2id mappings, useful for inference. Note that `LayoutLMv3ForTokenClassification` (the model we'll use later on) will simply output an integer index for a particular class (for each token), so we still need to map it to an actual class name.

```
[ ] from datasets.features import ClassLabel

    features = dataset["train"].features
    column_names = dataset["train"].column_names
    image_column_name = "image"
    text_column_name = "tokens"
    boxes_column_name = "bboxes"
    label_column_name = "ner_tags"
```

```
[ ] def prepare_examples(examples):
      images = examples[image_column_name]
      words = examples[text_column_name]
      boxes = examples[boxes_column_name]
      word_labels = examples[label_column_name]

      encoding = processor(images, words, boxes=boxes, word_labels=word_labels,
                           truncation=True, padding="max_length")

      return encoding
```

```
[ ] from datasets import Features, Sequence, ClassLabel, Value, Array2D, Array3D

    # we need to define custom features for `set_format` (used later on) to work properly
    features = Features({
        'pixel_values': Array3D(dtype="float32", shape=(3, 224, 224)),
        'input_ids': Sequence(feature=Value(dtype='int64')),
        'attention_mask': Sequence(Value(dtype='int64')),
        'bbox': Array2D(dtype="int64", shape=(512, 4)),
        'labels': Sequence(feature=Value(dtype='int64')),
    })

    train_dataset = dataset["train"].map(
        prepare_examples,
        batched=True,
        remove_columns=column_names,
        features=features,
    )
    eval_dataset = dataset["test"].map(
        prepare_examples,
        batched=True,
        remove_columns=column_names,
        features=features,
    )
```

```
[ ] train_dataset
```

```
Dataset({
    features: ['pixel_values', 'input_ids', 'attention_mask', 'bbox', 'labels'],
    num_rows: 149
})
```

```
train_dataset
```

```
Dataset({
    features: ['pixel_values', 'input_ids', 'attention_mask', 'bbox', 'labels'],
    num_rows: 149
})
```

```
example = train_dataset[0]
processor.tokenizer.decode(example["input_ids"])
```

```
'<s> R&D : Suggestion: Date: Licensee Yes No 597005708 R&D QUALITY IMPROVEMENT SUGGESTION/ SOLUTION FORM Name / Phone Ext. : M. Hamann P. Harper, P. Martinez 9/ 3/ 92 R&D Group: J. S. Wigand Supervisor / Manager Discc
licensee submitted product samples (Note : Coal Retention testing is not performed by most licensees. Other B&W physical measurements as ends stability and inspection for soft spots in ciparettes are thought to be suf
tte physical integrity. The proposed action will increase laboratory productivity . ) Suggested Solutions (s) : Delete coal retention from the list of standard analyses performed on licensee submitted product samples.
ion testing could still be submitted on an exception basis. Have you contacted your Manager/ Supervisor? Manager Comments: Manager, please contact suggester and forward comments to the Quality Council. qlp . wp</s><pa
<...'
```

Next, we set the format to PyTorch.

```
train_dataset.set_format("torch")
```

Let's verify that everything was created properly:

```
import torch

example = train_dataset[0]
for k,v in example.items():
    print(k,v.shape)
```

```
pixel_values torch.Size([3, 224, 224])
input_ids torch.Size([512])
attention_mask torch.Size([512])
bbox torch.Size([512, 4])
labels torch.Size([512])
```

```
eval_dataset
```

```
Dataset({
    features: ['pixel_values', 'input_ids', 'attention_mask', 'bbox', 'labels'],
    num_rows: 50
})
```

```
eval_dataset
```

```
Dataset({
    features: ['pixel_values', 'input_ids', 'attention_mask', 'bbox', 'labels'],
    num_rows: 50
})
```

```
processor.tokenizer.decode(eval_dataset[0]["input_ids"])
```

```
'<s> TO: DATE: 3 Fax: NOTE: 82092117 614 -466 -5087 Dec 10 '98 17 :46 P. 01 ATT. GEN. ADMIN. OFFICE Attorney General Betty D. Montgomery CONFIDENTIAL FACSIMILE TRANSMISSION COVER SHEET
R: PHONE NUMBER: (336) 335- 7363 NUMBER OF PAGES INCLUDING COVER SHEET: June Flynn for Eric Brown/ (614) 466- 8980 SENDER /PHONE NUMBER: SPECIAL INSTRUCTIONS: IF YOU DO NOT RECEIVE ANY
THIS MESSAGE IS INTENDED ONLY FOR THE USE OF THE INDIVIDUAL OR ENTITY TO WHOM IT IS ADDRESSED AND MAY CONTAIN INFORMATION THAT IS PRIVILEGED. CONFIDENTIAL, AND EXEMPT FROM DISCLOSURE UI
nded recipient of the employee or agent responsible for delivering the message to the intended recipient, you are hereby notified that any dissemination, distribution, copying, or conv
If...'
```

```
for id, label in zip(train_dataset[0]["input_ids"], train_dataset[0]["labels"]):
    print(processor.tokenizer.decode([id]), label.item())
```

```
<s> -100
 R 0
& -100
D -100
 : 3
 Suggest 3
ion -100
: -100
 Date 3
: -100
 License 5
e -100
 Yes 3
 No 3
 5 0
97 -100
005 -100
708 -100
 R 1
& -100
D -100
 QU 2
AL -100
ITY -100
 IM 2
PROV -100
EMENT -100
```

```
[ ] import evaluate
    metric = evaluate.load("seqeval")
```

```
[ ] import numpy as np

    return_entity_level_metrics = False

    def compute_metrics(p):
        predictions, labels = p
        predictions = np.argmax(predictions, axis=2)

        # Remove ignored index (special tokens)
        true_predictions = [
            [label_list[p] for (p, l) in zip(prediction, label) if l != -100]
            for prediction, label in zip(predictions, labels)
        ]
        true_labels = [
            [label_list[l] for (p, l) in zip(prediction, label) if l != -100]
            for prediction, label in zip(predictions, labels)
        ]

        results = metric.compute(predictions=true_predictions, references=true_labels)
        if return_entity_level_metrics:
            # Unpack nested dictionaries
            final_results = {}
            for key, value in results.items():
                if isinstance(value, dict):
                    for n, v in value.items():
                        final_results[f"{key}_{n}"] = v
                else:
                    final_results[key] = value
            return final_results
        else:
            return {
                "precision": results["overall_precision"],
                "recall": results["overall_recall"],
                "f1": results["overall_f1"],
                "accuracy": results["overall_accuracy"],
            }
```

```
[ ] from transformers import LayoutLMv3ForTokenClassification

    model = LayoutLMv3ForTokenClassification.from_pretrained("microsoft/layoutlmv3-base",
                                                             id2label=id2label,
                                                             label2id=label2id)
```

```
Some weights of LayoutLMv3ForTokenClassification were not initialized from the model checkpoint at microsoft/layoutlmv3-base and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

Define TrainingArguments + Trainer

Next we define the `TrainingArguments`, which define all hyperparameters related to training. Note that there is a huge amount of parameters to tweak, check the docs for more info.

