

**LARGE LANGUAGE MODEL APPLICATION INTEGRATION FOR
EXTRACTING UNSTRUCTURED DATA**

BY
HO JOE EE

A REPORT
SUBMITTED TO
Universiti Tunku Abdul Rahman
in partial fulfillment of the requirements
for the degree of
BACHELOR OF COMPUTER SCIENCE (HONOURS)
Faculty of Information and Communication Technology
(Kampar Campus)

JAN 2025

COPYRIGHT STATEMENT

© 2025 Ho Joe Ee. All rights reserved.

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to extend my deepest gratitude to my former supervisor, Ts. Dr. Ooi Boon Yaik, for providing me with the invaluable opportunity to undertake this project on developing a framework and application for extracting unstructured data from forms. His advice, practical suggestions, and encouragement really helped me get through the challenges I faced. He was always there for us when we needed guidance and played a key role in connecting me with industry partners, which was crucial for the success of this project. I am truly grateful for his support throughout this journey.

Next, I would also like to thank my current supervisor, Ts Tan Teik Boon, for his valuable input throughout the project, especially in teaching me the proper structure and formatting of a professional report. His clear explanations and attention to detail greatly helped me improve the quality and presentation of my documentation. Additionally, he provided clear guidance on how to conduct proper testing, which enhanced the reliability of the final system.

Besides, I would also like to express my sincere appreciation to my industrial supervisors, Ms. Anna Tan Kiat Yee and Mr. Daniel Chua Wei Yang, for their unwavering support and dedication throughout this final year project. Their time, thoughtful discussions, and valuable input have greatly enriched my work, and I am truly thankful for their commitment. They have been instrumental in ensuring that this project is implemented within GDEX Berhad, significantly enhancing its real-world impact. Thanks to their efforts, this project is now planned for deployment within GDEX's image data extraction processing pipeline, making it even more meaningful.

A heartfelt thank you to GDEX for sponsoring and supporting this project. Their generosity and belief in the potential of this work have been essential to its realization. The full financial sponsorship has not only enabled me to bring this project to life but has also alleviated a significant financial burden, for which I am deeply appreciative.

ABSTRACT

Large Language Models (LLMs) have unlocked new opportunities in processing unstructured information, which is increasingly prevalent in industries such as logistics, healthcare, and finance. To address this growing need despite the inherent inconsistency of LLM, this project presents a data extraction framework that could reliably integrate LLMs into data processing pipelines to extract structured information from unstructured data sources, particularly images containing handwriting. The methodology involved the development of a data processing pipeline that incorporate LLMs to automate data extraction while addressing the unpredictability of LLM outputs. A novel “**Sieve Methodology**” was introduced to enhance output reliability through multiple iterations of validation, comparison, and refinement of the data extracted by the LLMs. This approach was prominent in making sure that the outputs were suitable for downstream processing while optimizing batch processing efficiency. The system also features support for concurrency, confidence scoring and automated template matching with cropping to improve responsiveness and scalability in real-world deployment scenarios. The framework was rigorously tested on handwritten parcel data with printed text, checkbox and signature, which is a common challenge in the logistics industry. The results were significant: the project achieved up to **96.93% of accuracy**, a **reduction in processing time of up to 89.46%**, and a **37.70% reduction in token usage** compared to baseline method where images are directly fed into LLM for processing. These outcomes clearly demonstrate the effectiveness of the proposed framework in enhancing the efficiency and reliability of data extraction processes. In the end, this project successfully developed a dual-functionality system that allows the integration of LLMs as both a standalone tool with GUI and an intermediate module (web API) within automated workflows, highlighting its practical applicability and contribution to the advancement of intelligent data extraction systems.

Area of Study: Distributed System, Large Language Model (LLM)

Keywords: Data Processing Framework, Automation, Data Pipeline Optimization, Token Efficiency, Data Extraction

TABLE OF CONTENTS

TITLE PAGE	i
COPYRIGHT STATEMENT	ii
ACKNOWLEDGEMENTS	ii
ABSTRACT	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES	viii
LIST OF TABLES	xi
LIST OF ABBREVIATIONS	xiii
CHAPTER 1 INTRODUCTION	1
1.1 Project Background	1
1.2 Problem Statement	3
1.3 Motivation	4
1.4 Project Scope and Direction	5
1.4.1 Project Deliverables	5
1.4.2 Project Out of Scope	6
1.5 Objectives	7
1.6 Contributions	8
1.7 Report Organization	9
CHAPTER 2 LITERATURE REVIEW	10
2.1 Practices in Extracting Unstructured Data from Documents	10
2.2 Previous works on LLM-integrated Systems and their Framework	11
2.3 Application Integration Framework for LLMs in Image Data Extraction and Processing	13
2.4 Reviews on Existing AI-enhanced Data Extraction System	14
2.4.1 Google Document AI	14
2.4.2 Amazon Textract	15
2.5 Reviews on Existing LLM-based Data Extraction System	16
2.5.1 Unstructured	17
2.5.2 Unstract	18
2.5.3 LlamaIndex	19
2.5.4 Extracta.ai	20
	iv

2.6	Concluding Remark	21
CHAPTER 3 SYSTEM METHODOLOGY / APPROACH		23
3.1	Overview of Solution	23
3.2	System Architecture Diagram	24
3.3	User Requirements	26
3.3.1	Functional Requirements (FR)	26
3.3.2	NON-FUNCTIONAL REQUIREMENTS (NFR)	28
3.4	Use Case Diagram and Use Case Description	30
3.4.1	Use Case Diagram	30
3.4.2	Key Use Case Description	32
3.5	Activity Diagram	38
3.6	General Development Methodologies and General Work Procedures	44
3.7	Timeline and Gantt Chart	45
CHAPTER 4 SYSTEM DESIGN		46
4.1	System Flowchart	46
4.2	LLM Application Integration Framework for Unstructured Data Extraction	48
4.2.1	Data Extraction Framework	48
4.2.2	Application Pipeline Design for LLM Integration	50
4.3	System Block Diagram	56
4.4	Key Components Specification and Approaches for Data Processing Pipeline	58
4.4.1	Data Preparation Component	58
4.4.2	Data Extraction and Analysis Component	63
4.4.3	Data Post Processing Component	69
4.5	Components Specification and Approaches for Supporting System	74
4.5.1	Input Source Component	74
4.5.2	Data Storage and Output Component	75
4.5.3	Template Management Component	75
4.6	System Optimization Design	76
4.6.1	Thread Management	76
4.6.2	Asynchronous Operations	76
4.6.3	Memory and Resource Management	77

CHAPTER 5 SYSTEM IMPLEMENTATION	78
5.1 Hardware Setup	78
5.2 Software Setup	79
5.2.1 Integrated Development Environment (IDE) and Languages	79
5.2.2 Software Specification	80
5.2.3 Input Source Setup	83
5.2.4 Database and File Operation Setup	84
5.2.5 API Server Setup	85
5.3 Settings and Configuration	86
5.3.1 Installation and Setup Steps	86
5.3.2 Deployment Details	91
5.3.3 Distribution Method	91
5.3.4 License	92
5.3.5 Versioning	92
5.4 System Operations as Graphical User Interface	93
5.4.1 System Navigation	93
5.4.2 Template Manager	94
5.4.3 AI Model Manager	99
5.4.4 Input Source Setup	102
5.4.5 System Settings	104
5.4.6 Data Processing	104
5.4.7 Past Output Documents	108
5.4.8 System Output as Subsequent Input of Data Pipeline (API)	110
5.5 Implementation Issues and Challenges	113
5.6 Concluding Remark	114
CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION	115
6.1 System Verification and Validation Strategy Overview	115
6.2 System Performance Metrics	116
6.3 Functional Testing Setup and Result	118
6.3.1 Use Case Testing	118
6.3.2 Input Validation Testing	123
6.4 Non-Functional Testing Setup and Result	129
6.4.1 Testing Setup	129

6.4.2	Data Extraction Accuracy	130
6.4.3	Output Delivery Time	139
6.4.4	Resource Utilization	143
6.5	User Acceptance Testing (UAT)	146
6.6	Project Challenges	148
6.7	Overall Performance and Objective Evaluation	149
6.8	Concluding Remark	150
CHAPTER 7 CONCLUSION AND RECOMMENDATION		151
7.1	Conclusion	151
7.2	Future Considerations and Recommendations	152
REFERENCES		154
POSTER		157

LIST OF FIGURES

Figure Number	Title	Page
Figure 2.2.1	LLM as the terminal module	12
Figure 2.2.2	LLM as the intermediate module	12
Figure 3.2.1	System Architecture Diagram	24
Figure 3.4.1.1	Use Case Diagram	31
Figure 3.5.1	UC01 Manage Template Activity Diagram	38
Figure 3.5.2	UC02 Manage AI Model Activity Diagram	39
Figure 3.5.3	UC03 Manage Input Source Activity Diagram	40
Figure 3.5.4	UC04 Process Image and Extract Data Activity Diagram	41
Figure 3.5.5	UC05 View Past Output Documents & Results Activity Diagram	42
Figure 3.5.6	UC06 Serve API Endpoint Activity Diagram	43
Figure 3.6.1	Rapid Application Development (RAD) Model	44
Figure 3.7.1	Gantt Chart	45
Figure 4.1.1	System Flowchart	46
Figure 4.2.1.1	Data Extraction Framework	48
Figure 4.2.2.1	System Pipeline Design Part 1	50
Figure 4.2.2.2	System Pipeline Design Part 2	51
Figure 4.2.2.3	System Pipeline Design Part 3	52
Figure 4.3.1	System Block Diagram	56
Figure 4.4.3.1	Simplified Sieve Methodology Flowchart	69
Figure 5.3.1.1	Installer Destination Location Setup	87
Figure 5.3.1.2	Start Menu Folder Setup	87
Figure 5.3.1.3	Additional Tasks Setup	87
Figure 5.3.1.4	Installation Permission	88
Figure 5.3.1.5	Installing Application	88
Figure 5.3.1.6	Installation Completed	88
Figure 5.3.1.7	App in Start Menu	89
Figure 5.3.1.8	Desktop Shortcut in Window Home Page	89
Figure 5.3.1.9	App in Installed Apps settings	90
		viii

Figure 5.3.1.10	Uninstall App	90
Figure 5.3.1.11	Permission dialog box to remove application	90
Figure 5.3.1.12	Uninstallation completed	91
Figure 5.4.1.1	Collapsed Navigation Sidebar	93
Figure 5.4.2.1	Template Manager Screen	94
Figure 5.4.2.2	Template Manager Screen - Create New Extraction Template	95
Figure 5.4.2.3	Template Manager Screen – Upload Template Image	95
Figure 5.4.2.4	Template Manager Screen – Define Template Data and ROIs	96
Figure 5.4.2.5	Template Manager Screen – Define Field Properties	97
Figure 5.4.2.6	Template Manager Screen – Error Message	97
Figure 5.4.2.7	Template Manager Screen – Defined ROIs and Edit Buttons	98
Figure 5.4.2.8	Template Manager Screen – Auto Template Creation	98
Figure 5.4.2.9	Template Manager Screen – Successfully Created Template	99
Figure 5.4.3.1	AI Model Manager Screen – No API Key found	99
Figure 5.4.3.2	AI Model Manager Screen – Toggle Menu Option	100
Figure 5.4.3.3	AI Model Manager Screen – Add Public Model	101
Figure 5.4.3.4	AI Model Manager Screen – Add Custom Model	101
Figure 5.4.4.1	Input Source Setup Screen	102
Figure 5.4.4.2	Input Source Setup Screen – Example Configuration	103
Figure 5.4.4.3	Input Source Setup Screen – Populated Source	103
Figure 5.4.5.1	System Settings Screen	104
Figure 5.4.6.1	Image Processing and Extraction Hub	105
Figure 5.4.6.2	Data Extraction Screen	105
Figure 5.4.6.3	Data Extraction Screen – Start Processing	106
Figure 5.4.6.4	Data Extraction Screen – Result Screen	106
Figure 5.4.6.5	Data Extraction Screen – Display Output	107
Figure 5.4.6.6	Image Processing and Extraction Hub – Manage Process	108
Figure 5.4.7.1	Files populated in Output Documents Screen	109
Figure 5.4.7.2	Past Output Document Screen – File Search	109
Figure 5.4.7.3	Past Output Document Screen – View File Content	110
Figure 5.4.8.1	API Endpoint Screen – Guideline Part I	111
Figure 5.4.8.2	API Endpoint Screen – Guideline Part II	111
Figure 5.4.8.3	Start API Endpoint Connection	112

Figure 5.4.8.4	Stop API Server	112
Figure 6.3.1	GDEX Berhad CN Billing Copy Template	118
Figure 6.3.2.1	Test Data for IVTC-001	124
Figure 6.3.2.2	Test Data for IVTC-002	124
Figure 6.3.2.3	Test Data for IVTC-003	125
Figure 6.3.2.4	Test Data for IVTC-004	125
Figure 6.3.2.5	Test Data for IVTC-005	125
Figure 6.3.2.6	Test Data for IVTC-006	125
Figure 6.3.2.7	Test Data for IVTC-007	125
Figure 6.3.2.8	Test Data for IVTC-008	126
Figure 6.3.2.9	Test Data for IVTC-009	126
Figure 6.3.2.10	Test Data for IVTC-010	126
Figure 6.3.2.11	Test Data for IVTC-011	126
Figure 6.3.2.12	Test Data for IVTC-012	127
Figure 6.4.1	Data that needs to be extracted from billing copy	129
Figure 6.4.2.1	Bar chart to compare accuracy of approaches across 3 LLMs	133
Figure 6.4.2.2	Bar chart to compare field-level accuracy of approaches	134
Figure 6.4.2.3	Bar Chart of Average Accuracy for Different Types of Data Extraction	137
Figure 6.4.3.1	Bar Chart of Processing Time for Batch Processing	141
Figure 6.5.1	GDEX Berhad Process Flow Map with System Integration	146

LIST OF TABLES

Table Number	Title	Page
Table 3.4.2.1	UC01 Manage Template Use Case Description	33
Table 3.4.2.2	UC02 Manage AI Model Use Case Description	34
Table 3.4.2.3	UC03 Manage Input Source Use Case Description	35
Table 3.4.2.4	UC04 Process Image and Extract Data Use Case Description	36
Table 3.4.2.5	UC05 Access Past Output Documents Use Case Description	37
Table 3.4.2.6	UC06 Serve API Endpoint Use Case Description	37
Table 4.4.2.1	Types of Methods to Extract Data	63
Table 5.1.1	Specification of Processing Unit	78
Table 5.2.1.1	IDE and Git	79
Table 5.2.2.1	External Libraries/ Packages	82
Table 5.2.3.1	Input Sources	83
Table 6.2.1	System Performance Metrics	117
Table 6.3.1.1	Use Case Test 01	119
Table 6.3.1.2	Use Case Test 02	120
Table 6.3.1.3	Use Case Test 03	121
Table 6.3.1.4	Use Case Test 04	121
Table 6.3.1.5	Use Case Test 05	122
Table 6.3.1.6	Use Case Test 06	122
Table 6.3.2.1	Key Input Validation Test	124
Table 6.3.2.2	Input Validation Test Data	127
Table 6.4.1	Type of data extraction and description for each field	130
Table 6.4.2.1	Accuracy of different data fields with direct approach	131
Table 6.4.2.2	Accuracy of different data fields using system pipeline	132
Table 6.4.2.3	Comparison of average increased accuracy for 3 LLMs	132
Table 6.4.2.4	Average accuracy of different data fields using both approaches	133
Table 6.4.2.5	Overall Accuracy Improvement	135
Table 6.4.2.6	Average accuracy for each type of data for extraction	137
Table 6.4.3.1	Processing time of 3 LLMs using different approaches	139

Table 6.4.3.2	Average processing time and difference	139
Table 6.4.3.3	Total and average time per image for batch processing	140
Table 6.4.3.4	Average time per image for batch processing for different approaches	141
Table 6.4.3.5	Average processing time for difference approaches	141
Table 6.4.4.1	Token of 3 LLMs using different approaches	143
Table 6.4.4.2	Comparison of token usage of 3 LLMs using different approaches	143
Table 6.4.4.3	Average total token used for different approaches	143
Table 6.4.4.4	Comparison of cost for 3 LLMs using different approaches	144
Table 6.4.4.5	Average total cost used for different approaches	144

LIST OF ABBREVIATIONS

<i>LLM</i>	Large Language Model
<i>BERT</i>	Bidirectional Encoder Representations from Transformers
<i>BLOOM</i>	BigScience Large Open-science Open-access Multilingual Language Model
<i>GPT</i>	Generative Pre-Trained Transformers
<i>LLAMA</i>	Large Language Model Meta AI
<i>RoBERTa</i>	Robustly Optimized BERT Approach
<i>T5</i>	Text-to-Text Transfer Transformer
<i>CV</i>	Computer Vision
<i>OCR</i>	Optical Character Recognition
<i>GUI</i>	Graphical User Interface
<i>DICOM</i>	Digital Imaging and Communications in Medicine
<i>AI</i>	Artificial Intelligence
<i>NLP</i>	Natural Language Processing
<i>OpenCV</i>	Open-Source Computer Vision Library
<i>API</i>	Application Programming Interface
<i>ORB</i>	Oriented FAST and Rotated BRIEF
<i>ROI</i>	Regions of Interest
<i>IP</i>	Internet Protocol
<i>RAG</i>	Retrieval-Augmented Generation
<i>IoU</i>	Intersection over Union
<i>UAT</i>	User Acceptance Testing

CHAPTER 1 INTRODUCTION

In this chapter, we present the background, problem statements and motivations of our project, objectives to be achieved, project scope and direction, our contributions to the field of large language models and distribution system, as well as the outline of the thesis.

1.1 Project Background

In today's increasingly data-driven environment, organizations rely heavily on unstructured information, such as images, handwritten notes, and other non-standardized formats, for critical decision-making and operational efficiency. However, converting this unstructured data into meaningful, structured information remains a challenging and labour-intensive task. This challenge is especially pronounced in sectors such as logistics, healthcare, education, and finance, where speed and accuracy are essential. Established approaches often depend on manual data entry or conventional Optical Character Recognition (OCR) technologies, which frequently struggle with complex layouts or low-quality inputs, resulting in limited accuracy.

In fact, Large Language Models (LLMs) such as *BERT*, *BLOOM*, *GPT*, *LLAMA*, *RoBERTa*, or *T5*, as in [1] and [2], have fundamentally transformed how unstructured data is processed and interpreted. Their recent advancements showcase remarkable capabilities in understanding and generating human-like language, enabling the automation of complex data extraction tasks, even when information is implied rather than explicitly stated in the input (e.g., from images).

Although the idea is simple, its implementation is not. Despite their potential, integrating LLMs into existing applications has proven hard due to their non-deterministic nature, which in turn undermines the reliability of downstream processes depending on structured and accurate data. For example, LLM might generate different outputs even when given the same input prompt multiple times, or they might hallucinate throughout the process if the prompt is unclear and return the inaccurate result.

From our literature review, there is a lack of well-defined application integration framework for LLM system that detailed this problem. Despite the rise of frameworks

such as LangChain and Haystack that support the integration of LLMs into broader systems, there remains a critical gap in building pipelines that extract structured data from unstructured, image-based content with high accuracy, consistency, and efficiency.

Hence, to address this, this project “Large Language Model Application Integration for Extracting Unstructured Data” was conceived in light of these challenges of unstructured data extraction. By developing a system with framework that embed LLMs within conventional data processing systems, this project aims to bridge the gap between cutting-edge AI capabilities and the practical needs of industries dealing with unstructured data. This proposed framework will include new mechanisms for filtering, validating, and optimizing LLM outputs to provide high-quality, accurate, and consistent data extraction.

One of the key applications of this project is in the logistics industry, where handwritten parcel data must be digitized accurately and efficiently. Currently, manual data entry is time-consuming and error-prone, leading to inefficiencies and increased operational costs. Through automation by means of LLMs, time and resources will be saved with regards to processing parcel information, and eventually, there will be an enhancement in the accuracy of data fed into the systems. This project aims to unlock new levels of efficiency and accuracy for industries that rely on timely and precise information.

1.2 Problem Statement

Lack of Solidly Established Image-to-Structured Pipelines Framework with LLMs Integration

While powerful LLMs and CV/OCR models exist individually, they are not purpose-built for integrating LLMs into workflows that involve image-based unstructured data. There is no standardized or production-grade framework that unifies them into a cohesive pipeline. These workflows demand tight coordination between preprocessing, LLM prompting, and post-processing logic. There is no well-established design pattern or end-to-end framework to address this integration in an industrially reliable way.

Inconsistency and Inaccuracy in Data Extraction by LLMs

LLMs, by design, are probabilistic. The non-deterministic nature of LLMs results in varying outputs even when the same input is provided multiple times. This randomness complicates the determination of the correct output. For example, when extracting information from handwritten text in the image, LLMs might mistakenly interpret a character as either the number “1” or the letter “l.”. This issue becomes particularly pronounced in contexts that require high accuracy and consistency, such as logistics, healthcare, and finance. When extracting handwritten data from parcels, inconsistencies in interpreting postcodes, can lead to errors in shipment processing, and potential financial losses. Current solutions lack effective and thorough strategies for **enforcing deterministic behavior**, schema alignment, and error correction.

Inefficient Resource Utilisation in LLM-based Data Extraction

Current approaches to data extraction using LLMs, particularly from images, are compute-intensive and involving significant resource consumption. Each analysis performed by the LLM consumes tokens, which directly translates into higher operational costs. Frameworks tend to focus on LLM orchestration but overlook optimizations such as minimal-token prompting, and hybrid use of classical models (e.g., regex, rules) with LLMs to reduce cost and latency. Furthermore, the time required to process each image, particularly when dealing with large datasets, can be substantial. It makes the widespread adoption of LLMs in data-intensive industries less feasible, as the costs can quickly escalate, especially when accuracy is also a priority.

1.3 Motivation

The motivation for this project stems from the **growing demand for automated systems that can efficiently handle and process unstructured data**. The widespread generation of unstructured data in sectors such as logistics, healthcare, and finance, where information often exists in complex forms like handwritten notes and non-standard documents. These limitations in existing system create bottlenecks in data processing pipelines, highlighting the need for more effective and intelligent systems capable of understanding context and extracting meaningful information from challenging inputs. Automating the extraction and structuring of this data is critical to meet the increasing demands for speed, accuracy, and efficiency in data processing workflows.

While LLMs offer strong capabilities in recognizing patterns and extracting implicit information, their practical integration requires careful design to ensure consistent and accurate results. The motivation lies in **creating a framework that overcomes LLM limitations to unlock its full potential**, such as inefficiencies in token usage and processing delays, by optimizing system performance for real-time, scalable applications.

Besides, this project is driven by the initiative to develop techniques that enable businesses to **reduce operational costs, minimize errors, and scale efficiently**. By improving batch processing of image data and reducing token consumption, the system aims to make LLM-based solutions not only more effective but also economically viable for large-scale deployment.

1.4 Project Scope and Direction

The “Large Language Model Application Integration for Extracting Unstructured Data” project focuses on creating an inclusive system with a framework that leverages the power of Large Language Models (LLMs) to accurately extract and process unstructured data, such as handwritten parcel information, from images. The scope of the project includes the design, development, and deployment of LLMs integration framework into a system. This section will outline the components that will be delivered as part of the project, as well as those that are outside the project’s scope.

1.4.1 Project Deliverables

The project will design and implement an innovative **framework and data pipeline, including Sieve methodology that allows existing applications to seamlessly integrate LLMs**. This proposed framework will address the inherent non-deterministic nature of LLMs, ensuring that the data extracted from images is consistent, accurate, and structured in a usable format. This framework will involve multiple iterations of validation, comparison, and refinement of the data extracted by LLMs, ensuring consistency and accuracy.

Besides, an image processing **pipeline** will be developed to **automate the extraction of data** from images, particularly focusing on data that includes handwritten elements. The pipeline will incorporate techniques such as image processing, image quality checking, template matching, image cropping, batch processing, and validation using sieve approach to improve the accuracy and efficiency of data extraction. This process will reduce the need for manual data entry, especially in scenarios where the handwritten data needs to be converted into digital formats for storage in databases.

The focus will be on creating a dual-functionality system capable of operating both as a standalone tool for direct user interaction and as an intermediate module within automated workflows. The project will deliver a **user-friendly graphical user interface (GUI)**, particularly a **desktop application** that enables users to interact with the system intuitively. At the same time, the software will also function as an **intermediate module** (web API) that can be integrated into larger automated workflows, receiving and processing images before passing the extracted data to subsequent modules.

In the end, the project will produce a fully functional system that integrates LLMs into existing applications, facilitating the accurate extraction of unstructured data from images. This system will enhance data processing workflows, reduce manual entry efforts, and provide a scalable solution for industries dealing with large volumes of unstructured data. Providing this, organizations will have a powerful tool that simplifies the integration of LLMs into their existing workflows, thus improving operational efficiency and reducing manual labour.

1.4.2 Project Out of Scope

- The project will use pre-existing or pre-trained LLMs (e.g., GPT models) and will not involve the development or training of new language models from scratch; it will focus solely on integrating existing LLMs into the proposed framework.
- It will not address the extraction of data from non-image-based unstructured data sources, such as audio or video.
- It will not address the accurate extraction of data from images with overly poor quality.
- The system will not support extraction from image types or data formats that are not predefined or specified during the requirement gathering phase. For example, specialized image formats like DICOM (medical imaging) or extremely low-resolution images are out of scope.
- The system will not include advanced image recognition features beyond basic LLM-based text extraction, barcode scanning, and metadata extraction. Complex tasks such as object detection, facial recognition, or scene analysis are out of scope.
- It will not address the extraction of certain data from images such as watermarks, hidden text, stamps, seals and signatures.

1.5 Objectives

The primary objective of this project is to develop a solution that ensure consistent and accurate data extraction by providing more reliable outputs that can be seamlessly integrated into existing systems. It brings to a scalable framework that can harness the strengths of LLMs while mitigating their weaknesses, particularly in high-stakes environments like logistics and data processing.

The main objective can be divided into few sub-objectives as follows:

- **Facilitate reliable integration of LLMs into existing applications by designing a framework to reduce the variability of LLMs responses.** It aims to ensure that the outputs conform to required formats and are suitable for downstream processing in various industry-specific solutions. By transforming unstructured data, such as text from images or handwritten notes, into precise, structured formats, the framework will enable applications to effectively utilize the extracted information.
- Ensure consistent data extraction from unstructured sources using LLMs and **achieve at least a 10% improvement in accuracy** over existing methods with automated validation and correction mechanisms. It aims to guarantee that the extracted data meets predefined quality standards across a range of test cases. This accuracy will be measured by comparing the extracted data against manually verified results from a diverse set of 100 test images.
- Optimize efficiency of LLM-based data extraction pipeline by reducing **processing time and resource consumption by at least 25%**. This can be achieved via the implementation of batch processing techniques, image cropping technique and optimized data submission algorithm in Sieve Methodology within the framework to streamline the handling of large volumes of data. By minimizing token usage and decreasing processing time, the project aims to make the integration of LLMs more time-effective and scalable.

1.6 Contributions

This project tackles the critical issue of manual data entry in industries such as logistics by **automating the extraction of handwritten information using Large Language Models (LLMs)**. By replacing time-intensive and error-prone manual processes, the solution significantly **reduces operational costs** and **improves accuracy**. Its successful deployment in GDEX Berhad Sdn. Bhd's parcel processing pipeline underscores its real-world efficacy and potential for broader adoption across industries like healthcare and finance.

Notably, the solution has been successfully integrated into the parcel information processing pipeline of **GDEX Berhad Sdn. Bhd**, demonstrating its **real-world applicability** and value to the industry. Hence, by integrating LLMs into existing workflows, the project can indeed enhance data reliability. More than just an improvement, it represents a transformative leap forward for industries. It provides a chance to transform the industries with automation and accuracy. The impact extends beyond logistics, with applications in healthcare, finance, and beyond.

The project does not just stop at automating data extraction; it offers a **scalable and user-friendly solution that organizations of any size can adopt**. The development of a Graphical User Interface (GUI) allows users to easily interact with the system, making the technology accessible even to those without technical expertise. This democratization of advanced Artificial Intelligence (AI) tools means that small and medium-sized enterprises can benefit from the same efficiency and accuracy as large corporations, levelling the playing field and fostering innovation across industries.

On a broader scale, the significance of this project lies in **empowering people and organizations to achieve more with less effort**. By reducing the need for manual data entry, it frees up human resources for more strategic and creative tasks. It can lead to higher job satisfaction and productivity. In fields like healthcare, in which accurate data can mean the difference between life and death, this project's impact could be profound. Via ensuring that data is processed accurately and quickly, the project can support better decision-making, thus improving patient outcomes and saving lives. As AI continues to evolve, the ability to integrate these models into everyday applications will be the key for staying competitive.

1.7 Report Organization

This report is structured into several chapters to present the development, implementation, and evaluation of the proposed LLM integration framework for unstructured data extraction.

Chapter 1 introduces the project by outlining its background, problem statement, motivation, scope, and specific objectives. It also discusses the project contributions and provides an overview of the entire report structure.

Chapter 2 offers a comprehensive literature review covering existing technologies and methodologies relevant to data extraction, Optical Character Recognition (OCR), and LLM-based systems. It explores the strengths and limitations of current solutions and sets the foundation for the framework by identifying research gaps and opportunities.

Chapter 3 details the system methodology, including the overall solution architecture and user requirements. This chapter also presents use case diagrams and activity diagrams to describe the intended system behaviour and user interaction.

Chapter 4 tunnels into the system design by presenting flowcharts, architectural models, and detailed explanations of each component within the data extraction framework. It highlights how LLMs are integrated and explains the core Sieve Methodology used to enhance output reliability.

Chapter 5 discusses the implementation process, providing technical insights into the developed system. This includes GUI functionality, integration details, and software distribution methods. It also outlines how the system operates in real-world scenarios.

Chapter 6 focuses on system evaluation, where performance, accuracy, and efficiency of the implemented framework are assessed through functional and non-functional testing. The results are analyzed to validate the system's effectiveness in meeting the predefined objectives.

Finally, **Chapter 7** concludes the report with a summary of key achievements, reflections on project limitations, and potential areas for future enhancement. This chapter wraps up the project by emphasizing its contributions and applicability to real-world problems in logistics and other domains.

CHAPTER 2 LITERATURE REVIEW

This section reviews relevant literature and existing technologies related to Optical Character Recognition (OCR) systems, serving as a comparative foundation for evaluating the text extraction capabilities of Large Language Models (LLMs). It also examines various aspects of LLMs, including their architecture, applications in data extraction, and the challenges associated with their integration. Additionally, both traditional data extraction systems and LLM-based data extraction solutions are analyzed to provide a comprehensive understanding of the current technological landscape.

2.1 Practices in Extracting Unstructured Data from Documents

Optical Character Recognition (OCR) is a widely adopted technology in the field of data extraction, particularly for converting different types of documents, including scanned paper documents, images of documents, and image-only PDFs, into editable and searchable data formats [3]. OCR has become an integral part of text-based image retrieval systems, enabling the extraction of text from images which can then be indexed and searched efficiently. OCR can convert text from images or scanned documents into machine-readable formats, which is essential for applications such as document analysis, scene text recognition, and information retrieval from images [4].

OCR systems work by analysing the structure of the document image, segmenting it into text blocks, paragraphs, words, and characters. The paper [4] highlights several popular OCR libraries, including Tesseract, Keras, Easy OCR, and Paddle OCR, evaluating their performance on diverse datasets encompassing different text types, languages, and image qualities. The studies also illustrate the breakthrough of OCR technology by integrating deep learning approaches with traditional OCR engines [4]. It has resulted in improved accuracy and versatility, particularly in text detection and classification tasks. The study demonstrates how the combination enables more accurate recognition of complex and varied text types, including those found in noisy or low-quality images.

Besides, the integration of SAST for detection and TransformerOCR for recognition also represents a significant advancement in OCR capabilities, particularly in dealing

with complex and variable text layouts [5]. This combination allows the system to accurately detect and recognize text even in challenging scenarios, such as when text is multi-directional or embedded in intricate design elements on the book cover.

Despite its widespread use and its advancement, OCR technology has several limitations. Conventional OCR systems are highly dependent on the quality of the input image and are prone to errors when dealing with poor quality scans, images with low contrast, or handwritten text, which is mentioned in several studies including [3] [4] and [5]. Even if there is paper introduces OCR-Diff framework, which integrates a generative diffusion model with deep learning techniques to improve the recognition accuracy of text from low-resolution images [6], still, OCR systems are generally limited to recognizing text and cannot interpret the meaning or context behind the text. This limitation is significant in industries where understanding the context of the extracted data is crucial, such as in logistic, legal or medical fields.

While OCR has traditionally been limited to recognizing and converting text from simple, well-structured documents, this work pushes the boundaries by using LLMs to handle more complex, unstructured data. LLMs offer a level of understanding and context [7], in which the traditional OCR systems lack, enabling more accurate data extraction from images that contain handwritten notes, irregular fonts, or text embedded in complex graphical layouts. The integration of LLMs into the data processing pipeline represents a significant breakthrough, allowing for the extraction of not just text, but also the underlying meaning and context from images.

2.2 Previous works on LLM-integrated Systems and their Framework

Given the limitations of traditional OCR systems illustrated in Section 2.1, particularly in handling complex, unstructured data, the project adopts LLMs as a more advanced alternative. LLMs provide a deeper level of understanding and are capable of processing and extracting meaningful data from images that OCR systems would struggle with. It can go beyond simple text extraction, so that the extracted data is both meaningful and contextually relevant. Thus, this project focuses on the framework that can support integration of LLMs into the existing applications, especially for extracting unstructured data.

Numerous studies have explored the integration of Large Language Models (LLMs) within various system architectures. In general, these implementations fall into two categories based on the role of the LLM: as a terminal module or as an intermediate module. When used as a terminal module, the LLM functions as the final stage in the data processing pipeline, and its output, typically in textual form, is delivered directly to the user without further processing, as illustrated in Figure 2.2.1[8].

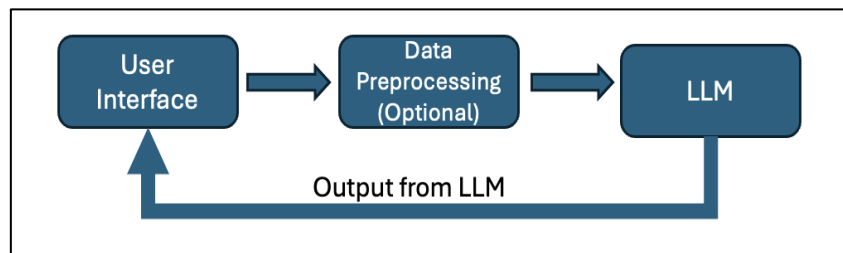


Figure 2.2.1 LLM as the terminal module

Source: Adopted from [8]

Conversely, when used as an intermediate module (Figure 2.2.2), the LLM's output is passed along to subsequent modules for additional processing. In this setup, the LLM serves as a bridge rather than an endpoint, with its output forming part of a more complex data pipeline[8].

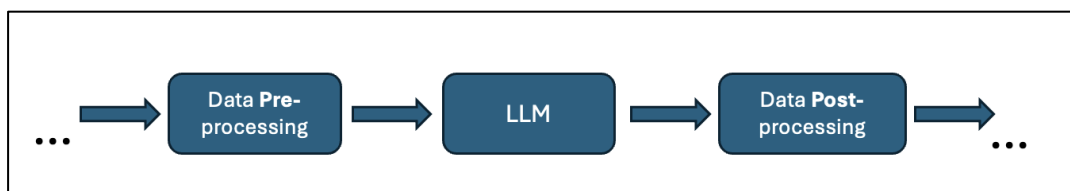


Figure 2.2.2 LLM as the intermediate module

Source: Adopted from [8]

Despite the widespread use of LLMs in both roles, many studies, as noted in [8], fail to thoroughly explain how LLMs are integrated into system pipelines. A significant challenge remains the non-deterministic behavior of LLMs. This stochastic nature is well-documented by providers like OpenAI, whose API documentation highlights the inherent variability in output, even when parameters like seed and temperature are fixed [9]. Although advances such as prompt engineering and structured outputs (e.g., JSON)

might suggest that LLM integration is straightforward, results often vary slightly with each call. This inconsistency poses a major obstacle for applications that depend on predictable, structured outputs for further processing. Minor variations in content, format, or structure can break downstream systems, limiting the reliability of LLMs in structured data workflows. Moreover, developers have limited control over hallucinations, outdated facts, or fabricated content, all of which are exacerbated by the LLM's dependence on static training data [10]. Fine-tuning LLMs to avoid such issues is possible but often cost-prohibitive for simple tasks.

Several factors contribute to this unpredictability. LLMs are probabilistic by design, generating outputs based on learned distributions of language patterns. Even with identical inputs, the use of probability introduces natural variation. Additionally, the transformer architecture, which underpins most modern LLMs, relies on self-attention mechanisms to capture relationships between words. Models are typically trained on massive text corpora and further refined through reinforcement learning based on human feedback. This iterative training process, while improving overall quality, also introduces additional randomness, reinforcing the inherently stochastic nature of LLM outputs [10].

2.3 Application Integration Framework for LLMs in Image Data Extraction and Processing

The relevant work identified involves applying LLMs in the domain of image data extraction and processing to perform text-to-SQL conversion, specifically in converting structured forms into relational databases without relying on pre-defined templates[11]. In this study, Anbarasi et al. [11] propose a pipeline that extracts semantic relationships from forms and converts the extracted data into SQL queries. The proposed pipeline employs FUDGE, which uses graph-based representation and dynamic graph editing to comprehend the form's structure, PaddleOCR to extract text from the identified form elements, and finally, an LLM to convert the text into SQL queries. For the text-to-SQL conversion, GPT-3 demonstrated superior performance compared to other models, especially in handling complex queries[11].

Additionally, the integration of LLMs in image data extraction and processing, as demonstrated in another study[12], involves a multi-stage pipeline combining web

scraping, OCR, and semantic data extraction techniques. The process begins with scraping scanned procurement documents using Scrapy and BeautifulSoup, followed by converting PDFs to images and applying PaddleOCR for text recognition. Preprocessing steps such as skew correction, noise removal, and binarization are employed to enhance OCR accuracy. Extracted text is then passed through a Retrieval-Augmented Generation (RAG) pipeline using the Mistral LLM, where the text is chunked, embedded into a vector database, and queried to extract structured data such as bidder names and financial offers[12]. This synergistic use of PaddleOCR and LLMs like Mistral allows for accurate and scalable transformation of unstructured image-based documents into structured formats suitable for analysis.