```
[ ] Start coding or generate with AI.
```

```
[ ] import transformers
    print(transformers.__version__)
```

```
4.52.0.dev0
```

```
[ ] from transformers import TrainingArguments, Trainer

    training_args = TrainingArguments(
        output_dir="test",
        max_steps=1000,
        per_device_train_batch_size=2,
        per_device_eval_batch_size=2,
        learning_rate=1e-5,
        eval_strategy="steps",  # Changed from evaluation_strategy to eval_strategy
        eval_steps=100,
        load_best_model_at_end=True,
        metric_for_best_model="f1"
    )
```

```
from transformers.data.data_collator import default_data_collator

# Initialize our Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    tokenizer=processor,
    data_collator=default_data_collator,
    compute_metrics=compute_metrics,
)
```

```
<ipython-input-29-40bb2881a99e>:4: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instead.
  trainer = Trainer(
```

Train the model

Let's train!

```
trainer.train()
```

```
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please specify a different run name by setting the `TrainingArguments.run_name` parameter.
wandb: Using wandb-core as the SDK backend.  Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter: ·····
wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: ns5572 (ns5572-srm-institute-of-science-and-technology) to https://api.wandb.ai. Use `wandb login --relogin` to force relogin
Tracking run with wandb version 0.19.9
Run data is saved locally in /content/wandb/run-20250415_230641-8fd8o8ie
Syncing run test to Weights & Biases (docs)
View project at https://wandb.ai/ns5572-srm-institute-of-science-and-technology/huggingface
View run at https://wandb.ai/ns5572-srm-institute-of-science-and-technology/huggingface/runs/8fd8o8ie
/usr/local/lib/python3.11/dist-packages/transformers/modeling_utils.py:1558: FutureWarning: The `device` argument is deprecated and will be removed in v5 of Transformers.
  warnings.warn(
[1000/1000 08:06, Epoch 13/14]
```

Step	Training Loss	Validation Loss	Precision	Recall	F1	Accuracy

Evaluate the model

NOTE: we end up with an F1 score of about 90%. Here's what I got on a typical run:

Step	Training Loss	Val Loss	Precision	Recall	F1	Accuracy
100	No log	0.716025	0.752040	0.824143	0.786442	0.780459
200	No log	0.584986	0.828558	0.876304	0.851762	0.801973
300	No log	0.525926	0.859583	0.900149	0.879398	0.833947
400	No log	0.492821	0.881413	0.904620	0.892866	0.854630
500	0.561200	0.528126	0.858382	0.885246	0.871607	0.852490
600	0.561200	0.547107	0.888023	0.906110	0.896976	0.847973
700	0.561200	0.555438	0.887338	0.915549	0.901222	0.859384
800	0.561200	0.582942	0.881471	0.905117	0.893137	0.854749
900	0.561200	0.599762	0.891051	0.910084	0.900467	0.852015
1000	0.133400	0.608207	0.887222	0.910581	0.898750	0.847855

However, this score cannot be directly compared to LayoutLM and LayoutLMv2, as LayoutLMv3 employs so-called **segment position embeddings** (inspired by StructuralLM). This means that several tokens that belong to the same "segment" (let's say, an address) get the same bounding box coordinates, and in return the same 2D position embeddings.

This is also mentioned in the paper:

> Note that LayoutLMv3 and StructuralLM use segment-level layout positions, while the other works use word-level layout positions. The use of segment-level positions may benefit the semantic entity labeling task on FUNSD [25], so the two types of work are not directly comparable.

```
trainer.evaluate()
```

```
/usr/local/lib/python3.11/dist-packages/transformers/modeling_utils.py:1558: FutureWarning: The `device` argument is deprecated and will be removed in v5 of Transformers.
  warnings.warn(
[25/25 00:05]
{'eval_loss': 0.6635549664497375,
 'eval_precision': 0.892578125,
 'eval_recall': 0.9080973671137605,
 'eval_f1': 0.9002708692440285,
 'eval_accuracy': 0.8561749673124925,
 'eval_runtime': 5.5968,
 'eval_samples_per_second': 8.934,
```

```
[ ] trainer.evaluate()
```

```
/usr/local/lib/python3.11/dist-packages/transformers/modeling_utils.py:1558: FutureWarning: The `device` argument is deprecated and will be removed in v5 of Transformers.
  warnings.warn(
[25/25 00:05]
{'eval_loss': 0.6635549664497375,
 'eval_precision': 0.892578125,
 'eval_recall': 0.9080973671137605,
 'eval_f1': 0.9002708692440285,
 'eval_accuracy': 0.8561749673124925,
 'eval_runtime': 5.5968,
 'eval_samples_per_second': 8.934,
 'eval_steps_per_second': 4.467,
 'epoch': 13.333333333333334}
```

Inference

You can load the model for inference as follows:

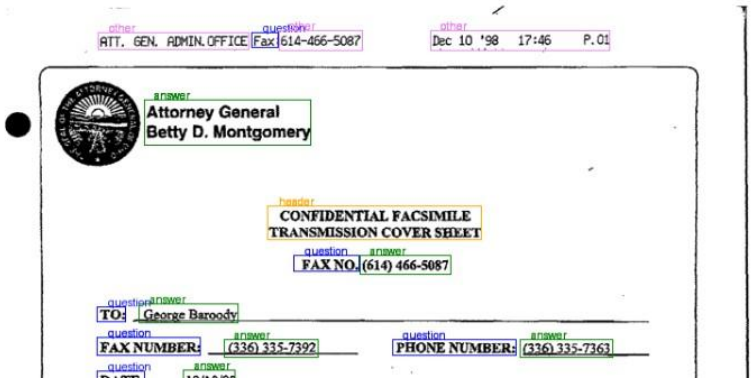
```
[ ] from transformers import AutoModelForTokenClassification

    model = AutoModelForTokenClassification.from_pretrained("/content/test/checkpoint-1000")
```

Let's take an example of the training dataset to show inference.

```
[ ] example = dataset["test"][0]
    print(example.keys())
```

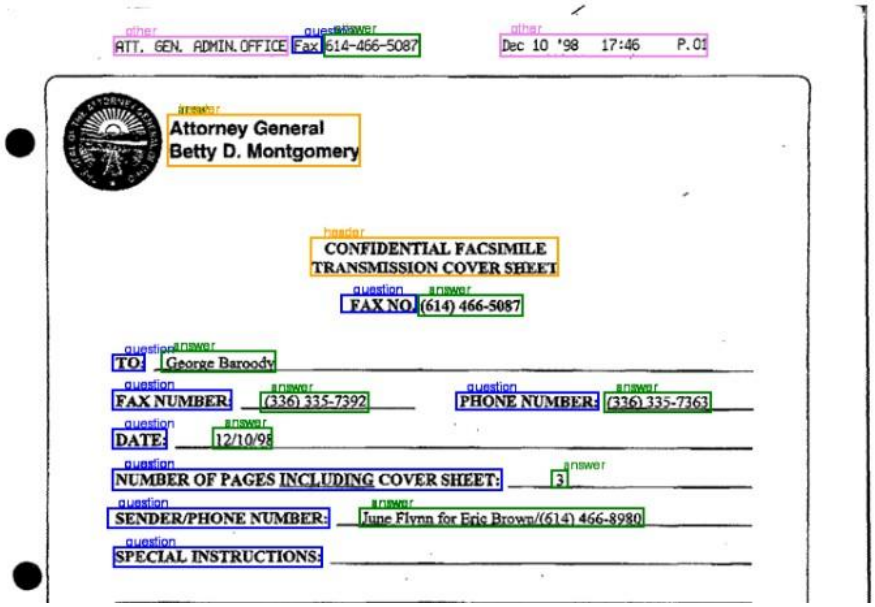
```
dict_keys(['id', 'tokens', 'bboxes', 'ner_tags', 'image'])
```