Despite its overall effectiveness, the methodology presents several limitations. The OCR system, though generally reliable, performs poorly on low-resolution or degraded scans, resulting in inaccurate text extraction even after applying preprocessing techniques. Furthermore, the approach faces challenges in handling multilingual documents, especially Arabic, due to insufficient support within the OCR model. Additionally, the paper lacks a clear explanation of how to utilize LLMs in a consistent and dependable manner, particularly in the context of data extraction pipelines where stability and predictability are crucial.

2.4 Reviews on Existing AI-enhanced Data Extraction System

This section reviews several notable AI-enhanced image data extraction systems, including Google Document AI, and Amazon Textract. Each of these systems brings unique approaches and capabilities to the table, catering to different use cases and levels of complexity. These tools combine OCR (Optical Character Recognition) with natural language processing (NLP) to extract structured information from various types of documents such as invoices, forms, and receipts.

2.4.1 Google Document AI

Google Document AI is a cloud-based solution that employs Google's AI and machine learning infrastructure to parse and understand scanned documents. It combines state-of-the-art OCR with pre-trained models for layout analysis, entity extraction, and semantic classification[13]. The tool offers specialized parsers for documents such as invoices, tax forms, and identity cards, making it particularly useful for enterprises

dealing with large volumes of standardized paperwork. One of the strengths of Document AI lies in its ability to maintain the structural and semantic integrity of documents through layout-aware models, enabling more accurate interpretation of tables, fields, and hierarchical content. Advanced features include handwriting recognition, font-style detection, and extraction of selection marks like checkboxes and radio buttons.

Document AI employs deep learning models, including convolutional neural networks (CNNs) and transformer-based architectures, to comprehend the semantic context of documents[14]. These models enable the system to classify documents, extract entities, and understand relationships between different elements within a document. By combining computer vision with NLP[13], Document AI can interpret complex documents, such as contracts and invoices, with high accuracy[14]. The Document AI Workbench allows users to build custom processors tailored to specific document types and business needs. Powered by generative AI, it enables users to fine-tune models with as few as 10 documents, facilitating rapid deployment of customized solutions[15].

Despite its advanced features and robust infrastructure, it faces several limitations. One key constraint is its reliance on predefined parsers for common document types such as invoices and tax forms, which is implied in the section in Google Cloud's documentation[16]. While effective for standardized inputs, this can limit the tool's adaptability to more specialized or unconventional documents without significant customization. Although the Document AI Workbench supports low-code model training, creating high-accuracy custom models for unique business needs may still demand technical expertise. The cloud-based nature of the system also introduces potential privacy and compliance concerns for organizations with strict data residency or security policies. Furthermore, its performance may be less optimal for non-English or multilingual documents, particularly those involving diverse handwriting styles or region-specific formatting, potentially affecting global usability.

2.4.2 Amazon Textract

Amazon Textract offers a highly scalable and flexible service for document data extraction, designed to work within the AWS ecosystem. It extracts printed and handwritten text from scanned documents and provides key-value pair recognition,

table extraction, and form analysis[17]. Textract's OCR capabilities are enhanced by deep learning models that can recognize printed and handwritten text in various fonts and styles, even handling noisy or distorted text[18]. This allows for accurate extraction of information from diverse document types, including forms and tables.

Unlike traditional OCR tools, Textract is optimized for semi-structured data and supports downstream integration with AWS services such as Comprehend and Lambda. Textract is designed to integrate seamlessly with other AWS services, such as Amazon Comprehend for natural language processing and Amazon SageMaker for building custom ML models. This integration enables the development of comprehensive document processing solutions tailored to specific business needs. Its API-based design makes it well-suited for automated pipelines in finance, healthcare, and insurance industries.

However, Textract's performance is often sensitive to document layout and structure. It tends to struggle with documents that deviate from standard formatting, reducing extraction accuracy without careful pre- or post-processing. While its integration with AWS services like Comprehend and SageMaker is a strength, this reliance can increase implementation complexity and introduce a degree of vendor lock-in, potentially limiting flexibility for organizations outside the AWS ecosystem. Additionally, Textract lacks strong built-in semantic analysis, meaning it may misinterpret context or relationships in complex documents unless combined with external NLP services. These limitations can necessitate additional development work to achieve accurate, domain-specific document understanding.

2.5 Reviews on Existing LLM-based Data Extraction System

This section reviews several prominent LLM-based data extraction systems, Unstructured, LlamaIndex, Unstract, and Extracta.ai, with a focus on their architecture, technological components, and relevance to modern data processing workflows. Unlike traditional rule-based or template-driven extraction tools, LLM-based systems demonstrate the capacity to interpret context, handle ambiguous structures, and extract data from a wide range of document types without extensive customization.

2.5.1 Unstructured

Unstructured is an open-source Python library designed to convert unstructured documents, such as PDFs, HTML files, Word documents, and images, into structured formats suitable for downstream processing with Large Language Models (LLMs) and other AI applications [19]. It serves as a critical preprocessing layer in Retrieval-Augmented Generation (RAG) pipelines, enabling organizations to extract meaningful insights from complex document repositories.

The core architecture of Unstructured is built on a modular system referred to as “bricks,” which consists of partitioning, cleaning, and staging components. Partitioning bricks are responsible for dissecting raw content into logical units such as titles, paragraphs, tables, and metadata blocks. Cleaning bricks apply filters to remove noise, boilerplate text, and fragmented content that may degrade model performance. Staging bricks then prepare the refined data for specific tasks such as information extraction, summarization, or indexing. This modular pipeline ensures flexibility and adaptability to a wide range of document types and processing goals. Moreover, Unstructured uses advanced parsing strategies to preserve layout, hierarchy, and context, ensuring fidelity in the transformation from raw to structured data.

From a technological standpoint, Unstructured supports seamless integration with modern LLM stacks such as LangChain, LlamaIndex, and Retrieval-Augmented Generation (RAG) systems. It also provides connectors for ingesting data from common cloud platforms like AWS S3 and Google Cloud Storage, facilitating scalable deployment. An available API allows users to access partitioning and enrichment functions without hosting the infrastructure themselves, enhancing usability for production systems.

Unstructured, while powerful in preprocessing diverse document types, primarily focuses on transforming raw data into structured segments without applying deeper semantic understanding or reasoning. It acts more as a utility layer than a complete extraction solution. This reliance on downstream models (e.g., LLMs via LangChain or LlamaIndex) means its effectiveness is heavily dependent on integration with other tools. Additionally, the partitioning logic may not generalize well across highly non-

standard documents such as handwritten notes, noisy scans, or multilingual content, limiting its reliability in such edge cases.

2.5.2 Unstract

Unstract is a modern, low-code platform designed to enable structured data extraction from unstructured documents using large language models (LLMs). Developed to simplify the deployment of AI-powered data extraction pipelines, Unstract provides an intuitive interface for building custom workflows that automate the conversion of raw documents, such as invoices, contracts, and reports, into clean, structured outputs. The platform's primary appeal lies in its ability to leverage LLMs for semantic understanding without requiring users to write extensive code, making advanced document intelligence more accessible to non-technical users and domain experts.

One of the core innovations in Unstract is its Prompt Studio, as noted in [19], a built-in feature that allows users to design, test, and optimize prompts that guide the LLM in extracting relevant fields. This approach puts the focus on human-in-the-loop prompt engineering, enabling iterative refinement of the extraction logic based on specific business needs. Behind the scenes, Unstract utilizes pre-trained transformer-based LLMs capable of zero-shot or few-shot learning, which allows the system to perform reliably even with minimal labeled training data. This drastically reduces the development cycle typically required in traditional ML pipelines.

Unstract also offers workflow deployment capabilities, allowing users to export their extraction logic as callable APIs. This facilitates seamless integration with external systems such as CRMs, ERPs, or automation platforms like Zapier or Make. Additionally, Unstract supports document ingestion from various sources and formats (e.g., PDF, DOCX, image-based files), automatically preprocessing them for LLM consumption. This level of abstraction over typical OCR and parsing tasks allows users to focus on defining what data they need, rather than how to get it. It addresses one of the key barriers in real-world LLM adoption: operationalization. By allowing users to build AI-powered data extraction pipelines without deep ML knowledge, Unstract lowers the entry point for organizations and researchers alike [19].

While lowering the barrier to entry with its no-code and low-code environment, Unstract trades off some level of control and adaptability. Its reliance on prompt-based

logic, although powerful, can become brittle in cases of ambiguous or overlapping document fields. There is also limited transparency and control over the underlying LLM's decision-making, making debugging difficult when results are inconsistent. Furthermore, performance heavily depends on the quality of the input prompts, and users may struggle to achieve optimal results without iterative refinement, which can be time-consuming.

2.5.3 LlamaIndex

LlamaIndex (formerly known as GPT Index) is an open-source framework designed to facilitate the integration of large language models (LLMs) with external, unstructured data sources for tasks such as information retrieval, document understanding, and data extraction[20]. It plays a critical role in the development of Retrieval-Augmented Generation (RAG) systems, where the goal is to enhance the performance of LLMs by grounding their outputs in domain-specific or user-provided knowledge[21]. LlamaIndex achieves this by offering a data indexing layer that transforms raw documents into formats that are easily queryable by language models.

At its core, LlamaIndex utilizes vector-based semantic indexing, where documents are chunked into smaller passages and embedded into high-dimensional vector space using sentence transformers or other pre-trained embedding models. These vectors are then stored in vector databases such as FAISS, Pinecone, or Chroma, enabling efficient similarity search [21]. This architecture ensures that when a query is issued, the most relevant pieces of context are retrieved and passed to the LLM, enhancing the quality and accuracy of responses. This retrieval mechanism is especially vital for data extraction tasks where specific facts or values need to be extracted from long or heterogeneous documents.

In terms of data structuring, LlamaIndex supports schema-guided extraction through integration with Pydantic models. Developers can define data structures or schemas, and the LLM is instructed, via prompt templates, to extract information from documents into these structured formats [22]. This allows for precise and consistent extraction of fields such as names, dates, invoice numbers, or contract terms across multiple documents. Furthermore, LlamaIndex includes tools for chunking strategies, metadata

enrichment, and document filtering, which enhance the control and granularity of data that is passed to the LLM.

LlamaIndex, although rich in functionality for retrieval-augmented generation (RAG) and schema-based extraction, has notable complexity in setup and usage. The need for embedding models, vector stores, chunking strategies, and careful prompt engineering can be overwhelming for non-technical users. Moreover, its performance hinges on the relevance of retrieved chunks and the prompt alignment with the extraction schema. If documents are noisy, poorly chunked, or semantically dense, the retrieval component may fail to capture sufficient context for accurate extraction. It is also largely optimized for English-language documents, which presents challenges for multilingual deployments.

2.5.4 Extracta.ai

Extracta.ai is a modern, AI-powered data extraction platform designed to automatically convert unstructured content from documents and images into structured data [23]. At the core of Extracta.ai's technology is the integration of Optical Character Recognition (OCR) and Large Language Models (LLMs) to perform intelligent data parsing without the need for rigid, pre-defined templates.

The system begins with advanced OCR processing to convert images and scanned documents into machine-readable text. Unlike traditional OCR tools that stop at text recognition, Extracta.ai feeds this textual data into an LLM-backed extraction engine [23]. The LLM, trained on a wide range of document types and extraction tasks, applies natural language understanding (NLU) to interpret the context of the text. This allows it to accurately identify and extract user-specified fields (such as invoice numbers, totals, names, or dates) even when the structure or formatting varies significantly between documents.

A notable feature of Extracta.ai's architecture is its template-free data extraction mechanism. Users simply define the fields they want to extract through a user interface or API, and the LLM infers the logic required to locate and retrieve this data. This is facilitated by few-shot learning, where the model can perform extraction tasks with minimal prior examples, drastically reducing the need for training data [23]. Additionally, the system provides built-in confidence scoring to evaluate the reliability

of its predictions, and it can flag low-confidence results for human review, enabling a semi-automated human-in-the-loop workflow when needed. From a systems integration standpoint, Extracta.ai exposes its functionalities through a RESTful API, supporting automated document pipelines in enterprise systems.

Extracta.ai, though effective in template-free and rapid extraction, operates largely as a black-box system. Its proprietary nature means limited visibility into its underlying models and customization options, which may be critical for organizations with domain-specific compliance or interpretability requirements. While the platform supports a range of formats and integrates OCR with LLMs, it may not offer granular tuning for unusual document types, and the quality of extraction can degrade in low-quality scans or documents with non-standard layouts. Also, the platform's dependency on internet-based APIs raises concerns for organizations needing offline or on-premise deployments.

2.6 Concluding Remark

The literature review highlights significant advancements in the field of unstructured data extraction, particularly with the integration of Optical Character Recognition (OCR) and Large Language Models (LLMs). Traditional OCR systems have evolved to incorporate deep learning and Transformer-based architectures, achieving notable gains in accuracy and versatility. However, these systems remain limited in their capacity to extract contextual meaning, interpret complex layouts, or handle poor-quality or multilingual documents reliably. Even with the emergence of AI-enhanced tools such as Google Document AI and Amazon Textract, limitations persist in adaptability, semantic understanding, and reliance on rigid, predefined templates.

In general, a key limitation shared by all the systems is the lack of handling framework for domain-specific documents that require contextual nuance or industry expertise. LLM-integrated data extraction platforms, such as LlamaIndex, Unstructured, Unstract, and Extracta.ai, have demonstrated greater flexibility and contextual awareness. Yet, each comes with its own set of constraints. Many of these systems either function as black boxes with minimal customizability (e.g., Extracta.ai) or require substantial technical expertise to implement and fine-tune (e.g., LlamaIndex).

Moreover, challenges related to prompt engineering, multilingual support, and integration consistency, especially under the probabilistic nature of LLM outputs, remain largely unresolved. While LLMs are powerful, they are prone to hallucination and may return confidently incorrect results if not constrained appropriately. These issues limit the reliability and accessibility of existing solutions in real-world deployments, particularly in industries where interpretability, precision, and adaptability are paramount.

Despite the growing adoption of Large Language Models (LLMs) in data extraction tasks, there remains a critical gap in the current landscape: **no comprehensive framework effectively addresses the inherent limitations of LLMs while enabling their seamless integration into existing data extraction pipelines.** Most existing solutions either embed LLMs as isolated components, focusing solely on final output generation, or rely on ad hoc integrations that lack stability, transparency, and interoperability.

These approaches often fail to mitigate issues such as non-deterministic outputs, poor performance on edge-case documents, and the complexity of configuring embeddings, vector stores, and prompts. Furthermore, the absence of a standardized architecture for incorporating LLMs into end-to-end pipelines hampers consistency and scalability, making it difficult for organizations to operationalize these models within their existing infrastructure. This void underscores the need for a modular framework that can both harness the strengths of LLMs and systematically manage their weaknesses within practical, real-world data workflows.

Against this backdrop, this project proposes a novel, multimodal framework that integrates LLMs with image and document understanding in a more holistic and structured manner. By introducing a methodology that incorporates multimodal inputs including text, layout, and visual cues, the framework aims to overcome current limitations in unstructured data processing.

CHAPTER 3 SYSTEM METHODOLOGY / APPROACH

In this section, we provide an overview of the proposed solution such as system architecture diagrams, user requirements, use case diagram, use case description and activity diagrams to give a broad understanding of the project's approach. The section also covers development methodologies and timeline for project completion.

3.1 Overview of Solution

The proposed solution is centred around developing a flexible, efficient, and user-friendly system for data extraction from images by integration integrating Large Language Models (LLMs) into the process of extracting unstructured data. The solution involves a dual-functionality system that can be used both as a terminal module for direct user interaction (a standalone application) and as an intermediate module integrated into larger automated workflows by serving as API endpoint.

The system architecture supports image processing through a desktop application. Users interact with the system through a graphical interface that enables template-based extraction, where custom templates can be created, edited, and managed to define Regions of Interest (ROI), regular expressions, and field mappings. These templates act as blueprints that guide the extraction of data such as text, barcodes, and tabular content from uploaded or sourced images.

The integration of LLMs is central to enhancing the system's ability to handle extraction of various data types from images, which is a primary focus of the project. Key features of the proposed solution include an image processing pipeline that handles different image formats and ensures high-quality inputs through processing techniques. The data extraction phase utilizes advanced techniques like LLM integration, image processing, template creation and matching with cropping approaches and Sieve approach, with a focus on accurately identifying and processing data from images. A notable innovation in the solution is the "Sieve Approach", which addresses the non-deterministic nature of LLMs by validating, comparing, and refining extracted data across multiple iterations.

3.2 System Architecture Diagram

The system architecture diagram in Figure 3.2.1 illustrates the dual functionality of the project, showcasing both terminal module and intermediate module pipelines. It highlights how the system can be utilized by end-users directly through a desktop application interface, as well as how it integrates into existing automated workflows as an intermediate processing module via API.

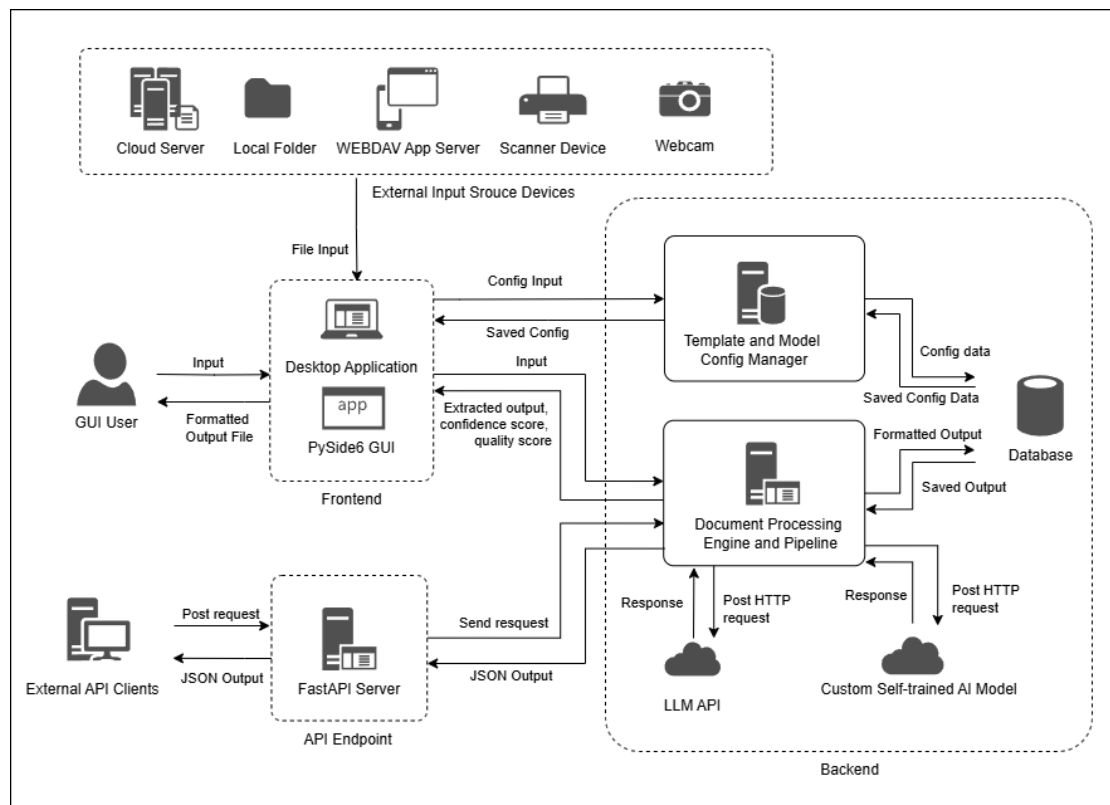


Figure 3.2.1 System Architecture Diagram

This architecture diagram represents the full data flow and modular design of the LLM Data Extraction System, structured into Frontend, Backend, Input Sources, and External Interfaces. It illustrates how documents move from various sources, through a GUI or API interface, into a processing pipeline that leverages LLMs for intelligent document understanding and finally outputs structured data.