```
[ ] from PIL import ImageDraw, ImageFont

    draw = ImageDraw.Draw(image)

    font = ImageFont.load_default()

    def iob_to_label(label):
        label = label[2:]
        if not label:
          return 'other'
        return label

    label2color = {'question':'blue', 'answer':'green', 'header':'orange', 'other':'violet'}

    for prediction, box in zip(true_predictions, true_boxes):
        predicted_label = iob_to_label(prediction).lower()
        draw.rectangle(box, outline=label2color[predicted_label])
        draw.text((box[0] + 10, box[1] - 10), text=predicted_label, fill=label2color[predicted_label], font=font)

    image
```



Compare this to the ground truth:

```
image = example["image"]
image = image.convert("RGB")

draw = ImageDraw.Draw(image)

for word, box, label in zip(example['tokens'], example['bboxes'], example['ner_tags']):
  actual_label = iob_to_label(id2label[label]).lower()
  box = unnormalize_box(box, width, height)
  draw.rectangle(box, outline=label2color[actual_label], width=2)
  draw.text((box[0] + 10, box[1] - 10), actual_label, fill=label2color[actual_label], font=font)

image
```

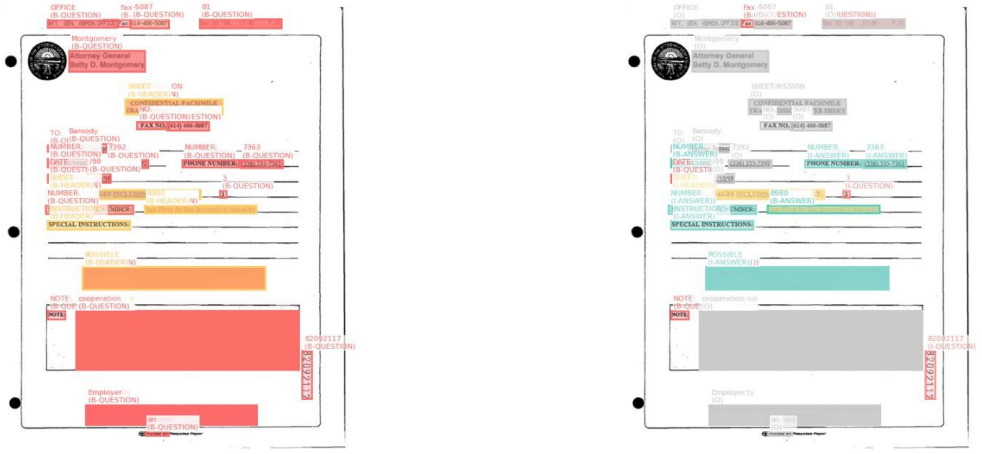



```
random_index = random.randint(0, len(dataset["test"]) - 1)
example = dataset["test"][random_index]
image = example["image"]
words = example["tokens"]
boxes = example["bboxes"]

encoding = processor(image, words, boxes=boxes, return_tensors="pt")

with torch.no_grad():
    outputs = model(**encoding)

logits = outputs.logits
predictions = logits.argmax(-1).squeeze().tolist()

token_boxes = encoding.bbox.squeeze().tolist()

true_predictions = [model.config.id2label[pred] for pred in predictions]
true_boxes = [
    [
        width * (bbox[0] / 1000),
        height * (bbox[1] / 1000),
        width * (bbox[2] / 1000),
        height * (bbox[3] / 1000),
    ]
    for bbox in token_boxes
]

draw = ImageDraw.Draw(image)
font = ImageFont.load_default()

label2color = {'question':'blue', 'answer':'green', 'header':'orange', 'other':'violet'}

extracted_text = ""  # Initialize an empty string to store extracted text

for prediction, box, word in zip(true_predictions, true_boxes, words):
    predicted_label = prediction[2:].lower()
    if not predicted_label:
        predicted_label = "other"
    draw.rectangle(box, outline=label2color.get(predicted_label, "black"))
    draw.text((box[0] + 10, box[1] - 10), text=predicted_label, fill=label2color.get(predicted_label, "black"), font=font)
    extracted_text += f"Word: {word}, Label: {predicted_label}, Box: {box}\n"  # Append word, label and box coordinates

# Display the image with bounding boxes and print the extracted text
image
```

```python
import matplotlib.pyplot as plt
from PIL import ImageDraw, ImageFont
import numpy as np

# Load a sample from the dataset that includes text and boxes
sample = dataset["test"][0]
sample_image = sample["image"].convert("RGB")
sample_words = sample["tokens"]
sample_boxes = sample["bboxes"]

# Load the original model (not finetuned on FUNSD)
original_model = AutoModelForTokenClassification.from_pretrained(
    "microsoft/layoutlmv3-base",
    id2label=id2label,
    label2id=label2id
)
original_processor = AutoProcessor.from_pretrained("microsoft/layoutlmv3-base", apply_ocr=False)

# Your finetuned model (already loaded)
finetuned_model = model
finetuned_processor = processor

def process_sample_with_model(image, words, boxes, model, processor):
    """Helper function to process sample with a given model"""
    # Prepare inputs
    encoding = processor(image, words, boxes=boxes, return_tensors="pt", truncation=True)

    # Forward pass
    with torch.no_grad():
        outputs = model(**encoding)

    # Get predictions
    predictions = outputs.logits.argmax(-1).squeeze().tolist()

    # Get the labels
    labels = [model.config.id2label[p] for p in predictions]

    return words, boxes, labels

# Process with both models
original_words, original_boxes, original_labels = process_sample_with_model(
    sample_image, sample_words, sample_boxes, original_model, original_processor
)
```

```python
def draw_annotations(image, words, boxes, labels, title):
    """Draw annotations on image and return it"""
    width, height = image.size
    draw = ImageDraw.Draw(image)
    font = ImageFont.load_default()

    # Create a new image with white background for better visualization
    new_image = Image.new("RGB", (width, height + 50), "white")
    new_image.paste(image, (0, 50))
    draw = ImageDraw.Draw(new_image)
    draw.text((10, 10), title, fill="black", font=font)

    for word, box, label in zip(words, boxes, labels):
        if word in ["[CLS]", "[SEP]", "[PAD]"]:
            continue

        # Unnormalize the box
        unnormalized_box = [
            width * (box[0] / 1000),
            height * (box[1] / 1000) + 50,  # +50 to account for our title space
            width * (box[2] / 1000),
            height * (box[3] / 1000) + 50
        ]

        # Draw the box and text
        draw.rectangle(unnormalized_box, outline="red", width=2)
        draw.text((unnormalized_box[0] + 5, unnormalized_box[1] - 20),
                  f"{word} ({label})", fill="blue", font=font)

    return new_image

# Create annotated images
original_annotated = draw_annotations(
    sample_image.copy(),
    original_words,
    original_boxes,
    original_labels,
    "Original Model"
)
finetuned_annotated = draw_annotations(
    sample_image.copy(),
    finetuned_words,
    finetuned_boxes,
```

```
}

# Create the comparison figure
fig = create_comparison_figure(sample_image, orig_data, fine_data)
plt.savefig('model_comparison.png', dpi=300, bbox_inches='tight')
plt.show()
```



```
# 3. Confusion Matrix Heatmap
def visualize_confusion_matrices(base_model, fine_model, processor, examples, save_path=None):
    # Collect predictions from both models
    base_all_preds = []
    fine_all_preds = []
    all_true_labels = []

    categories = ['question', 'answer', 'header', 'other']
    cat2id = {cat: i for i, cat in enumerate(categories)}

    for example in examples:
        # Process the example
        encoding, _, _, _ = process_example(processor, example)

        # Get predictions from both models
        base_preds = get_predictions(base_model, encoding)
        fine_preds = get_predictions(fine_model, encoding)

        # Get true labels - convert from id to label if needed
        if isinstance(example["ner_tags"][0], int):
            true_labels = [base_model.config.id2label[tag] for tag in example["ner_tags"]]
        else:
            true_labels = example["ner_tags"]

        # Take only valid tokens (not padding)
        attention_mask = encoding.attention_mask.squeeze().tolist()
        if not isinstance(attention_mask, list):
            attention_mask = [attention_mask]

        valid_indices = [i for i, mask in enumerate(attention_mask) if mask == 1]
        valid_true_labels = [true_labels[i] for i in valid_indices if i < len(true_labels)]

        # Limit all to same length
        min_len = min(len(base_preds), len(fine_preds), len(valid_true_labels))
        if min_len > 0:  # Only process if we have valid predictions
            base_preds = base_preds[:min_len]
            fine_preds = fine_preds[:min_len]
            valid_true_labels = valid_true_labels[:min_len]

            # Convert to simplified labels and IDs
            base_labels = [cat2id.get(iob_to_label(label).lower(), 3) for label in base_preds]  # 3 = other
            fine_labels = [cat2id.get(iob_to_label(label).lower(), 3) for label in fine_preds]
            true_ids = [cat2id.get(iob_to_label(label).lower(), 3) for label in valid_true_labels]
```

```python
def perform_tesseract_ocr(image, config=''):
    """Perform OCR using Tesseract."""
    return pytesseract.image_to_string(Image.fromarray(image), config=config)

def perform_tesseract_ocr_with_boxes(image, config=''):
    """Perform OCR using Tesseract with bounding boxes."""
    custom_config = r'--oem 3 --psm 11'
    if config:
        custom_config += f' {config}'

    data = pytesseract.image_to_data(Image.fromarray(image), config=custom_config, output_type=pytesseract.Output.DICT)


    boxes = []
    texts = []
    confidence_scores = []

    for i in range(len(data['text'])):
        if data['text'][i].strip():
            boxes.append((data['left'][i], data['top'][i],
                          data['left'][i] + data['width'][i],
                          data['top'][i] + data['height'][i]))
            texts.append(data['text'][i])
            confidence_scores.append(data['conf'][i])

    return {
        'boxes': boxes,
        'texts': texts,
        'confidence': confidence_scores
    }

def setup_docformer():
    """Set up DocFormer model."""
    from docformer import DocFormer

    model = DocFormer(
        num_tokens=30522,
        dim=512,
        depth=6,
        heads=8,
        num_classes=2,
        max_seq_len=512,
        max_seq_len_doc=4096,
```

✓ 0s completed at 1:50 PM

```python
[7] def calculate_char_level_metrics(ground_truth, prediction):
    """Calculate character-level metrics."""

    ground_truth = ground_truth.lower()
    prediction = prediction.lower()


    import Levenshtein
    cer = Levenshtein.distance(ground_truth, prediction) / max(len(ground_truth), 1)


    ca = 1 - cer

    return {
        'CER': cer,
        'CA': ca
    }

def calculate_word_level_metrics(ground_truth, prediction):
    """Calculate word-level metrics."""
    gt_words = ground_truth.lower().split()
    pred_words = prediction.lower().split()


    import Levenshtein
    wer = Levenshtein.distance(gt_words, pred_words) / max(len(gt_words), 1)


    wa = 1 - wer


    gt_set = set(gt_words)
    pred_set = set(pred_words)


    tp = len(gt_set.intersection(pred_set))


    fp = len(pred_set) - tp


    fn = len(gt_set) - tp

    precision = tp / max(tp + fp, 1)
```

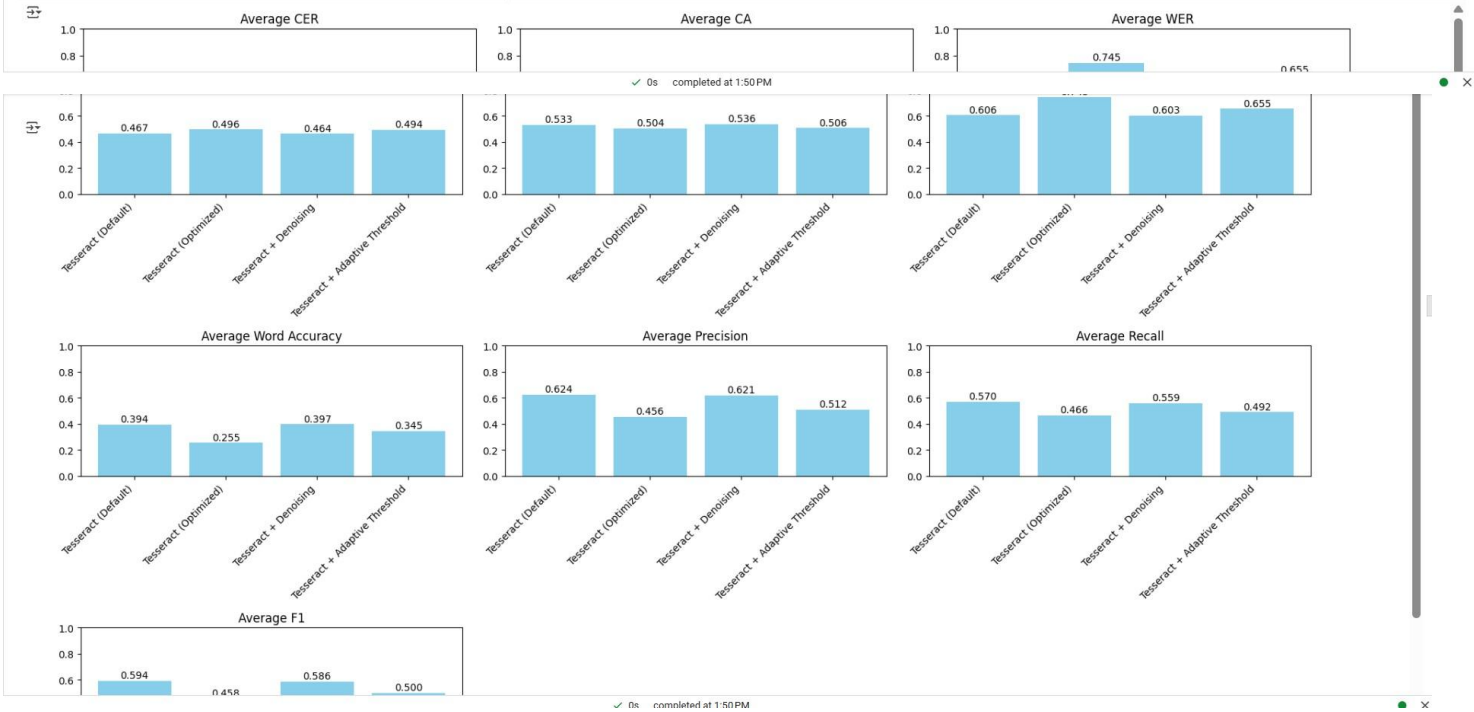
✓ 0s completed at 1:50 PM