Frontend Layer for Terminal Module as Standalone Tool

The system offers a desktop application (PySide6 GUI) for end-users, serving as the primary interface for interacting with the system. Users can upload or select image files

from various external input sources , such as cloud servers, local folders, scanner devices, webcams, or WebDAV-connected folders. The GUI allows users to select templates and models, initiate document processing, and then receive structured results like extracted text, confidence scores, and quality scores. The system also communicates with a FastAPI Server for REST API-based access, enabling external systems to interact with the extraction engine via HTTP POST requests.

Backend Layer

The backend is composed of two main components: Template and Model Config Manager and the most important part in our project – **Document Processing Engine and Pipeline**. Both components interface with a central database that stores configuration data, saved templates, model definitions, and extracted outputs.

Template and Model Config Manager is responsible for managing templates, ROI mappings, field configurations, and AI model credentials. It provides necessary config data to both the GUI and the processing engine.

On the other hand, Document Processing Engine and Pipeline is the core processor that handles the actual document transformation. It performs quality checks, invokes OCR LLM-based inference (via both custom-trained models and external LLM APIs) to perform advanced text extraction, applies regex validations via Sieve approach, and computes confidence scores. The processing engine receives configuration data and returns output to the GUI or API client in structured format (JSON, XLSX, etc.).

External Interfaces and Integration

The system is extensible and interoperable. External API clients can submit document processing requests directly to the FastAPI server, which forwards these to the backend engine and returns JSON-formatted output. Similarly, external LLM services (or local fine-tuned models) are used for document understanding via HTTP requests.

This architecture effectively ensures clear separation of concerns: the GUI handles interaction, the backend performs computation, and the data is stored in an accessible, persistent format. This setup supports both real-time GUI usage and remote automation via API, making it suitable for enterprise-level intelligent document processing.

3.3 User Requirements

The user requirements define what the end-users expect from the system in terms of functionality, performance, and usability. These requirements guide the development process to ensure that the final product aligns with user needs and enhances the workflows. The requirements are divided into functional requirements and non-functional requirements.

3.3.1 Functional Requirements (FR)

FR1: Template Management

- **FR1.1:** The system shall allow users to create new extraction templates with ROIs and regex. The system shall provide tools for users to define specific areas of an image for data extraction, including the ability to specify data types and apply constraints (e.g., limiting postcodes to 5 digits).
- **FR1.2:** The system shall allow users to view, edit, and delete existing templates.
- **FR1.3:** The system shall allow users to export templates to a .zip file (containing .json and .png).
- **FR1.4:** The system shall allow users to import templates from a .zip file.
- **FR1.5:** The system shall display templates in a dropdown list on the Data Extraction screen.
- The system shall support sorting existing templates by name, number of ROIs.

FR2: AI Model Management

- **FR2.1:** The system shall allow users to add new AI models, including public and custom models.
- **FR2.2:** The system shall allow users to view, edit, delete, and copy AI model configurations.
- **FR2.3:** The system shall enable users to set an AI model as default.
- **FR2.4:** The system shall support searching and sorting AI models by name and visibility (public/private).
- **FR2.5:** The system shall display available AI models in a dropdown list on the Data Extraction screen.

FR3: Input Source Management

- **FR3.1:** The system shall allow users to add and configure various image sources (e.g., scanner, webcam, local folder, Google Drive).
- **FR3.2:** The system shall allow users to test connection to image sources before saving.
- **FR3.3:** The system shall allow users to delete image sources.
- **FR3.4:** Configured image sources shall be visible and selectable in the Data Extraction screen.

FR4: Data Extraction

- **FR4.1:** The system shall allow users to upload sample images via a GUI.
- **FR4.2:** The system shall allow users to choose image source, model to be used, template and type of output file before conducting data extraction.
- **FR4.3:** The system shall enable users to define the format of the extracted data (e.g., JSON, plain text, CSV) and select the file type for download.
- The system shall extract barcode data using barcode scanning techniques.
- **FR4.4:** The system shall allow users to extract data from a single image file.
- **FR4.5:** The system shall allow users to batch process multiple image files.
- **FR4.6:** The system shall perform image format conversion to standardized format (e.g., PNG).
- **FR4.7:** The system shall conduct image quality checks before processing.
- **FR4.8:** The system shall provide clear error messages, such as notifying users when an image's quality does not meet the threshold for processing.
- **FR4.9:** The system shall perform template detection and utilize saved templates to instruct how to process future images, including determining what to crop, scan barcodes and produce dynamic prompt.
- **FR4.10:** The system shall process cropped areas with an LLM integration to extract relevant text data and return the output in JSON format.
- **FR4.11:** The system shall perform confidence scoring.
- **FR4.12:** The system shall display extraction results in a data grid, downloadable file, or return API response to external API clients.
- **FR4.13:** The system shall format extracted data according to user specifications.

- **FR4.14:** The system shall provide the final formatted data for download through the desktop application interface if the module is terminal.
- **FR4.15:** The system shall support resuming interrupted jobs, including auto-retry and manual resume.

FR5: Output Document Management

- **FR5.1:** The system shall provide a view of past output documents, including file names, types, and creation dates.
- **FR5.2:** The system shall allow users to search, sort, download, and delete output files.

FR6: API Endpoint Serving

- **FR6.1:** The system shall provide a local API endpoint for external access to the extraction service.
- **FR6.2:** The system shall display example requests/responses and local IP address for API integration.
- **FR6.3:** The system shall allow users to start the server to serve the API endpoint.
- **FR6.3:** The system shall display server status and allow user to disconnect it at any time.

3.3.2 NON-FUNCTIONAL REQUIREMENTS (NFR)

NFR1: Usability

- **NFR1.1:** The system shall have an intuitive GUI with accessible controls (buttons, dropdowns, tabs).
- **NFR1.2:** The system shall provide tooltips or help messages where appropriate.
- **NFR1.3:** The system shall support integration with existing enterprise software via APIs, ensuring that extracted data can be seamlessly transferred to other platforms for further processing or storage.

NFR2: Performance

- **NFR2.1:** The system shall process a single image extraction job in under 5 seconds, under normal conditions.
- **NFR2.2:** The batch processing feature shall display progress percentage and estimated completion time.
- **NFR2.3:** The system shall handle batch processing without significant performance degradation, maintaining processing efficiency for large datasets.

NFR3: Reliability

- **NFR3.1:** The system shall retry interrupted jobs up to 3 times automatically.
- **NFR3.2:** If a job fails, users shall be able to manually resume from where it left off.
- **NFR3.3:** The system shall reliably save and apply templates for future use without data loss or corruption.
- **NFR3.4:** The system shall ensure consistent data extraction results by accurately applying user-defined templates and processing criteria.

NFR4: Portability

- **NFR4.1:** The system shall be deployable on Windows and Linux machines with minimal configuration.

NFR5: Compatibility

- **NFR6.1:** The system shall support integration with cloud sources such as Google Drive, Dropbox, OneDrive.

NFR6: Maintainability

- **NFR7.1:** The system shall separate configuration files (models, templates) in a modular format (.json, .png, .zip) for easy management and updates.
- **NFR7.2:** The system shall be designed for easy updates and maintenance, allowing for quick implementation of new data types, extraction methods, or system enhancements.

3.4 Use Case Diagram and Use Case Description

This section outlines the interactions between the GUI user, API client, LLM-based data extraction system.

3.4.1 Use Case Diagram

This diagram in Figure 3.4.1.1 illustrates the different ways the actors, including user, input source or scanner, external API devices can interact with the LLM Data Extraction System. It outlines the functionalities (use cases) that the system provides and how these actors utilize them.

Actors:

- **User (GUI):** This actor interacts with the graphical user interface to perform most of the functions.
- **Input Source/ Scanner:** This represents a hardware device (like a scanner) or an external system that provides image data to the system.
- **External API Devices (Application):** This represents external devices or software that interacts with the system programmatically through the API endpoint in the system.

The diagram illustrates six main generalized use cases: UC01 – Manage Template, UC02 – Manage AI Model, UC03 – Manage Input Source, UC04 – Process Image and Extract Data, UC05 – View Past Output Documents and Results, and UC06 – Serve API Endpoint. Among these, the system primarily centers on **UC04 – Process Image and Extract Data**, as it integrates the proposed processing pipeline, which will be discussed in detail in Chapter 4.

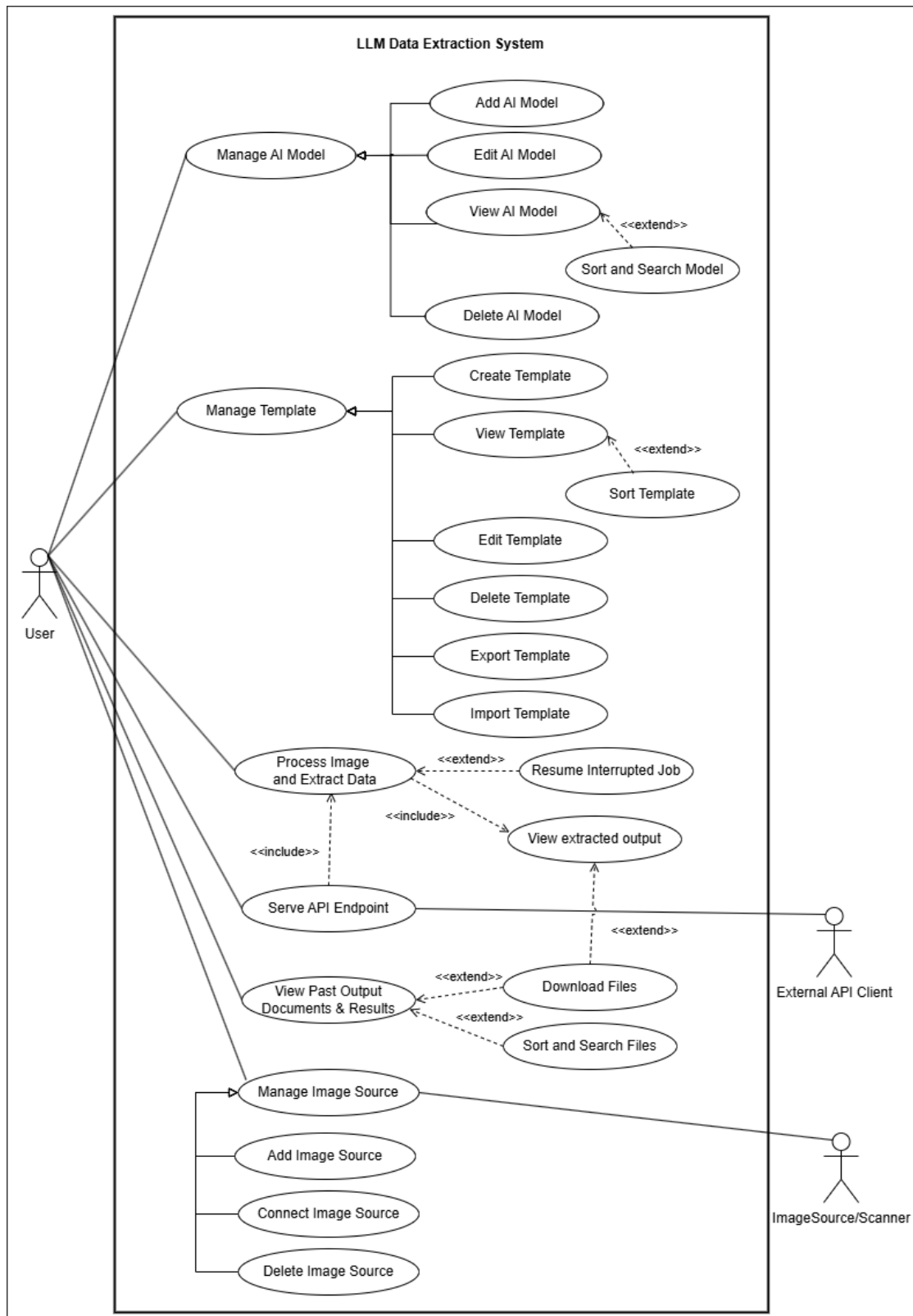


Figure 3.4.1.1 Use Case Diagram

3.4.2 Key Use Case Description

This section provides detailed descriptions of the six primary use cases, which serve as the foundation for use case testing discussed in Chapter 6.3. Use case descriptions help define system functionality from the user's perspective, ensuring that development aligns with user needs and expectations. Among these, **UC04 – Process Image and Extract Data** stands out for its unique functionality, representing the project's most innovative and impactful feature.

UC01 Manage Template

Field	Value	
ID	UC01	
Purpose	To create, edit, delete, view, export, and import extraction templates for data extraction.	
Primary Actor(s)	User	
Trigger	User selects Template Manager from the navigation sidebar.	
Sub-Functions	<ul style="list-style-type: none"> • Create Template • View Template • Edit Template • Delete Template • Export Template • Import Template 	
Pre-conditions	<ul style="list-style-type: none"> • System is <i>On</i> and <i>Idle</i>. 	
Scenario Name	Step	Action
Main Flow	1	User launches Template Manager screen.
	2	User clicks one of the buttons on Template Hub.
Alternate Flow – Create Template	2.1.1	User clicks (+) button.
	2.1.2	User fills in template details, uploads image, creates ROIs, saves template.
	2.1.3	User uploads template image
	2.1.4	User creates ROIs on the template with fields and regex.
	2.1.5	User saves template.
Alternate Flow –Edit Template	2.2.1	User clicks edit button on a template.
	2.2.2	User modifies uploaded image and fields.
	2.2.3	User saves template.
Alternate Flow – Delete Template	2.3.1	User clicks delete button on a template.
	2.3.2	System prompts confirmation.

	2.3.3	If confirmed, system deletes the template.
Alternate Flow – View Template	2.4.1	User clicks view button on a template.
	2.4.2	System displays template details with sorting.
Alternate Flow – Export Template	2.5.1	User clicks export button.
	2.5.2	System prompts for the location to download the template.
	2.5.3	User chooses file location and save exported template.
Alternate Flow – Import Template	2.6.1	User clicks import button.
	2.6.2	User selects template and image files to upload.
	2.6.3	System imports and displays the new template.
Post-conditions	Template is added, edited, deleted, viewed, exported, or imported successfully and appears in the Template Hub.	

Table 3.4.2.1 UC01 Manage Template Use Case Description

UC02 Manage AI Model

Field	Value	
ID	UC02	
Purpose	To create, edit, delete AI models for document processing.	
Primary Actor(s)	User	
Trigger	User selects AI Model Config from the navigation sidebar.	
Sub-Functions	<ul style="list-style-type: none"> • Add AI Model • Edit AI Model • Delete AI Model 	
Pre-conditions	<ul style="list-style-type: none"> • System is <i>On</i> and <i>Idle</i>. 	
Scenario Name	Step	Action
Main Flow	1	User launches AI Model Config screen.
	2	User clicks one of the buttons on AI Model Config.
Alternate Flow –Add AI Mode	2.1.1	User clicks (+) button.
	2.1.2	User chooses public/custom model
	2.1.3	User fills in API key and AI model details.
	2.1.4	User toggles default option.
	2.1.5	User saves AI model.
Alternate Flow – View AI Model	2.2.1	User clicks view button on a AI model.
	2.2.3	User can sort or search based on model name.

	2.2.3	System displays template details including API key with sorting and searching results.
Alternate Flow –<i>Edit AI Model</i>	2.3.1	User clicks edit button on an AI model.
	2.3.2	User modifies existing AI details.
	2.3.3	User saves template.
	2.3.4	User saves AI model details.
Alternate Flow –<i>Delete Template</i>	2.4.1	User clicks delete button on an AI model.
	2.4.2	System prompts confirmation.
	2.4.3	If confirmed, system deletes the AI model.
Post-conditions	AI Model is created, edited, deleted, viewed, or copied and shown in the Model and API Manager list.	

Table 3.4.2.2 UC02 Manage AI Model Use Case Description

UC03 Manage Input Source

Field	Value	
ID	UC03	
Purpose	To add, connect, and delete input sources for input file acquisition.	
Primary Actor(s)	User, Input Source/Scanner	
Trigger	User selects Input Source Setup from the navigation sidebar.	
Sub-Functions	<ul style="list-style-type: none"> • Add Input Source • Connect Input Source • Delete Input Source 	
Pre-conditions	<ul style="list-style-type: none"> • System is <i>On</i> and <i>Idle</i>. 	
Scenario Name	Step	Action
Main Flow	1	User launches Input Source Setup screen.
	2	User clicks one of the buttons on AI Model Config.
Alternate Flow –<i>Add Input Source</i>	2.1.1	User clicks (+) button.
	2.1.2	User selects source (scanner, webcam, cloud storage).
	2.1.3	User fills in input source details.
	2.1.4	User test connection.
	2.1.5	User saves input source.
Alternate Flow –<i>Connect Input source</i>	2.2.1	User clicks connect button on a template.
	2.2.3	System connects to the input source.
	2.2.3	System displays input source connection status.
	2.3.1	User clicks delete button on an input source.

Alternate Flow – Delete Input Source	2.3.2	System prompts confirmation.
	2.3.3	If confirmed, system deletes the input source.
Post-conditions	Input source is added, connected, or deleted, and displayed in the Input Source/Scanner Setup screen.	

Table 3.4.2.3 UC03 Manage Input Source Use Case Description

UC04 Process Image and Extract Data

Field	Value	
ID	UC04	
Purpose	To allow a user extract data from uploaded/connected images and receive structured results.	
Primary Actor(s)	User, External API Client	
Trigger	User navigates to Data Extraction screen clicks (+) button in the GUI (Image Processing & Extraction Hub screen) or API Client sends POST /process.	
Pre-conditions	<ul style="list-style-type: none"> System is <i>On</i> and <i>Idle</i>. At least one template exists <i>or</i> template-less mode is enabled. At least one API model exists. 	
Scenario Name	Step	Action
Main Flow	1	User launches Data Extraction screen
	2	User uploads image(s) or select input source dropdown.
	3	User selects AI model, template, and output type.
	4	User clicks “Extract Data” button.
	5	System converts input files to standardized format - PNG (if needed).
	6	System performs Image Quality Check.
	7	If pass, the system detects template (barcode/ML).
	8	System preprocess, analyses the image and extracts structured data (ROI extraction, text recognition, validation, Sieve Approach, Challenger, Referee).
	9	System calculates confidence.
	10	System returns data grid, summary table (GUI), output files to be downloaded or JSON (API).
Alternate Flow – Template not found	1.1.1	System prompts user to create template by displaying “Template Not Found” message.
	1.1.2	System displays button to navigate to template creation screen.

Alternate Flow –API Model not found	1.2.1	System prompts user to create template by displaying “API Model Not Found” message.
	1.2.2	System Display button to navigate to API model manager screen.
Alternate Flow –Low Image Quality	6.1	System performs image rectification process.
	6.2	Repeat Step 4
Alternate Flow –Low Image Quality Persists	6.2.1	System rejects the image and skips data extraction.
Alternate Flow –No Template Matched	7.1	System uses fall back features (use default template or skip processing and return null or stop processing and notify user) to process the image
Alternate Flow – Job Interrupted	8.1	System automatically resumes processing for 3 tries.
	8.2	System prompts the user to manually resume processing.
Post-conditions	<ul style="list-style-type: none"> • Process entry visible in Image Processing and Extraction Manager. • Data extracted and available for download. • Output files can be accessed in Output Documents Screen. 	