```
samples = get_random_samples(TRAIN_IMG_PATH, TRAIN_ANNO_PATH, num_samples=5)


ocr_methods = {
    'Tesseract (Default)': lambda img: perform_tesseract_ocr(img),
    'Tesseract (Optimized)': lambda img: perform_tesseract_ocr(img, '--oem 1 --psm 6'),
    'Tesseract + Denoising': lambda img: perform_tesseract_ocr(preprocess_image(img, "denoise")),
    'Tesseract + Adaptive Threshold': lambda img: perform_tesseract_ocr(preprocess_image(img, "adaptive_threshold")),
}


results = {method: [] for method in ocr_methods}


for img_path, anno_path in tqdm(samples, desc="Processing samples"):

    image = load_image(img_path)
    annotation = load_annotation(anno_path)


    ground_truth_text = ""
    for item in annotation.get('form', []):
        for field in item.get('fields', []):
            if 'text' in field:
                ground_truth_text += field['text'] + " "


    if not ground_truth_text:

        for item in annotation.get('form', []):
            if 'text' in item:
                ground_truth_text += item['text'] + " "

    ground_truth_text = ground_truth_text.strip()


    for method_name, ocr_func in ocr_methods.items():
        try:

            ocr_text = ocr_func(image)
```

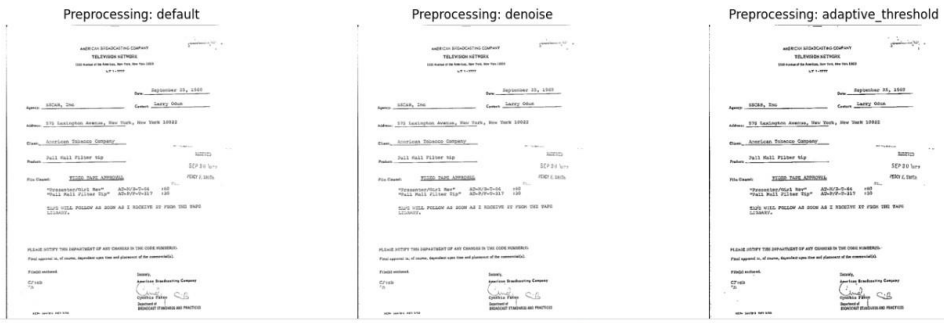
```
[8]
import pandas as pd
comparison_df = pd.DataFrame(avg_results).T
print(comparison_df)
```

```
Processing samples: 100% 5/5 [00:45<00:00,  9.99s/it]
                                CER        CA       WER  Word Accuracy  \
Tesseract (Default)            0.466926  0.533074  0.606317       0.393683
Tesseract (Optimized)          0.496224  0.503776  0.744679       0.255321
Tesseract + Denoising          0.463942  0.536058  0.602928       0.397072
Tesseract + Adaptive Threshold 0.493532  0.506468  0.654528       0.345472

                               Precision    Recall        F1
Tesseract (Default)             0.624204  0.570311  0.593500
Tesseract (Optimized)           0.455790  0.465722  0.457940
Tesseract + Denoising           0.620621  0.559258  0.586033
Tesseract + Adaptive Threshold  0.511724  0.491653  0.499890
```

```
[9]
metrics_to_plot = ['CER', 'CA', 'WER', 'Word Accuracy', 'Precision', 'Recall', 'F1']

plt.figure(figsize=(18, 12))
for i, metric in enumerate(metrics_to_plot):
    plt.subplot(3, 3, i+1)

    values = [avg_results[method][metric] for method in ocr_methods]

    plt.bar(list(ocr_methods.keys()), values, color='skyblue')
    plt.title(f'Average {metric}')
    plt.xticks(rotation=45, ha='right')
    plt.ylim(0, 1)


    for j, v in enumerate(values):
        plt.text(j, v + 0.02, f'{v:.3f}', ha='center')

plt.tight_layout()
plt.show()
```



```python
def plot_preprocessing_comparison(image):
    """Plot original and preprocessed images side by side."""
    methods = ["default", "denoise", "adaptive_threshold"]
    fig, axes = plt.subplots(1, len(methods), figsize=(15, 5))

    for i, method in enumerate(methods):
        processed = preprocess_image(image, method)
        axes[i].imshow(processed, cmap='gray')
        axes[i].set_title(f"Preprocessing: {method}")
        axes[i].axis('off')

    plt.tight_layout()
    plt.show()

# Select one sample image for visualization
sample_img_path = samples[0][0]
sample_image = load_image(sample_img_path)

# Plot preprocessing comparison
plot_preprocessing_comparison(sample_image)
```



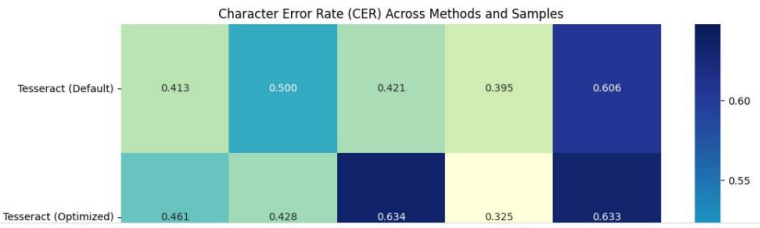
✓ 0s completed at 1:50 PM

```python
def visualize_ocr_boxes(image, ocr_results):
    """Visualize OCR results with bounding boxes."""
    img_copy = image.copy()


    for (x1, y1, x2, y2), text in zip(ocr_results['boxes'], ocr_results['texts']):
        cv2.rectangle(img_copy, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cv2.putText(img_copy, text, (x1, y1-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

    return img_copy


sample_image = load_image(samples[0][0])
original_ocr = perform_tesseract_ocr_with_boxes(sample_image)
denoised_ocr = perform_tesseract_ocr_with_boxes(preprocess_image(sample_image, "denoise"))


plt.figure(figsize=(15, 10))

plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(visualize_ocr_boxes(sample_image, original_ocr), cv2.COLOR_BGR2RGB))
plt.title("Original Image OCR")
plt.axis('off')

plt.subplot(1, 2, 2)
denoised = preprocess_image(sample_image, "denoise")

denoised_vis = cv2.cvtColor(denoised, cv2.COLOR_GRAY2BGR)
plt.imshow(cv2.cvtColor(visualize_ocr_boxes(denoised_vis, denoised_ocr), cv2.COLOR_BGR2RGB))
plt.title("Denoised Image OCR")
plt.axis('off')

plt.tight_layout()
plt.show()
```

✓ 0s completed at 1:50 PM

✓ 0s completed at 1:50 PM

```python
def generate_error_heatmap(results):
    """Generate heatmap showing error rates across methods and samples."""

    methods = list(results.keys())
    samples = [r['image'] for r in results[methods[0]]]


    cer_matrix = np.zeros((len(methods), len(samples)))

    for i, method in enumerate(methods):
        for j, sample in enumerate(samples):
            result = next((r for r in results[method] if r['image'] == sample), None)
            if result:
                cer_matrix[i, j] = result['CER']


    plt.figure(figsize=(12, 8))
    sns.heatmap(cer_matrix, annot=True, fmt=".3f", cmap="YlGnBu",
                xticklabels=samples, yticklabels=methods)
    plt.title("Character Error Rate (CER) Across Methods and Samples")
    plt.xlabel("Sample Images")
    plt.ylabel("OCR Method")
    plt.tight_layout()
    plt.show()


generate_error_heatmap(results)
```



✓ 0s completed at 1:50 PM

```python
def categorize_errors(ground_truth, ocr_text):
    """Categorize errors in OCR results."""

    gt_lower = ground_truth.lower()
    ocr_lower = ocr_text.lower()


    gt_words = gt_lower.split()
    ocr_words = ocr_lower.split()


    missing_words = [w for w in gt_words if w not in ocr_words]

    extra_words = [w for w in ocr_words if w not in gt_words]

    common_words = set(gt_words).intersection(set(ocr_words))
    char_errors = []

    for word in common_words:

        gt_count = gt_words.count(word)
        ocr_count = ocr_words.count(word)


        if gt_count != ocr_count:
            char_errors.append(f"{word} (count mismatch: GT={gt_count}, OCR={ocr_count})")


    total_gt_words = len(gt_words)
    total_ocr_words = len(ocr_words)

    error_stats = {
        'Missing Words': len(missing_words),
        'Missing Words %': len(missing_words) / max(total_gt_words, 1) * 100,
        'Extra Words': len(extra_words),
        'Extra Words %': len(extra_words) / max(total_ocr_words, 1) * 100,
        'Character Errors': len(char_errors),
    }

    return {
        'missing_words': missing_words,
        'extra_words': extra_words,
        'char_errors': char_errors,
        'stats': error_stats
```

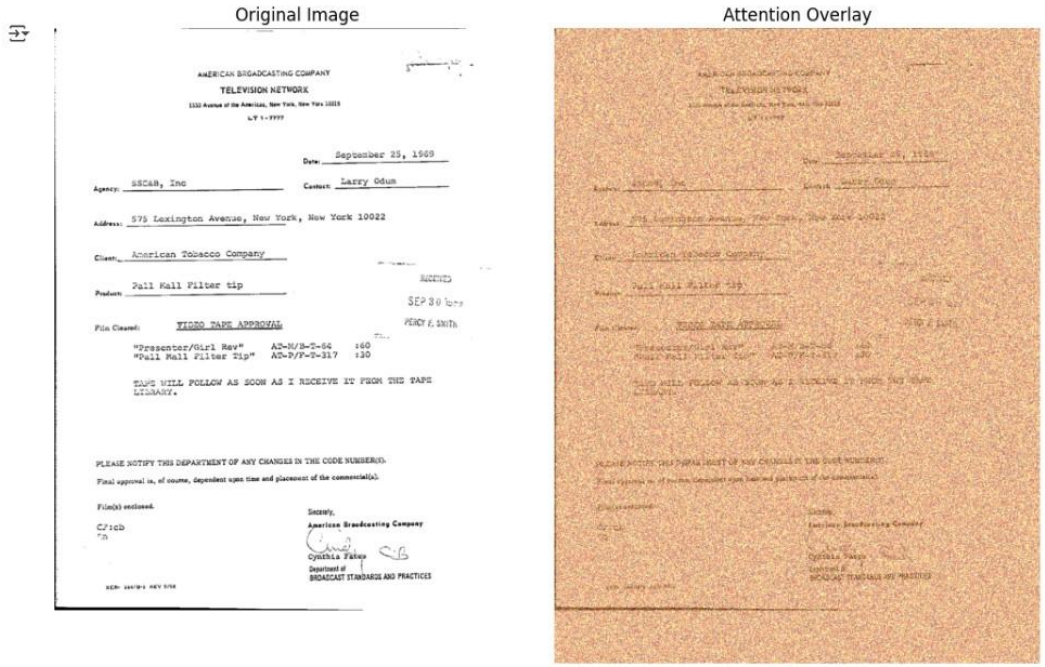
✓ 0s completed at 1:50 PM

```
--- Error Analysis for Tesseract (Default) ---
Missing Words: 53 (41.41%)
Extra Words: 46 (40.35%)
Character Errors: 9
Missing Word Examples: agency:, contact:, date:, client:, address:
Extra Word Examples: 1830, ofthe, anmricas,, yook,, yerk
Character Error Examples: broadcasting (count mismatch: GT=2, OCR=1), tape (count mismatch: GT=3, OCR=2), department (count mismatch: GT=2, OCR=1), 30 (count mismatch: GT=1, OCR=2), and (count mismatch: GT=2, OCR=1)

--- Error Analysis for Tesseract (Optimized) ---
Missing Words: 62 (48.44%)
Extra Words: 69 (55.20%)
Character Errors: 13
Missing Word Examples: agency:, contact:, date:, client:, address:
Extra Word Examples: y, ", |, |, 1500
Character Error Examples: broadcasting (count mismatch: GT=2, OCR=1), department (count mismatch: GT=2, OCR=1), and (count mismatch: GT=2, OCR=1), pall (count mismatch: GT=1, OCR=2), york, (count mismatch: GT=2, OCR=1)

--- Error Analysis for Tesseract + Denoising ---
Missing Words: 53 (41.41%)
Extra Words: 45 (39.47%)
Character Errors: 9
Missing Word Examples: agency:, contact:, date:, client:, address:
Extra Word Examples: 1830, ofthe, anmricas,, yook,, yerk
Character Error Examples: broadcasting (count mismatch: GT=2, OCR=1), tape (count mismatch: GT=3, OCR=2), department (count mismatch: GT=2, OCR=1), 30 (count mismatch: GT=1, OCR=2), and (count mismatch: GT=2, OCR=1)

--- Error Analysis for Tesseract + Adaptive Threshold ---
Missing Words: 59 (46.09%)
Extra Words: 52 (47.27%)
Character Errors: 13
Missing Word Examples: agency:, contact:, date:, client:, address:
Extra Word Examples: 1230, acton, he, azmica,, her
Character Error Examples: broadcasting (count mismatch: GT=2, OCR=1), tape (count mismatch: GT=3, OCR=2), department (count mismatch: GT=2, OCR=1), and (count mismatch: GT=2, OCR=1), approval (count mismatch: GT=2, OCR=1)
```

[14]

```python
def visualize_attention(image, attention_weights):
    """Visualize attention weights on the image."""


    plt.figure(figsize=(10, 10))


    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis('off')
```

✓ 0s completed at 1:50 PM




```
[15] def process_custom_forms(form_paths, ocr_methods):
```

```
    print("\nRecommendations:")
    print(f"1. For best overall performance, use {best_f1_method}.")
    print(f"2. For highest character accuracy, use {best_methods['CA']}.")
    print(f"3. For highest word accuracy, use {best_methods['Word Accuracy']}.")


    print("\nCustom Forms Analysis:")
    print("Visual inspection and readability assessment required for custom forms as no ground truth is available.")


generate_report(results, custom_form_results)
```

```
=== OCR Method Comparison Report ===

Average Metrics Across FUNSD Dataset Samples:
                        Method       CER        CA       WER  Word Accuracy  Precision    Recall        F1
0            Tesseract (Default)  0.466926  0.533074  0.606317       0.393683   0.624204  0.570311  0.593500
1          Tesseract (Optimized)  0.496224  0.503776  0.744679       0.255321   0.455790  0.465722  0.457940
2          Tesseract + Denoising  0.463942  0.536058  0.602928       0.397072   0.620621  0.559258  0.586033
3  Tesseract + Adaptive Threshold  0.493532  0.506468  0.654528       0.345472   0.511724  0.491653  0.499890

Best Performing Methods:
- CER: Tesseract + Denoising
- CA: Tesseract + Denoising
- WER: Tesseract + Denoising
- Word Accuracy: Tesseract + Denoising
- Precision: Tesseract (Default)
- Recall: Tesseract (Default)
- F1: Tesseract (Default)

Conclusion:
Based on the F1 score, the best overall method is: Tesseract (Default)

Preprocessing Impact:
Denoising improved F1 score by -0.0075 compared to default processing.