Table 3.4.2.4 UC04 Process Image and Extract Data Use Case Description

UC05 Access Past Output Documents & Results

Field	Value	
ID	UC05	
Purpose	To view, search, sort, download, and delete past output documents and results.	
Primary Actor(s)	User	
Trigger	User selects Output Documents from the navigation sidebar.	
Pre-conditions	<ul style="list-style-type: none"> • System is <i>On</i> and <i>Idle</i>. • Data extraction must have been performed previously. 	
Scenario Name	Step	Action
Main Flow	1	User launches Output Documents screen
	2	User views file names, types, creation times.
	3	User searches or sorts files.
	4	User downloads or deletes files as needed.
Alternate Flow –File not found	4.1	System prompt error message.

Post-conditions	<ul style="list-style-type: none"> Selected documents are downloaded or deleted, and the list updates accordingly.
------------------------	---

Table 3.4.2.5 UC05 Access Past Output Documents Use Case Description

UC06 Serve API Endpoint

Field	Value	
ID	UC06	
Purpose	To serve data extraction functionality via an API endpoint for external clients.	
Primary Actor(s)	User, External API Client	
Trigger	User selects Serve API Endpoint from the navigation sidebar.	
Pre-conditions	<ul style="list-style-type: none"> System is <i>On</i> and <i>Idle</i>. 	
Scenario Name	Step	Action
Main Flow	1	User launches Serve API Endpoint screen
	2	User copies example request/response formats and local IP address for establishing connection purpose.
	3	User clicks Start Server to begin serving API.
	4	System starts serving API endpoint.
	5	System display API endpoint serving status.
Alternate Flow – Server failure	3.1	System shows error with possible troubleshooting steps.
Post-conditions	<ul style="list-style-type: none"> Server is running. External API clients can interact with the endpoint. 	

Table 3.4.2.6 UC06 Serve API Endpoint Use Case Description

3.5 Activity Diagram

This section presents six activity diagrams representing key flow of use cases of the project. Among these, the most significant is **UC04 – Process Image and Extract Data Activity Diagram**, which embodies the core innovation and adds the greatest value to the project.

UC01 – Manage Template

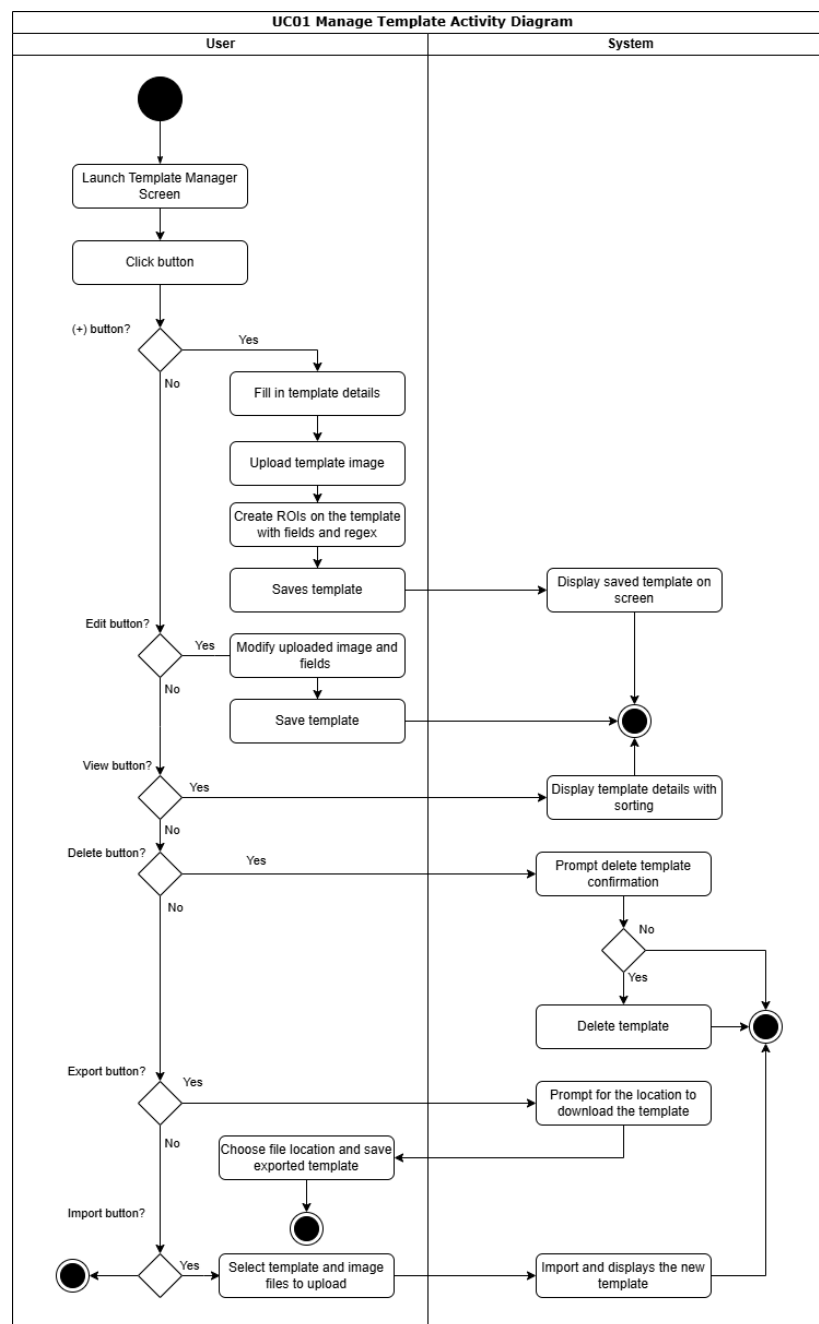


Figure 3.5.1 UC01 Manage Template Activity Diagram

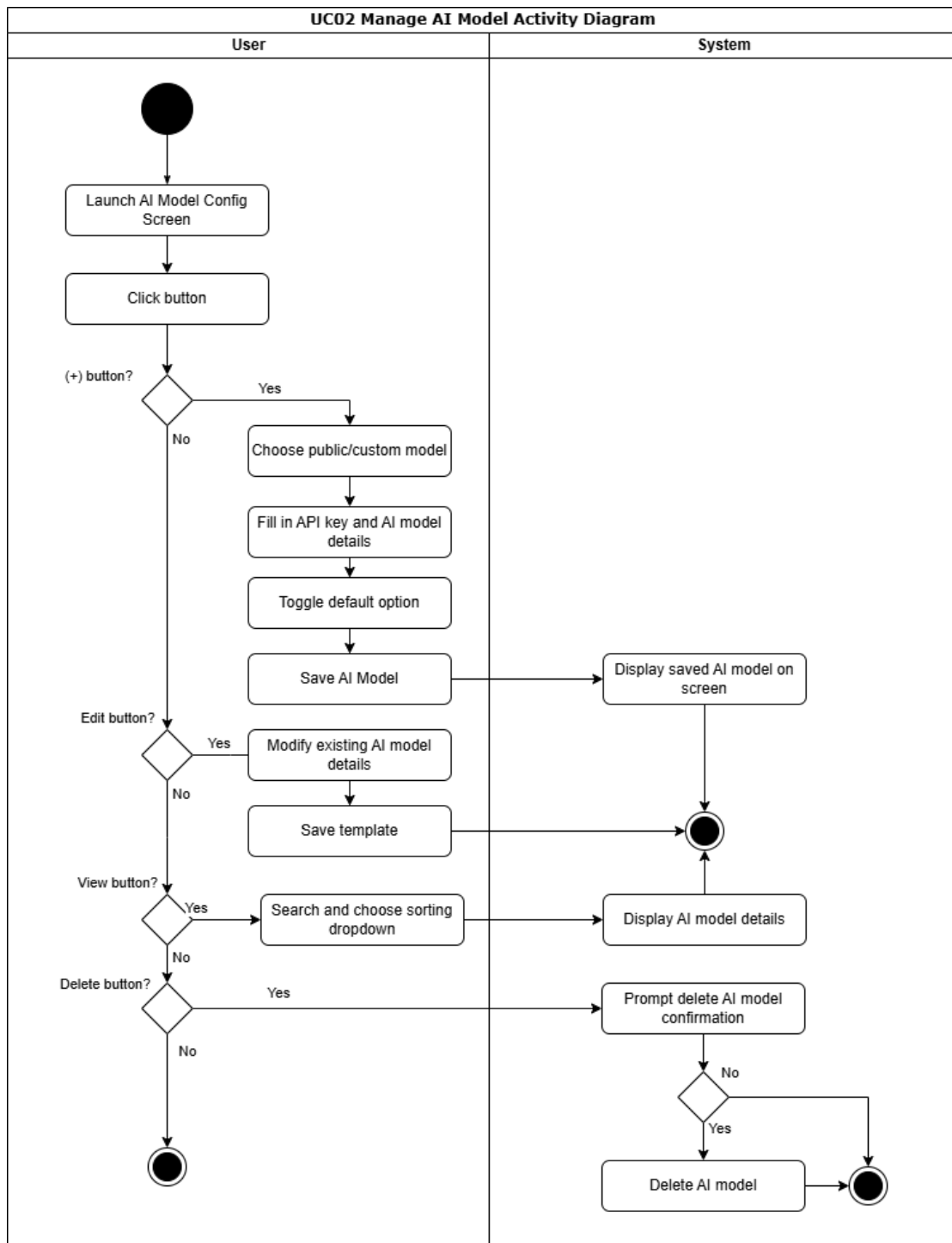
UC02 – Manage AI Model

Figure 3.5.2 UC02 Manage AI Model Activity Diagram

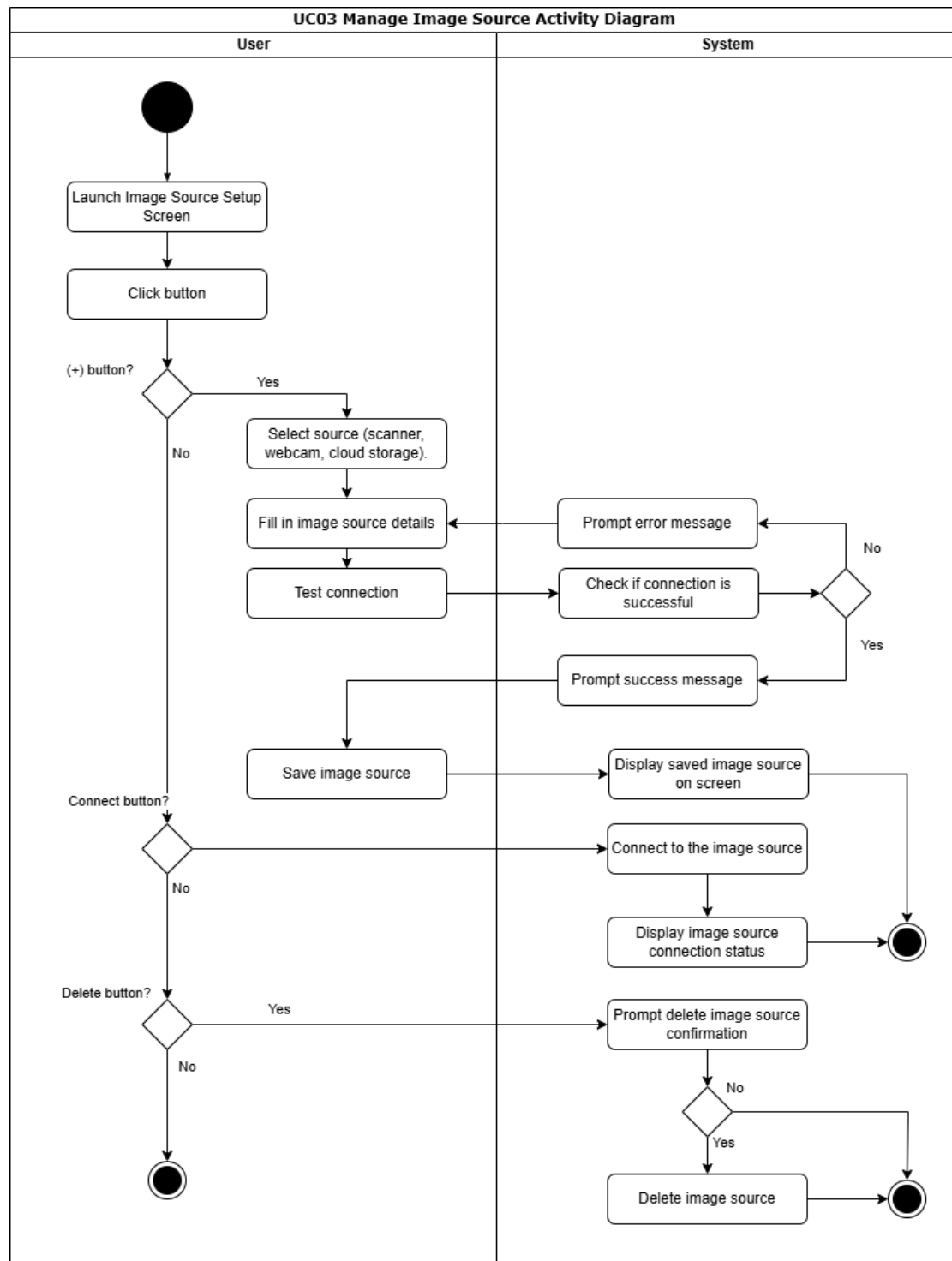
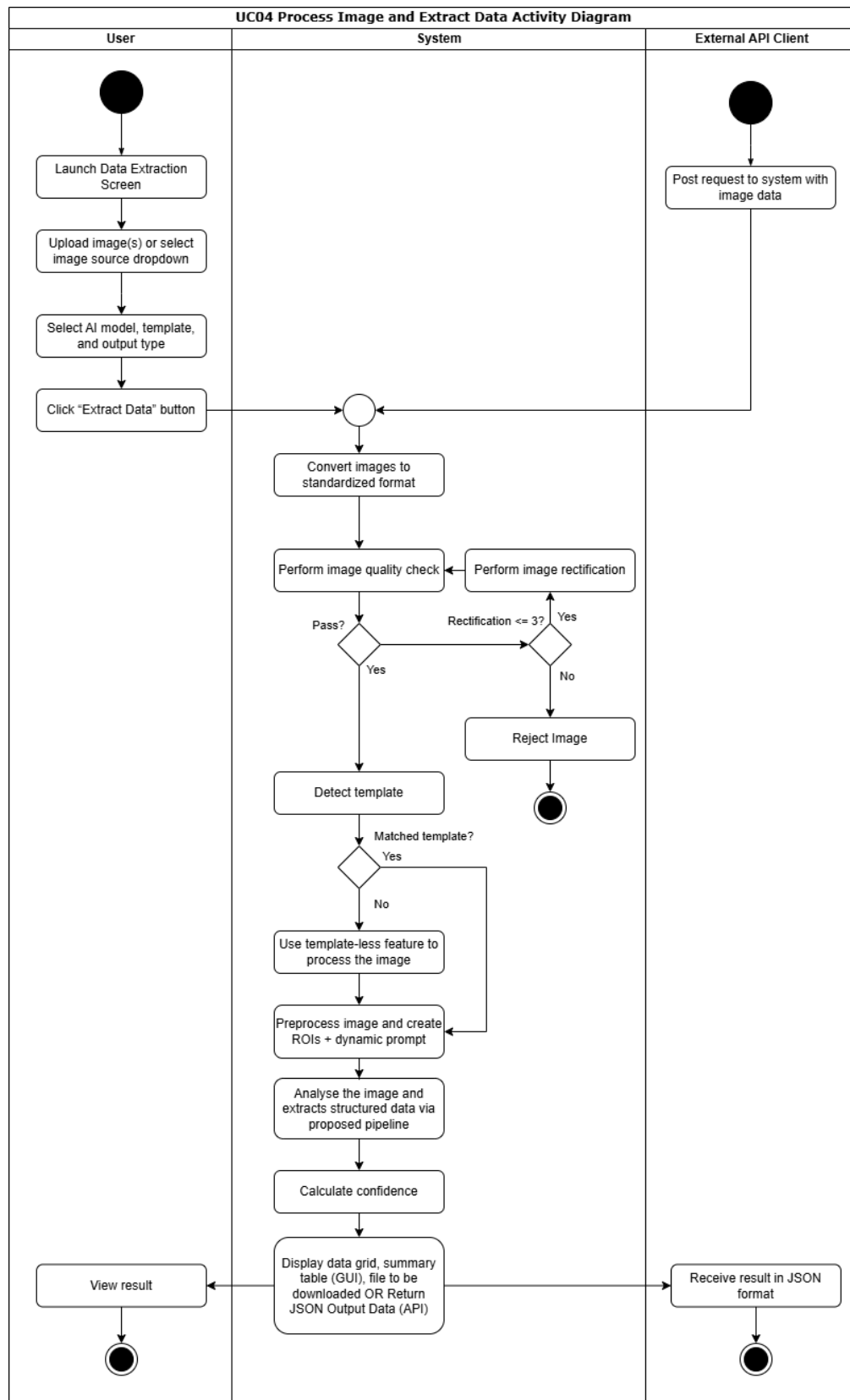
UC03 – Manage Input Source

Figure 3.5.3 UC03 Manage Input Source Activity Diagram

UC04 - Process Image and Extract Data (Main Use Case)*Figure 3.5.4 UC04 Process Image and Extract Data Activity Diagram*

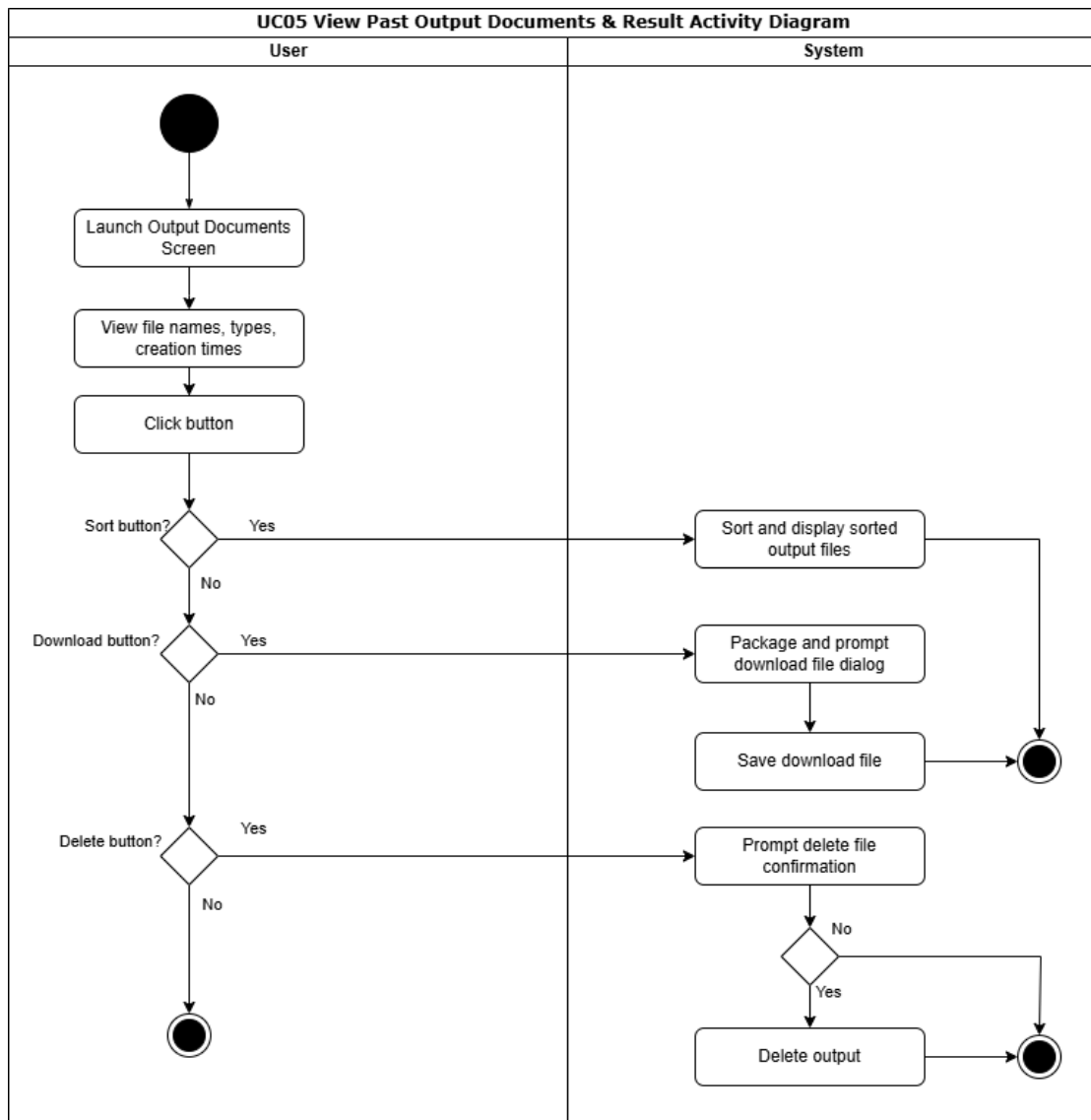
UC05 – View Past Output Documents & Result