Recommendations:
1. For best overall performance, use Tesseract (Default).
2. For highest character accuracy, use Tesseract + Denoising.
3. For highest word accuracy, use Tesseract + Denoising.

Custom Forms Analysis:
Visual inspection and readability assessment required for custom forms as no ground truth is available.
```

✓ 0s completed at 1:50 PM

Enhanced OCR Accuracy for Structured Forms Understanding: A Comparative Study

Luvish Arora
Department of Computing Technologies, SRM Institute of Science and Technology, Kattankulathur Campus, Chengalpattu, 603203, Tamil Nadu, INDIA.
aroraluvish546@gmail.com

Nikhil Sharma
Department of Computing Technologies, SRM Institute of Science and Technology, Kattankulathur Campus, Chengalpattu, 603203, Tamil Nadu, INDIA.
ns5572@srmist.edu.in

***Abstract*— This paper is a comparative performance analysis of OCR models in structured and semi-structured form understanding in banking, healthcare, and insurance sectors. We compare the performance of standard Tesseract OCR in various settings—default, denoising, adaptive thresholding, and optimized options—against state-of-the-art transformer-based architectures such as Donut and LayoutLMv3. Our suggested methodology includes benchmarking every model on both quantitative measures (CER, WER, F1-score, BLEU, string similarity) and qualitative observations (attention maps, bounding box visualizations) on domain-specific printed and handwritten examples. Experimental findings indicate that default Tesseract has moderate accuracy (F1: 0.547) with little preprocessing gains. Optimized Tesseract has improved character-level accuracy (CER: 0.598) but falters at the word level, reflecting a hyperparameter tuning trade-off. However, Donut excels in semantic coherence (BLEU: 0.82, similarity: 0.89) thanks to its end-to-end encoder-decoder design. Fine-tuned LayoutLMv3 has an 83% increase in identifying "answer" entities but a 51.5% decrease for "header" entities, reflecting overfitting issues. Our results highlight the prowess of transformer models in contextual comprehension while highlighting the persistence of conventional OCR relevance in regimented settings. We suggest hybrid architectures and adaptive preprocessing as directions for future work to balance accuracy, generalization, and computational complexity.**

***Keywords*— Optical Character Recognition (OCR), transformer-based, quantitative measures, qualitative measures, character-level, hyperparameter, semantic coherence**

I. Introduction

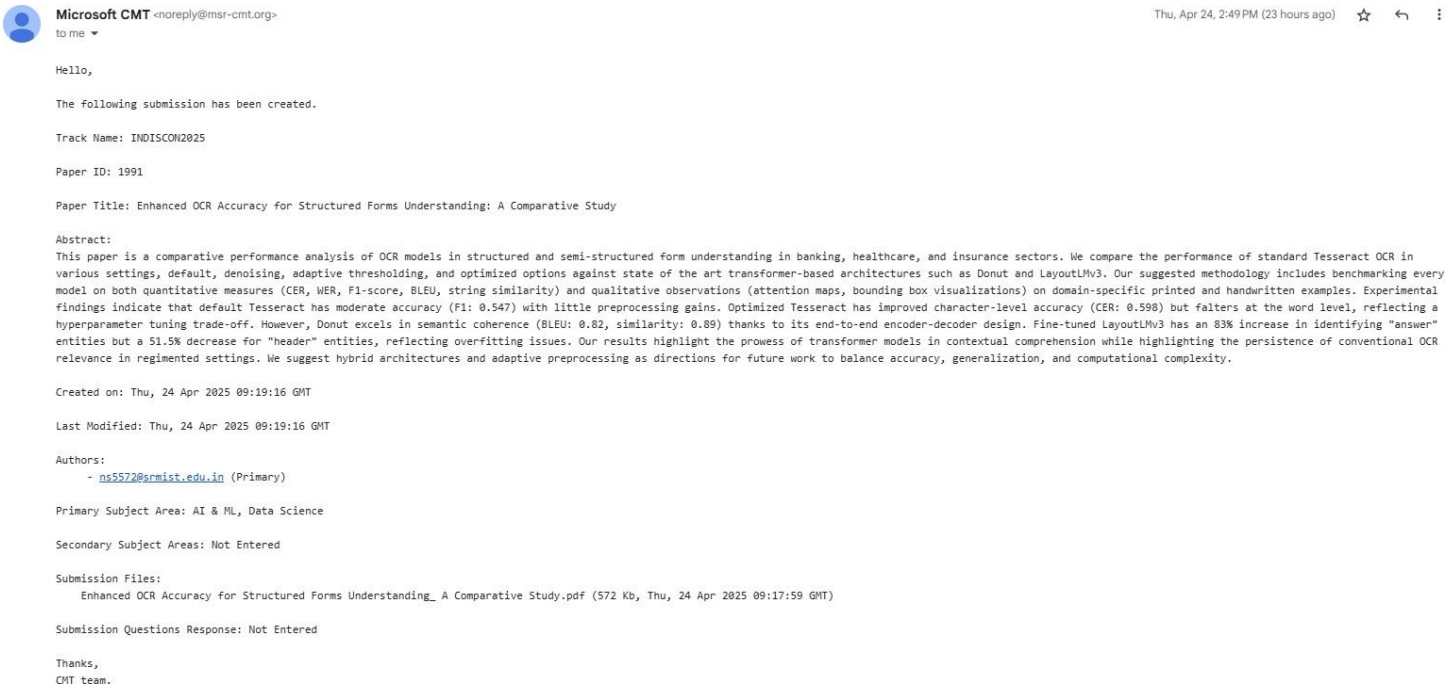
Automated document information extraction is an underlying requirement in banking, healthcare, and insurance industries where large volumes of forms and records need to be digitized, analyzed, and stored cost-effectively. Historically, Optical Character Recognition (OCR) systems have been at the center of performing this task by translating scanned images of printed or handwritten paper into machine-readable formats. Although OCR has facilitated tremendous progress in document digitization and workflow automation, the heterogeneity and variability of actual forms—varying from well-structured templates to semi-structured and noisy layouts—still present enormous challenges for reliable information extraction throughout occurrences, and are therefore best suited for template-based OCR extraction. In these cases, classical OCR engines such as Tesseract can perform well, since the data layout is predictable. Most real-life documents encountered in banking (e.g., KYC forms, loan applications), healthcare (e.g., insurance claims, patient intake forms), and insurance (e.g., claim forms, policy documents) are semi-structured. These documents have variability in field positions, blended content types (printed and handwritten text, checkboxes, tables), and frequently include noise from scanning or manual markups. This variability makes rule-based or template-driven OCR ineffective, resulting in data loss, misinterpretation, or the requirement for heavy manual correction.

Challenges of extracting information from semi-structured documents are three. First, document structure and layout variations require OCR systems to be flexible dynamically because key fields move about or change format from document to document. Second, semi-structured formats tend to have nested or hierarchical data—tables within sections or multi-level lists—which complicates text recognition and semantic linking. Third, the availability of noise, low contrast printing, or handwritten notes will lower the recognition rate even further, especially with more mature OCR engines. Advanced preprocessing methods—in the form of denoising, contrast enhancement, binarization, and normalization—will typically be used to enhance image quality and text legibility before OCR. Moreover, layout analysis algorithms are utilized to segment documents into reasonable areas in a way that the OCR engine will concentrate on the area of interest and enhance extraction accuracy.