Figure 3.5.5 UC05 View Past Output Documents & Results Activity Diagram

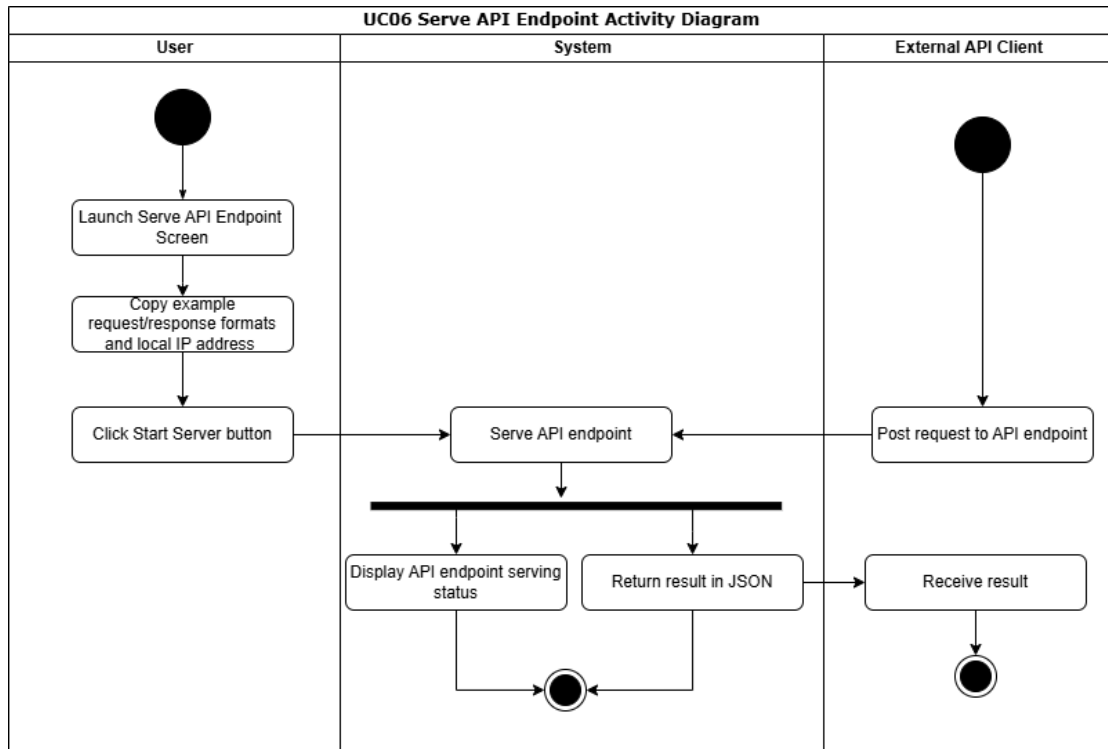
UC06 – Serve API Endpoint

Figure 3.5.6 UC06 Serve API Endpoint Activity Diagram

3.6 General Development Methodologies and General Work Procedures

Figure 3.6.1 shows a rapid application development (RAD) model that is used in this project.

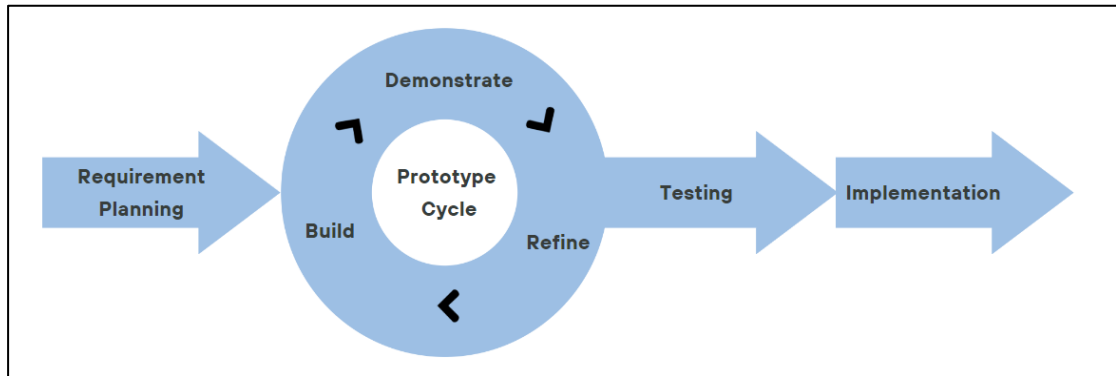


Figure 3.6.1 Rapid Application Development (RAD) Model

Rapid Application Development (RAD) methodology, with its emphasis on quick and iterative development cycles are used in this project. There are few stages in this methodology, including requirement planning, user design and prototyping, construction (development) and cutover (testing and implementation).

First, in requirement planning, requirements such as the goals, scope, and user requirements are clearly defined. For this project, this phase would involve gathering all necessary information about the specific needs of integrating LLMs for unstructured data extraction, including understanding the challenges with current methods, desired outcomes, and user requirements for the final system. The outcome of this phase will be a clear set of requirements that guide the subsequent development stages.

Moving on comes a prototype cycle. In this stage, the initial prototypes of the system are developed based on the requirements gathered. For this project, the prototypes would include early versions of the LLM integration framework, the data extraction processes, and the user interface. A basic version of the framework and interface is created, focusing on the core functionalities such as LLM integration, data extraction accuracy, and efficiency optimization. The developed prototype is demonstrated to stakeholders and end-users to gather feedback. Based on the feedback received during the demonstration, the prototype is refined to address any issues or incorporate

suggested improvements. This iterative process ensures the final product is aligned with user needs.

Once the prototype has gone through sufficient iterations and refinements, it moves into the testing phase. This is where the system is thoroughly tested to ensure it meets all functional and non-functional requirements, such as performance, accuracy, and user-friendliness. Extensive testing of the framework will be conducted across different datasets, scenarios, and edge cases to ensure reliability. Any bugs or issues identified will be addressed before moving to the final implementation phase.

After successful testing, the final system is implemented in the real-world environment. The final, refined framework will be deployed into the target environment, such as a logistics operation.

The RAD model’s iterative approach allows this project to rapidly develop and refine a framework for LLM integration in data extraction, ensuring that the final product is well-aligned with user needs and can be implemented efficiently. By continuously incorporating user feedback and refining the system through multiple prototype cycles, the project minimizes risks and maximizes the chances of delivering a high-quality, user-friendly solution that meets all defined requirements.

3.7 Timeline and Gantt Chart

Figure 3.7.1 below illustrates a Gantt Chart for this project in Project 1 and Project II.

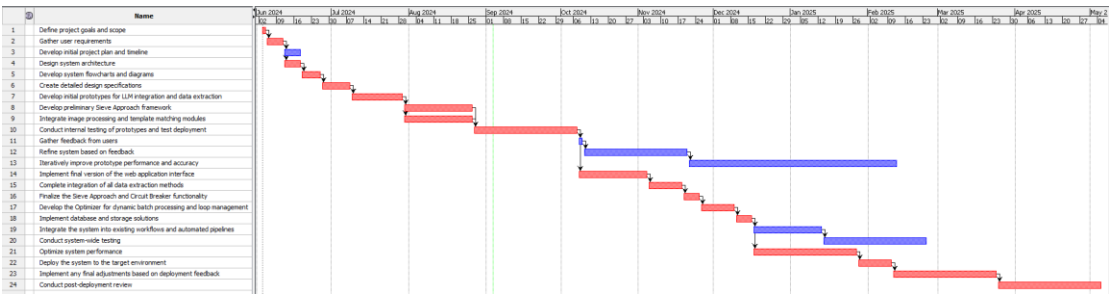


Figure 3.7.1 Gantt Chart

The Gantt Chart provides an estimated timeline for the project’s deliverables and milestones, spanning a year. This project is to be finished by May 2025.

CHAPTER 4 SYSTEM DESIGN

In this chapter, we present the system design via system flowchart, system block diagram and details about our proposed application integration pipeline for LLM.

4.1 System Flowchart

The system flowchart in the Figure 4.1.1 outlines a high-level process involved in the project, illustrating how data is processed from initial input to final output, whether used as a terminal or intermediate module within a larger system.

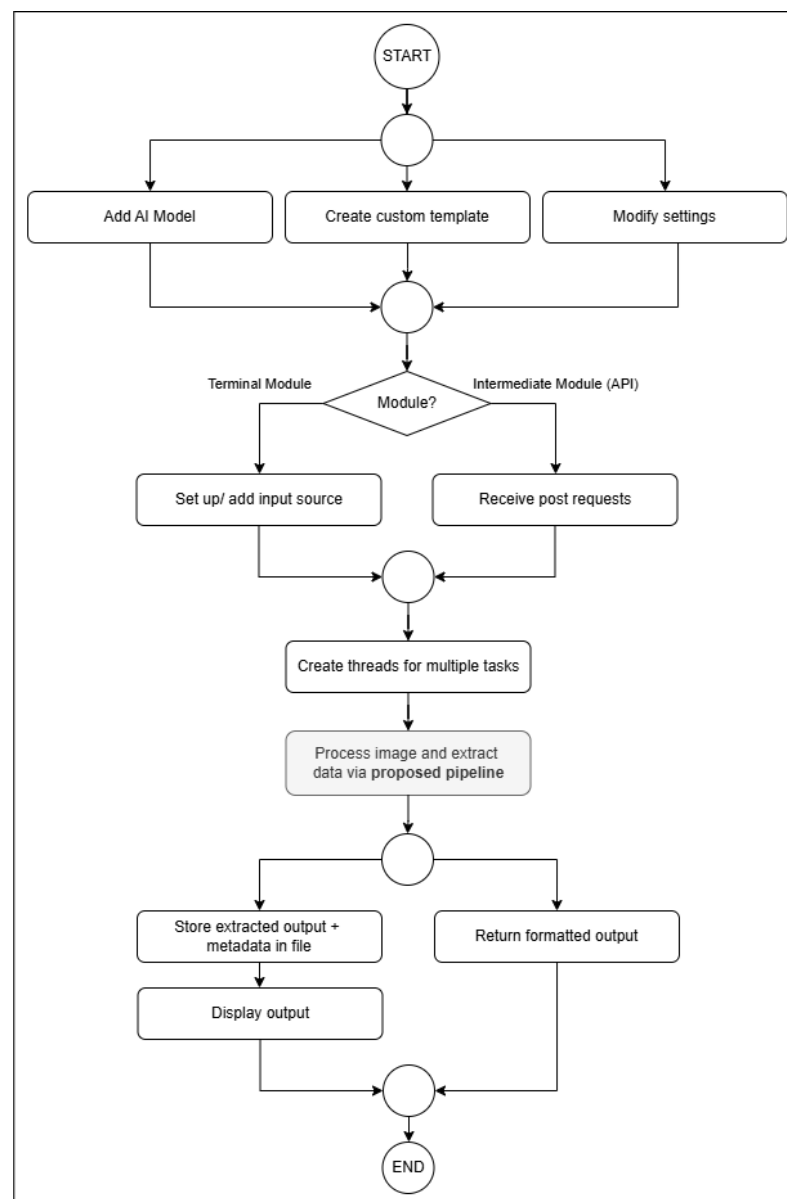


Figure 4.1.1 System Flowchart

Figure 4.1.1 illustrates the high-level operational flow of the proposed LLM-enhanced image data extraction system. The process begins with three core configuration steps: adding AI models, creating custom templates, and modifying system settings. These preparatory actions are essential for tailoring the system to specific use cases. Adding an AI model allows the integration of pre-trained language models or custom APIs for advanced text recognition. Custom templates define extraction rules by specifying regions of interest (ROI), expected field types, and regex patterns. Modifying system settings enables users to configure advanced features such as template matching strategies (e.g., barcode-based, content-based or manual selection), select default templates or models, and determine pipeline behaviour, including whether to activate the LLM Challenger and Referee mechanism.

Following configuration, the system determines the mode of operation, either as a Terminal Module (for standalone desktop use) or an Intermediate Module (serving as an API endpoint). In terminal mode, users proceed by setting up or connecting to input sources such as scanners, webcams, local folders, or cloud storage platforms. In API mode, the system instead begins processing upon receiving external POST requests from integrated systems. Subsequently, the system spawns multiple threads to handle parallel processing tasks. This multithreaded design supports efficient execution, particularly during batch processing or when performing complex pipelines involving OCR, validation, and scoring concurrently.

The core functionality takes place in the data extraction pipeline, which includes several sequential stages such as image pre-processing, format normalization, quality verification, template detection, and data extraction via LLMs. Advanced techniques such as the Sieve Approach, and the LLM Challenger and Referee mechanism are used during post-processing to validate, compare, and refine extracted outputs, addressing the non-deterministic nature of language models.

Finally, the system handles the output based on the selected module. In terminal mode, the extracted data and its associated metadata (e.g., confidence levels, timestamps, template IDs) are stored in local files and then displayed in a user-accessible output screen. In API mode, the system returns the formatted response, ready for integration into downstream systems.

4.2 LLM Application Integration Framework for Unstructured Data Extraction

This section presents the data extraction framework and provides a detailed overview of the application pipeline designed for integrating LLMs. The framework and pipeline constitute a central contribution of this project, enabling effective and scalable processing of unstructured data through LLM integration.

4.2.1 Data Extraction Framework

The data extraction framework is shown in Figure 4.2.1.1. below, which represents a structured approach to extract image data in the project, leveraging various techniques and components, including a LLM, for comprehensive data extraction. Besides, the Sieve methodology that is illustrated in the diagram below will be mainly introduced in the Chapter 4.4.3. Data Post-Processing Phase, as it is the main contribution of this work.

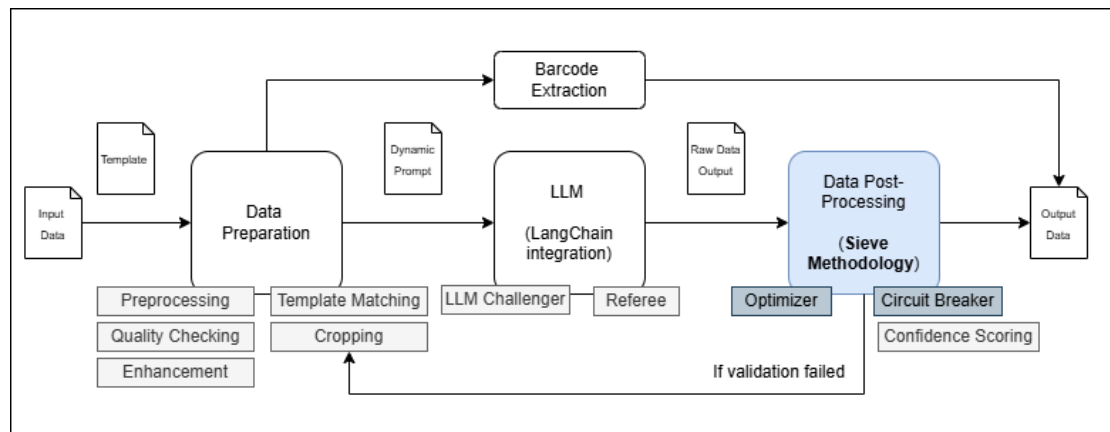


Figure 4.2.1.1 Data Extraction Framework

The process begins with Data Preparation, where input images undergo several steps, including preprocessing, quality checking, enhancement, and template matching. This stage ensures that the input data is clean and structured enough for downstream analysis. The selected template guides the system in cropping specific regions of interest from the document, which are essential for focused data extraction.

Following preparation, the cropped data is sent to the LLM component, which integrates with LangChain for advanced language model capabilities. This component dynamically generates prompts based on the content and structure of the input regions and uses them to extract raw data. To ensure reliability, the LLM process includes

elements such as a “Challenger” to allow the use of 2 LLMs to process the data and compare the output, and a “Referee” to arbitrate between conflicting outputs when necessary. Concurrently, barcode information, if present, is extracted and used to guide template selection or validation.

The extracted data then enters the Data Post-Processing stage, specifically the Sieve Methodology, which plays a critical role in validating, refining, and finalizing the results. This step includes an Optimizer and a Circuit Breaker to manage errors or flag major inconsistencies. Confidence Scoring is also applied, providing a metric for assessing the reliability of the extracted information. If validation fails at any point, the system can loop back, allowing for re-processing or optimization of the data before final output.

In the end, the framework produces structured output data, either directly or after going through post-processing validation cycles. This architecture emphasizes accuracy, adaptability, and error handling, making it suitable for complex document processing tasks in real-world applications.

4.2.2 Application Pipeline Design for LLM Integration

The application pipeline that is designed for LLM integration is illustrated across the three flowchart diagrams in Figure 4.2.2.1, Figure 4.2.2.2, Figure 4.2.2.3, presents a comprehensive and modular design for the project. This pipeline serves as a critical foundation and blueprint for the proposed solution, highlighting key strategies and components involved in integrating LLMs effectively. It introduces the core concepts and methodologies that underpin the system's functionality, which will be further elaborated in the Chapter 4.3.