Recent deep learning advancements led to transformer-based document understanding models such as LayoutLMv3 and Donut. They integrate visual, spatial, and text data and are thus able to comprehend sophisticated layouts and semantically relate engaged fields despite noise and variability. Unlike typical OCR pipelines, which tend to yield plain text, transformer models have the ability to output structured representations from images, supporting downstream tasks like named entity recognition and key-value extraction. For example, Donut frames document understanding as a vision-to-text generation task without intermediate OCR steps and shows exemplary

APPENDIX B

CONFERENCE SUBMISSION DETAILS



Microsoft CMT <noreply@msr-cmt.org>
to me

Thu, Apr 24, 2:49 PM (23 hours ago)

Hello,

The following submission has been created.

Track Name: INDISCON2025

Paper ID: 1991

Paper Title: Enhanced OCR Accuracy for Structured Forms Understanding: A Comparative Study

Abstract:
This paper is a comparative performance analysis of OCR models in structured and semi-structured form understanding in banking, healthcare, and insurance sectors. We compare the performance of standard Tesseract OCR in various settings, default, denoising, adaptive thresholding, and optimized options against state of the art transformer-based architectures such as Donut and LayoutLMv3. Our suggested methodology includes benchmarking every model on both quantitative measures (CER, WER, F1-score, BLEU, string similarity) and qualitative observations (attention maps, bounding box visualizations) on domain-specific printed and handwritten examples. Experimental findings indicate that default Tesseract has moderate accuracy (F1: 0.547) with little preprocessing gains. Optimized Tesseract has improved character-level accuracy (CER: 0.598) but falters at the word level, reflecting a hyperparameter tuning trade-off. However, Donut excels in semantic coherence (BLEU: 0.82, similarity: 0.89) thanks to its end-to-end encoder-decoder design. Fine-tuned LayoutLMv3 has an 83% increase in identifying "answer" entities but a 51.5% decrease for "header" entities, reflecting overfitting issues. Our results highlight the prowess of transformer models in contextual comprehension while highlighting the persistence of conventional OCR relevance in regimented settings. We suggest hybrid architectures and adaptive preprocessing as directions for future work to balance accuracy, generalization, and computational complexity.

Created on: Thu, 24 Apr 2025 09:19:16 GMT

Last Modified: Thu, 24 Apr 2025 09:19:16 GMT

Authors:
- ns5572@srmist.edu.in (Primary)

Primary Subject Area: AI & ML, Data Science

Secondary Subject Areas: Not Entered

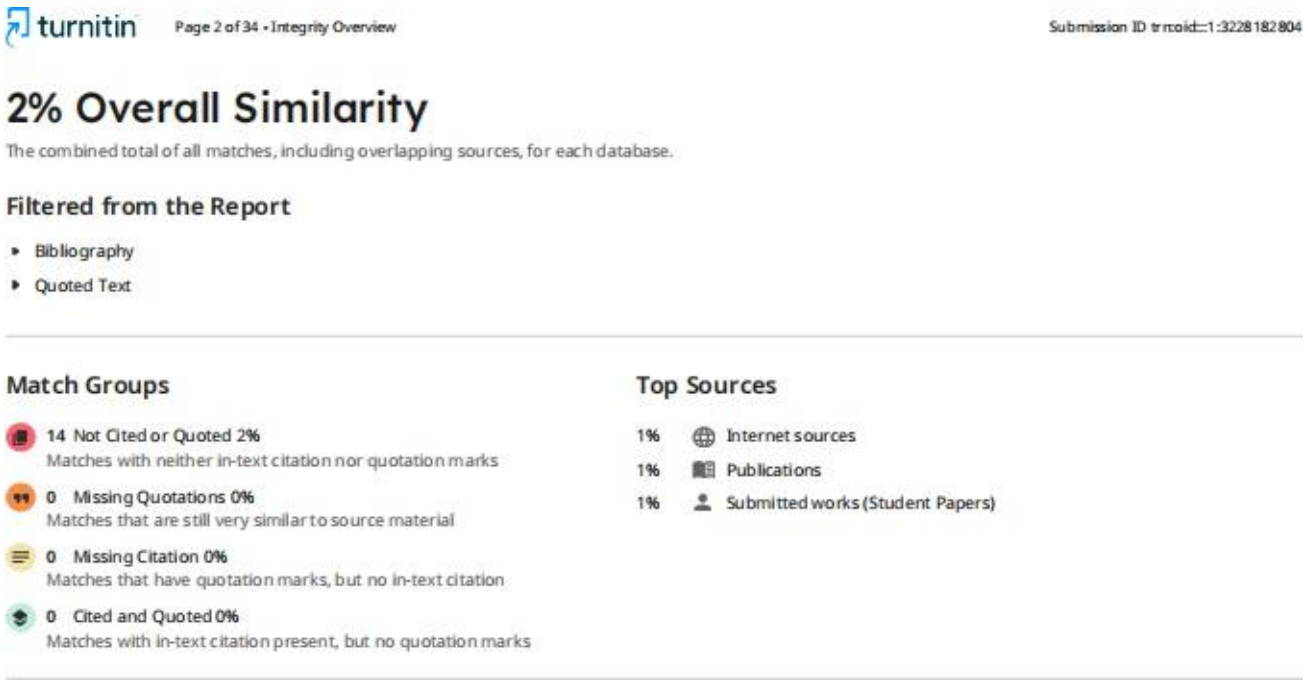
Submission Files:
Enhanced OCR Accuracy for Structured Forms Understanding_ A Comparative Study.pdf (572 Kb, Thu, 24 Apr 2025 09:17:59 GMT)

Submission Questions Response: Not Entered

Thanks,
CMT team.

APPENDIX C

PLAGIARISM REPORT



turnitin Page 2 of 34 - Integrity Overview Submission ID trn:oid:::1:3228182804

2% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Quoted Text

Match Groups

- 14 Not Cited or Quoted 2%
  Matches with neither in-text citation nor quotation marks
- 0 Missing Quotations 0%
  Matches that are still very similar to source material
- 0 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

1% Internet sources
1% Publications
1% Submitted works (Student Papers)

Match Groups

- 14 Not Cited or Quoted 2%
  Matches with neither in-text citation nor quotation marks
- 0 Missing Quotations 0%
  Matches that are still very similar to source material
- 0 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

1% Internet sources
1% Publications
1% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Publication	
Kaiming Tao, Jinru Zhou, Zachary A. Osman, Vineet Ahluwalia, Chiara Sabati, Rob...		<1%
2	Publication	
R. N. V. Jagan Mohan, B. H. V. S. Rama Krishnam Raju, V. Chandra Sekhar, T. V. K. P...		<1%
3	Student papers	
University of Lancaster		<1%
4	Internet	
businessday.ng		<1%
5	Student papers	
Häme University of Applied Sciences		<1%
6	Internet	
dspace.daffodilvarsity.edu.bd:8080		<1%
7	Publication	
Kaiming Tao, Jinru Zhou, Zachary A. Osman, Vineet Ahluwalia, Chiara Sabatti, Rob...		<1%
8	Internet	
fastercapital.com		<1%