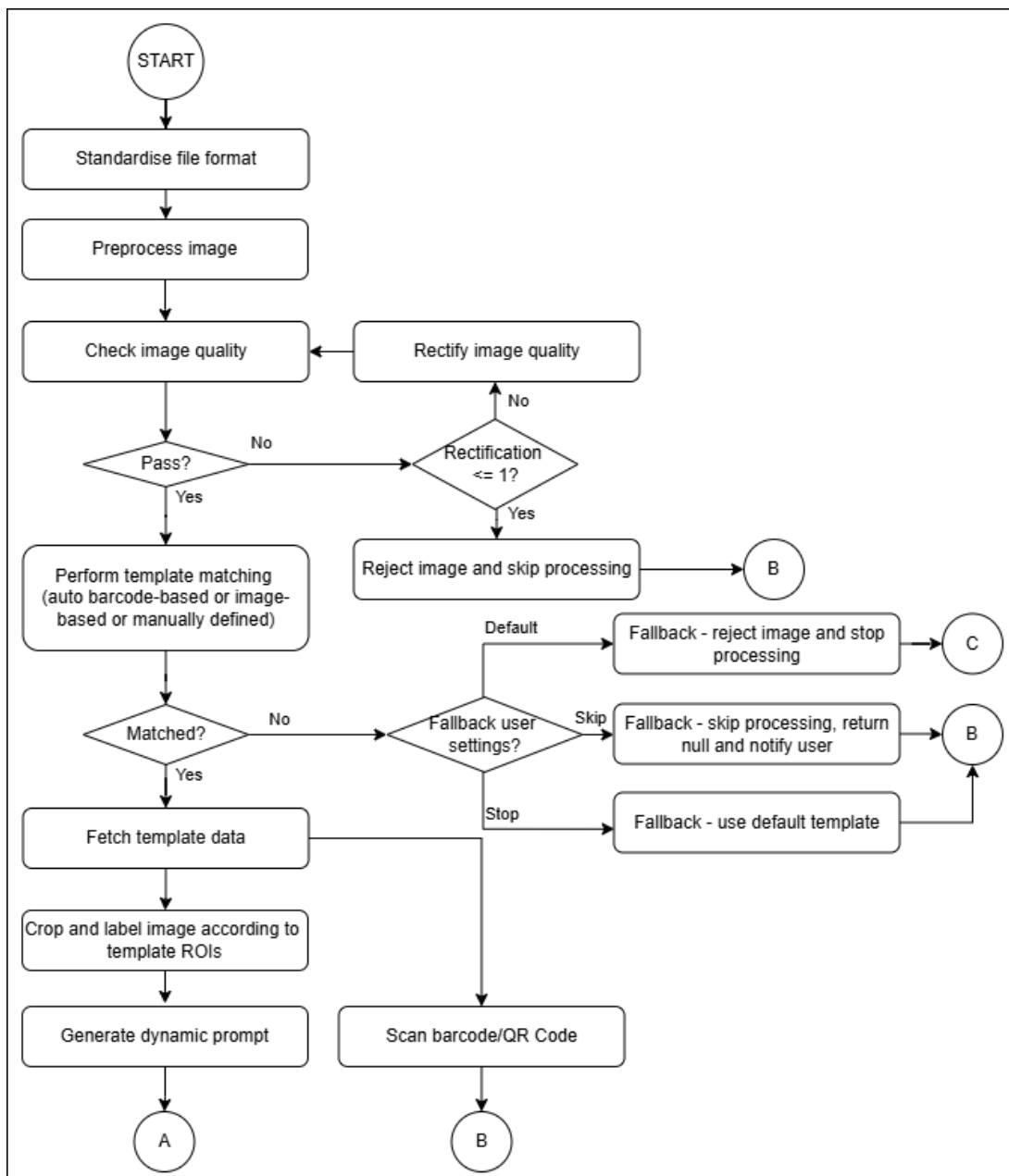


Figure 4.2.2.1 System Pipeline Design Part 1

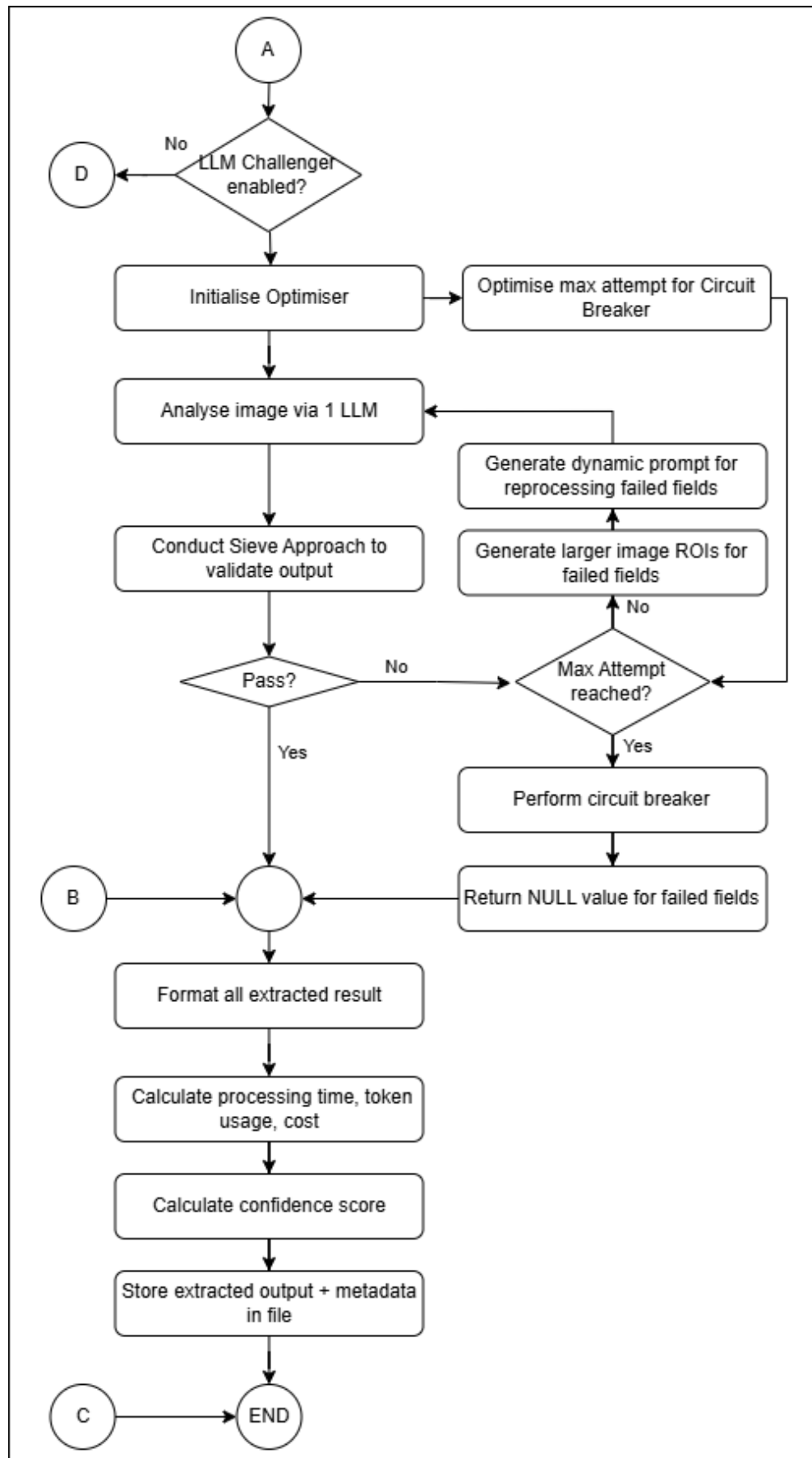


Figure 4.2.2.2 System Pipeline Design Part 2

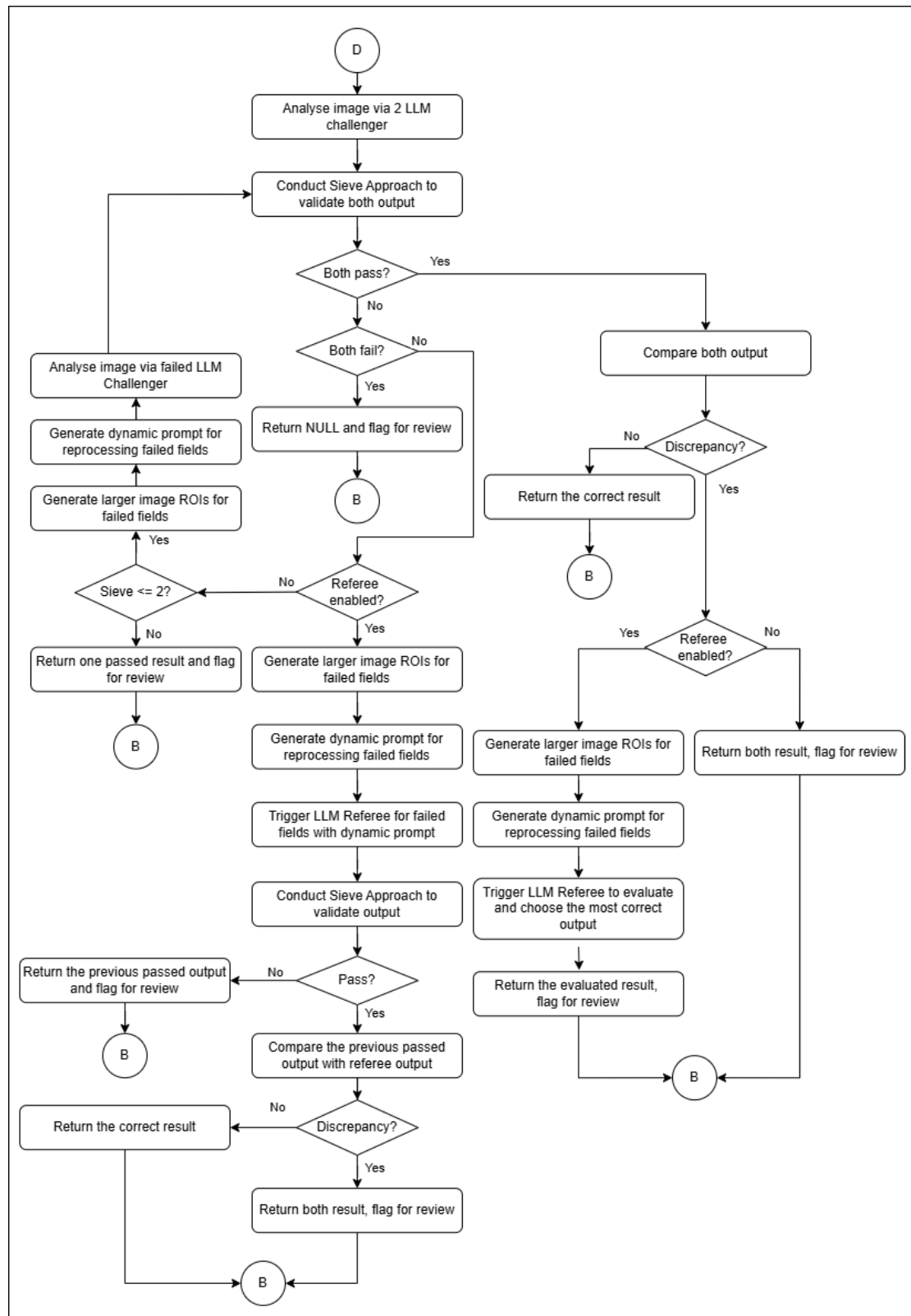


Figure 4.2.2.3 System Pipeline Design Part 3

Data Preparation and Template Matching (Pipeline Part 1)

The process begins with the preprocessing and conditioning of files, which leads to standardization of the image file format, followed by an image quality check. If the image quality fails, a rectification mechanism is triggered. Should the quality still be insufficient after rectification, the image is rejected or handled based on fallback logic defined by the user, options include using a default template, skipping processing, or halting execution.

Next, the system performs template matching, which may be automatic (barcode-based or content-based) or manual. If matching is successful, the system retrieves the relevant template data and initiates image preprocessing. This includes cropping the image according to template-defined ROIs (Regions of Interest) and generating dynamic prompts for the LLM based on the template structure and detected fields. If matching fails, fallback behavior (default template, skip, or stop) is executed based on user settings.

Sieve Methodology - Validation, LLM Challenger and Referee (Pipeline Part 2 & 3)

The second stage of the pipeline centers around LLM-based analysis and decision validation through a structured control flow. Depending on the system configuration, the image is either processed through a single LLM or handled by a dual-LLM challenger framework.

In the standard configuration, the extracted results from the LLM are routed into the **Sieve Methodology**, which performs a series of structured validations based on the document's associated template. This methodology checks for correctness, completeness, and conformance to required formats, ensuring data integrity before final output. If the validation passes, the results are formatted and outputted. However, if validation fails, the system enters a controlled retry loop. During this phase, dynamic prompts are generated, and larger or refined image regions (ROIs) are utilized to give the LLM better context for correcting the failed extractions. This process is repeated with a maximum retry threshold enforced by a **Circuit Breaker**, preventing infinite loops and gracefully terminating extraction attempts in persistent failure scenarios. Detailed mechanics of this retry process are elaborated in Section 4.4.3.

In the enhanced **Challenger Mode**, the system makes use of two separate LLMs to independently analyze the same image in parallel. Their outputs are then passed through the Sieve validation process. If both outputs pass, they are compared for consistency. If no discrepancies are found, the result is accepted and the system continues with its standard processing flow. However, if discrepancies arise between the two valid outputs, the system evaluates whether a **Referee module** is enabled. If it is, the Referee performs arbitration by reprocessing the disputed fields to determine the most accurate result. If the Referee is not enabled, both outputs are flagged for manual review.

If both LLM challengers fail validation, the system recognizes this as a critical failure and immediately flags the input as invalid, resulting in a null or error-marked output. In the case where only one challenger fails, the system attempts recovery by reprocessing the failed challenger using updated prompts and expanded image regions to improve contextual understanding. If the number of failed fields is within a threshold (e.g., ≤ 2), the corrected output is validated and returned alongside the previous passed result for manual review. If the threshold is exceeded, the system bypasses further arbitration and flags one of the passed results for review to avoid excessive computation or ambiguous decision paths. This recovery path may also invoke the Referee module if enabled, offering an additional layer of verification.

The Referee process is particularly nuanced. If the arbitration is required (either after failure or discrepancy), it triggers a refined re-extraction process. The Referee is given dynamic prompts and broader context via expanded image regions. It performs an fresh independent extraction, which is validated using the Sieve Methodology. If the Referee's result passes, it is compared to the previously passed challenger output. If both results match, the system returns the agreed-upon result. If they **do not align**, both outputs are returned and flagged for manual inspection, though the Referee's version may be provisionally favoured based on system rules or configuration. If the Referee's attempt fails validation altogether, the system reverts to the previous passed result and flags it as uncertain.

It is important to note that while the Challenger and Referee modules significantly enhance accuracy, they also **introduce increased computational overhead and latency**. Their use is most beneficial in high-stakes environments where precision outweighs performance concerns, such as legal, financial, or regulatory document

processing. Due to their resource intensity and conditional effectiveness, these modules are **recommended only when maximum accuracy is essential**. For this reason, their detailed specifications are omitted from Section 4.4's core component descriptions, as their behaviour and value are better explained in this contextual discussion.

Final Formatting, Scoring, and Storage (Pipeline Part 3 end)

Once validated data is ready, the system formats all extracted outputs into structured forms (e.g., JSON, CSV, or table format). It then calculates metadata such as processing time, token usage, and estimated cost, followed by the generation of a confidence score based on validation metrics and LLM feedback. The final output, along with metadata, is stored securely in a designated file system or database.

The pipeline concludes with a clean handoff, either to the user interface for display in a terminal module, or as a formatted API response for integration into external workflows. This modular yet fault-tolerant design ensures high reliability even in uncertain or low-quality data scenarios.

This multi-phase pipeline, with fallback options, iterative LLM validation (Sieve, Challenger, Referee), and structured output, demonstrates a production-grade approach to intelligent data extraction from images. It balances automation, accuracy, and human oversight, making it suitable for real-world document processing environments.

4.3 System Block Diagram

The system block diagram in Figure 4.3.1 presents a high-level overview of a modular architecture designed for data extraction with LLM integration. Each component within the diagram will be further detailed in Chapter 4.4, including an explanation of how they interconnect to form the complete system.

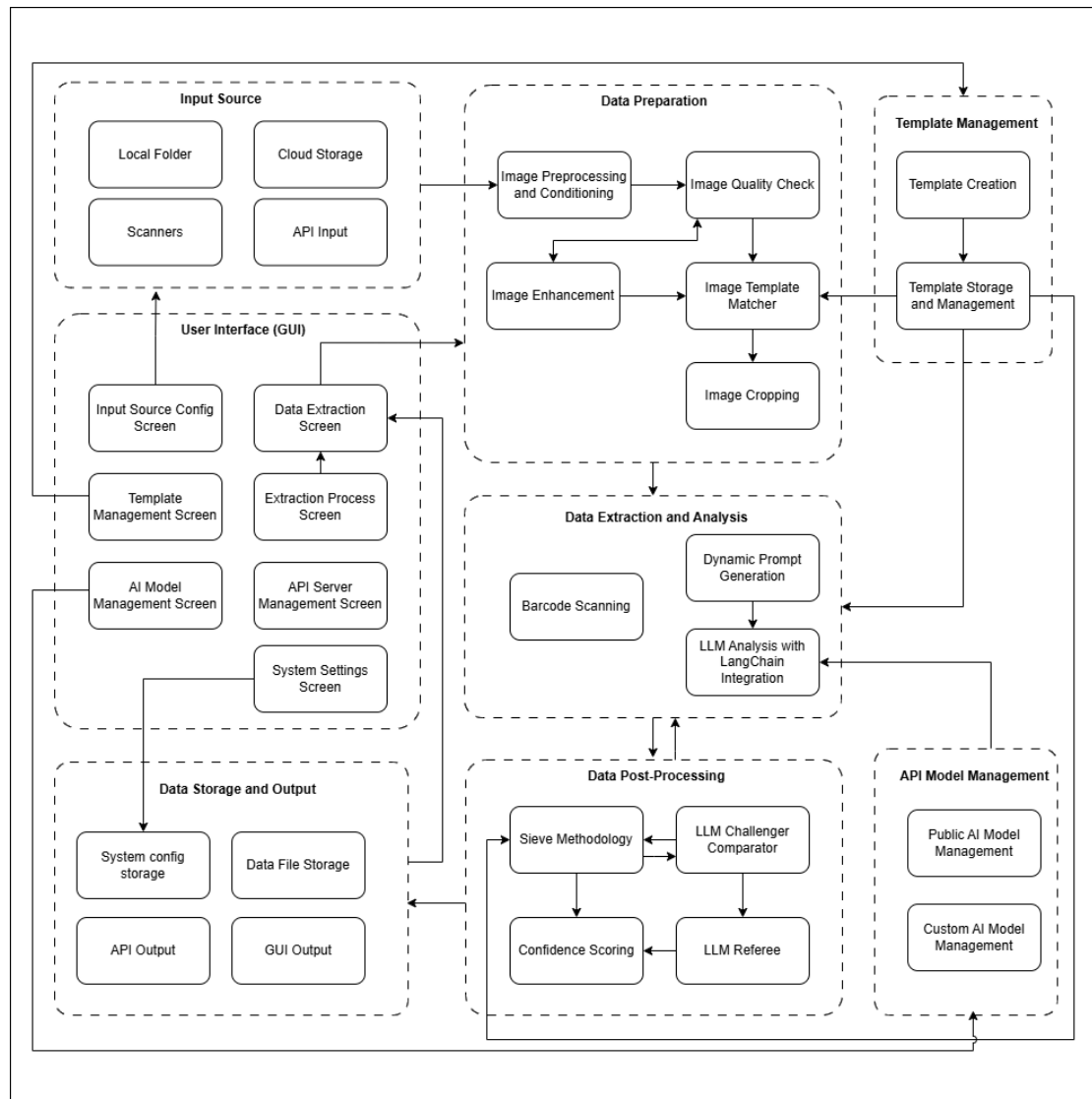


Figure 4.3.1 System Block Diagram

The workflow begins with various **Input Sources**, including local folders, cloud storage, scanners, and APIs, enabling the system to ingest images from both manual and automated sources. These images are funneled into a comprehensive Data Preparation phase where they are preprocessed, quality-checked, enhanced, matched to

templates, and cropped to extract regions of interest. This ensures the input data is clean, standardized, and ready for detailed analysis.

The **Data Extraction and Analysis** module plays a central role in the system, utilizing advanced methods such as barcode scanning and dynamic prompt generation to extract and interpret data from segmented images. One of its key features is the integration of large language models (LLMs) via frameworks like LangChain, enabling sophisticated analysis and understanding of document content. After extraction, the **Post-Processing** stage applies multiple validation techniques, including confidence scoring, cross-model comparison, and referee-based arbitration, to ensure the accuracy and reliability of the results.

The system features a user-friendly Graphical User Interface (GUI), allowing users to manage input sources, templates, models, and system settings. This interface plays a critical role in system configuration, monitoring, and real-time control. Meanwhile, the Data Storage and Output segment handles the persistence of configuration files and extracted data, and ensures that outputs are made available either through the GUI or via APIs for integration with other systems.

Finally, the API Model Management block provides administrative control over AI models, supporting both public and custom-trained models to meet diverse processing needs. Overall, this architecture is designed to be highly modular, scalable, and integrative, capable of automating complex document processing workflows across various domains.

4.4 Key Components Specification and Approaches for Data Processing Pipeline

In this section, we will introduce the newly proposed pipeline, which consists of an image preparation phase and a data extraction phase. A key methodology employed in this project is known as the **sieve methodology** in data extraction and validation phase, along with Challenger and Referee. This approach is our main contribution in integrating LLMs into the existing data extraction solution while mitigating the associated non-deterministic risks.

4.4.1 Data Preparation Component

After the images are obtained from input source including local folder, cloud storage, scanners or API input as source from the previous module, there are several types of approaches are proposed to preprocess image, including:

- Image preprocessing and conditioning
- Image quality check
- Image quality enhancement
- Template matching

Image Preprocessing and Conditioning

After system receives files from input source, the system identifies its current format and uses Python libraries including Pillow, pdf2image and OpenCV to convert the image to *.png*. This ensures that regardless of the original format, the image processing pipeline can handle the images consistently without worrying about format-specific quirks. The system is designed to support a variety of image formats, including *.tif*, *.tiff*, *.bmp*, *.jpg*, *.jpeg*, *.png*, *.webp*, *.gif*, *.heic*, *.pdf*, and *.ico*.

The image conditioning involves using powerful libraries such as Python Imaging Library (Pillow), OpenCV, and Scikit-image to enhance the quality and extract meaningful features from the images. These steps include converting the image to grayscale to simplify the data, resizing it to a standard size to ensure consistency, and applying filters, Gaussian blurring to reduce noise and enhance edges. The image will be resized to match the standard dimensions specified in the template width and length provided by the user. Additionally, techniques such as adaptive thresholding and

histogram equalization are used to enhance contrast and improve the visibility of features like text or barcodes.

Image Quality Checking

The purpose of this step is to prevent low-quality images, those that are too blurry, noisy, or have poor contrast, from degrading the accuracy of the subsequent processing tasks. This project assesses various image quality metrics, such as brightness, contrast, sharpness, noise, blurriness, edge density, and resolution.

The image is first loaded, and various filters and statistical measures are then applied to assess the quality metrics: brightness and contrast are derived from the grayscale image's statistical properties, sharpness is evaluated by detecting edges, and noise is estimated by comparing the original image with a filtered version. Blurriness is measured using the variance of the Laplacian, while edge density is determined by detecting edges with the Canny algorithm. Finally, the image's resolution is checked to ensure it meets the minimum acceptable standards.

It then calculates a composite quality score based on the aforementioned metrics. The quality score is a weighted sum of these metrics, each normalized to a 0-100 scale. This score represents the overall quality of the image, with higher scores indicating better quality. If the quality score meets or exceeds a predefined threshold (e.g., 70), the image is considered acceptable for further processing; Otherwise, it is passed through an image enhancement module. However, should the image still fail to meet the quality criteria after enhancement, it is rejected, flagged as unsuitable, and the user is notified that it does not meet the required standards.

Image Enhancement

If image quality still fails validation after the initial preprocessing steps, additional and more advanced image enhancement techniques may be required to further refine the data for successful extraction. These next-level improvements aim to address more complex issues such as extreme blurriness, uneven lighting, compression artifacts, or poorly contrasted features that are not resolved by standard grayscale conversion, blurring, and thresholding.

Our approach is to apply deblurring or sharpening filters, such as the unsharp mask or Wiener filter, which can help recover edge detail lost in blurry or low-resolution scans. These methods enhance the clarity of boundaries between regions, which is particularly important for extracting text or structured data. In cases of uneven illumination, applying illumination correction or background subtraction help normalize brightness levels across the image, ensuring that lighter or darker areas don't obscure important features.

Template Matching

In this project, template matching serves as a fundamental mechanism for automating document classification and data extraction. It enables the system to intelligently identify the correct template to apply for each incoming document image, accommodate a wide range of document types and workflows., thereby streamlining the extraction of structured data. The system incorporates two primary approaches for template matching: **Content-based Matching** and **Barcode-based Matching**, both of which can be configured based on the use case or the nature of the documents being processed.

Content-based Matching relies on the visual and spatial features of the document. In this approach, the system first applies Optical Character Recognition (OCR) using PaddleOCR to both the input image and the candidate templates to extract "blocks" of text. Each block includes not only the textual content but also its Region of Interest (ROI) along with spatial metadata, which captures its location and dimensions within the image. Then, each text block from the input image is compared to the text blocks from every available template using a multi-faceted scoring system. The system compares each block in the input image to every block in each template based on two criteria: text content and spatial alignment.

For the content comparison, it uses a combination of exact label matching and fuzzy string matching (via the FuzzyWuzzy library) to accommodate textual variations or OCR inaccuracies. For spatial comparison, it calculates the Intersection over Union (IoU) between the ROIs to evaluate how well blocks align in terms of position and size. It conducted this by evaluating the degree of overlap between corresponding regions. Next, a weighted score is computed for each pair of blocks, and the average of the best matches is used to derive an aggregate similarity score for each template. The template

with the highest overall similarity score is then selected for data extraction. However, if all templates score below the predefined threshold, a fallback mechanism is triggered, either defaulting to a predefined template or skipping processing entirely and notifying the user. This behaviour is configurable based on user preferences.

On the other hand, **Barcode-based Matching** leverages the structured information or **encoded identifiers** embedded in barcodes for template identification. This method is especially effective in environments where documents are systematically tagged with unique barcodes. The system first scans and decodes the barcode(s) present in the image. Using user-defined settings such as start position and number of characters, it extracts a specific substring from the barcode value. This substring is then compared against template names. If a match is found, the corresponding template is chosen. This approach provides a high degree of precision and is suitable for use cases involving standardized forms or documents with consistent barcode formats and identifiers, such as shipping labels, invoices, or medical forms.

These two approaches are integrated into the system in a configurable and user-centric manner. Through the GUI, users can select either “Content-based” or “Barcode-based” matching as their default mode. During the data extraction process, if the “Auto” template matching setting is enabled, the system automatically applies the selected approach to identify the correct template for each image. If barcode-based matching is enabled, the system adheres to the configured substring settings for parsing barcode values. Furthermore, in the case where a match cannot be found using the chosen method, the system follows a predefined fallback behaviour, such as defaulting to a specific template or alerting the user, to ensure robust handling of exceptions.

Image Cropping

The cropping approach in this project is designed to extract specific regions of interest (ROIs) from an image, which are defined either by user interaction through the web application for predefined coordinates. This approach is crucial for isolating relevant parts of an image, such as a postcode information, so that only the relevant sections of an image are processed further and avoid interference of extraneous information. By extracting these areas, the system can apply specialized LLM processing techniques tailored to each region.

Before cropping, the image undergoes a conditional resizing step that maintains high resolution and visual clarity. The resizing logic checks if the image's dimensions significantly differ from the expected template size. If the difference exceeds a defined threshold, the image is resized accordingly using **LANCZOS4 interpolation** for upscaling and **INTER_AREA interpolation** for downscaling. This selective and quality-aware resizing ensures consistency across diverse inputs without degrading image quality, which is crucial for accurate ROI extraction.

ROIs are defined either manually by users or programmatically through template data defined by the user. The system also merges overlapping ROIs when multiple fields share the same coordinates, allowing efficient grouping and labelling of fields. To account for potential field detection issues, such as format mismatches or blank regions, the system offers a recovery mechanism. Specific ROIs can be re-cropped with expanded padding. These larger ROIs are saved and can be recombined into a new image, ensuring the extraction process is resilient even when expected data is partially missing or misaligned.

Once ROIs are successfully cropped, each segment is labelled using metadata derived from the ROI template. These labelled crops are then combined into a single image for easier consumption by downstream systems or LLMs. The combination logic is layout-aware and optimized to reduce whitespace and minimize token usage. This involves several steps:

1. The system calculates each cropped image's width and height, as well as the width of its associated text label.
2. The images are sorted by width in descending order to define the maximum width of the combined image canvas.
3. Images are arranged into rows, fitting as many as possible per row without exceeding the canvas width. Row height is dynamically tracked to ensure consistent vertical alignment.
4. The total height of the combined image is computed by summing the row heights and adding padding for labels.
5. Finally, a blank canvas is generated, and all images are placed in their calculated positions with centered alignment and labels rendered above each crop.

This results in a compact, neatly arranged final image that reduces redundant space, maximizes visual clarity, and aligns well with LLM input constraints. All the input images and their labels, with each image centered and aligned to avoid any excess white space or misalignment enables LLM to recognize and process while reducing token usage. The inclusion of fallback and retry mechanisms, conditional resizing, and metadata-based labelling adds adaptability to the cropping pipeline.

4.4.2 Data Extraction and Analysis Component

There are many types of data to be detected and extracted from images with predefined formats (such as parcel data, documents, checks, invoices, contracts, prescriptions, reports, and records etc.) as in Table 4.4.2.1.

Data to be Extracted	Methods to Extract Data
Printed Text	Proposed Pipeline of LLM Integration
Typed Text	Proposed Pipeline of LLM Integration
Handwritten Text	Proposed Pipeline of LLM Integration
1D Barcode	Barcode Scanning Approach
2D Barcode	Barcode Scanning Approach
Checkboxes and Form Elements	Proposed Pipeline of LLM Integration
Labels and Tag	Proposed Pipeline of LLM Integration
Symbols	Proposed Pipeline of LLM Integration
Existence of Signature	Proposed Pipeline of LLM Integration
Existence of Stamp/ Chop	Proposed Pipeline of LLM Integration
Value of character-based Stamp	Proposed Pipeline of LLM Integration
Short description of picture content	Proposed Pipeline of LLM Integration

Table 4.4.2.1 Types of Methods to Extract Data

In this phase, there are several types of approaches are proposed to extract different types of data, including:

- Barcode scanning approach
- Proposed pipeline of LLM Integration for data extraction, which is the main focus of this project, where in this section we will focus on:
 - Dynamic Prompt Generation
 - LangChain Integration
 - Prompt engineering of LLM

Barcode Scanning Approach

The barcode scanning approach in this project is designed to automatically detect and decode barcodes from images uploaded by users, a critical functionality for extracting structured data embedded within images such as tracking numbers, product codes or inventory details. The process is straightforward: it is implemented using OpenCV and the Pyzbar library, facilitates this process by first loading the image and then identifying any barcodes present within it. Then, it retrieves the decoded barcode data as a string, which can be used in subsequent processing steps or exported for further use.

The supported 1D barcodes include *EAN-13*, *UPC-A*, *Code 128*, *Code 39*, *Interleaved 2 of 5 (I2/5)*, *Codabar* and *EAN-8*, while 2D barcodes include *QR Code*, *Data Matrix*, *PDF417* and *Aztec Code*.

Dynamic Prompt Generation

Dynamic prompting in this system is a methodical approach that adapts prompt generation for the Large Language Model (LLM) based on the specific characteristics of each document and the extraction goals defined in associated templates. Rather than using a static prompt, which can be too generic and error-prone, the system constructs highly tailored prompts that instruct the LLM with precision, improving both the accuracy and reliability of the extracted data.

The process begins with template-driven prompt generation. Each document is associated with a template that defines metadata such as the type of document (e.g., invoice, shipping label), the language it's written in, the fields to extract, and whether it contains structured data like tables. This template guides the construction of the base prompt using the `generate_dynamic_prompt` function. For instance, if the document is multilingual or contains tabular data, the prompt explicitly tells the LLM to account for those conditions. By incorporating the document type and field-level descriptions directly into the prompt, the system provides the LLM with rich context, allowing it to “understand” the nature and structure of the document it's analyzing.

To further control the output format and ensure structured responses, the system generates a JSON schema using the `generate_schema_properties` function. This schema outlines the expected fields, data types, and constraints (like expected character length

or example values), and is embedded in the LLM prompt. The LLM is instructed not only to extract specific values but to return them in a JSON object that matches this schema, significantly reducing the need for post-processing or manual validation.

In cases where the initial extraction fails, due to format mismatches, OCR errors, or ambiguous field values, the system employs retry prompting through the `generate_retry_prompt` function. These retry prompts are specifically crafted to target the problematic fields and provide the LLM with feedback from the previous attempt, including the erroneous values and the type of error encountered. The retry prompt may also restate constraints or expected formats, effectively guiding the LLM to correct its output. This iterative refinement ensures durability and boosts the system's ability to handle noisy or inconsistent inputs.

In essence, dynamic prompting in this pipeline is a feedback-aware, context-rich orchestration that leverages template metadata and structured schemas to guide LLMs toward precise and schema-compliant outputs. It balances flexibility and structure, enabling general-purpose language models to perform highly specialized extraction tasks with minimal manual intervention.

LangChain Integration

LangChain is integrated into the system as a foundational framework that brings structure, modularity, and resilience to the way Large Language Models (LLMs) are accessed and used. Rather than making direct API calls to LLM providers like OpenAI or Google, the system relies on LangChain's abstraction layers and tooling to streamline model interaction, prompt construction, and output validation. It abstracts model APIs, standardizes prompt structures, enforces output schemas, and provides a scalable framework for future enhancements. This integration allows the system to scale, adapt, and maintain LLM functionalities with significantly less effort and risk of inconsistency.

The system introduces a LangChain factory which encapsulates the logic for instantiating different LLMs. Whether the system uses OpenAI's ChatOpenAI or Google's ChatGoogleGenerativeAI, the LangChain provides a centralized and extensible interface for switching or adding models without changing downstream logic. The system further uses this factory to select and initialize the appropriate model. This

separation of concerns ensures that the business logic for image analysis and data extraction remains clean and decoupled from the underlying LLM implementations.

LangChain's prompt management tools, such as ChatPromptTemplate, SystemMessage, and HumanMessage are used to construct well-structured messages that guide the LLM in generating reliable outputs. In particular, the system uses a SystemMessage to define the LLM's role (e.g., an expert in data extraction) and a HumanMessage that includes both the image content (base64-encoded) and a dynamic prompt. This structured communication format helps the LLM better understand the task context, leading to improved accuracy and consistency in the responses.

To ensure the output from the LLM is usable and properly formatted, the system uses LangChain's JsonOutputParser. This parser works in tandem with schema definitions to validate that the LLM's response matches the expected structure, such as a JSON object with specific fields and types. This parsing layer is essential for automated downstream processing and reduces the risk of runtime errors due to malformed data.

Finally, the system simplifies image analysis workflows through the use of the a wrapper around the LangChain-enhanced model abstraction. They will invoke LangChain-enabled processes while hiding the complexity of message construction, schema enforcement, and model selection from higher-level code. This results in a clean and maintainable interface for document analysis tasks.

Prompt Engineering of LLM

In this project, prompt engineering has been of central importance in guiding a large language model (LLM) to accurately extract data from images, particularly in scenarios where the images contain structured information such as parcel details. The purpose of this approach is to leverage the powerful natural language processing capabilities of the LLM to interpret and extract specific data points from images, and to return the results in a well-defined JSON format that can be easily processed further.

The prompt is constructed with specific instructions, asking the model to extract data from the image and return it in a structured JSON format. To enhance the LLM's understanding, the prompt includes several techniques of prompt engineering including:

Role Assignment and Context Setting

In the payload, the role of the LLM is explicitly defined in the messages field under the role key. The system message is used to establish the role of the LLM and context as “an expert in data extraction.” This is important as it informs the LLM to focus on tasks related to this domain, potentially enhancing the relevance and accuracy of the responses.

Prompt Design

The prompt variable provides a clear and concise instruction: “Extract data from the image below and return in JSON.” This direct instruction helps the LLM understand the task it needs to perform and specifying the type of data expected. The prompt combines text and image processing by providing a base64-encoded image within the conversation. This is done through the base64 library function and then embedding it within the user’s message in a specific format that the model can process.

Schema Enforcement

It specifies the desired data schema, which outlines the structure and type of each expected data field. The `json_schema` in the `response_format` payload defines the exact structure of the output, including required fields, data types, and descriptions according to template defined. By setting a strict mode with `strict: True` within the `json_schema`, the LLM is explicitly instructed to follow the schema rigidly, reducing the likelihood of deviations in the output structure. This schema acts as a strict guideline for the LLM to follow, which is critical for ensuring that the output is well-formed and adheres to the desired format.

Parameter Tuning

The `temperature` parameter is set to 0, indicating that the model should generate deterministic and focused responses. A lower temperature is useful when the goal is to achieve precision and consistency, especially in tasks that involve structured data extraction.

The `max_tokens` parameter is set to 300, which limits the length of the response and token. This constraint helps in managing the cost associated with the API call and ensures the response remains concise and within expected limits.

Cost and Performance monitoring

The system calculates the number of tokens used in the prompt and the completion, then estimates the cost based on token usage. This is crucial for understanding and optimizing the cost-efficiency of using the LLM, particularly for tasks that might involve high volumes of data or frequent API calls. The code also measures the processing time for each API call, which is important for assessing the performance of the LLM and optimizing the workflow in real-time applications.

Error Handling

The code includes thorough error handling for API responses, checking for status codes and attempting to decode the JSON response. It also raises specific exceptions if the response is not as expected. This is important in production environments where stability and reliability are crucial. The system is designed to handle potential errors like `JSONDecodeError` or `KeyError`, ensuring that the system can gracefully manage unexpected outputs or issues without crashing.

Token Estimation

The system also estimates the number of tokens an image might consume based on its dimensions. This is a unique aspect of prompt engineering for image-based LLM tasks, ensuring the images are processed in a way that optimizes token usage and adheres to model limits.

All in all, the payload sent to the LLM API includes this prompt, the image, and a detailed JSON schema that describes the expected output structure. The API response is processed to extract and validate the JSON content, ensuring that it conforms to the specified schema. Additionally, the prompt engineering includes managing token usage efficiently by setting constraints on the number of tokens, temperature (which controls the randomness of the output), and the maximum allowed tokens, ensuring that the model's response is both cost-effective and aligned with the task's requirements.

4.4.3 Data Post Processing Component

In this section, **Sieve Methodology**, shown in Figure 4.5..1., is a main contribution in this project proposed to ensure the accurate and reliable extraction of data from images using LLM. This is part of the pipeline extracted from Figure 4.2.2.2 in Section 4.2.2.

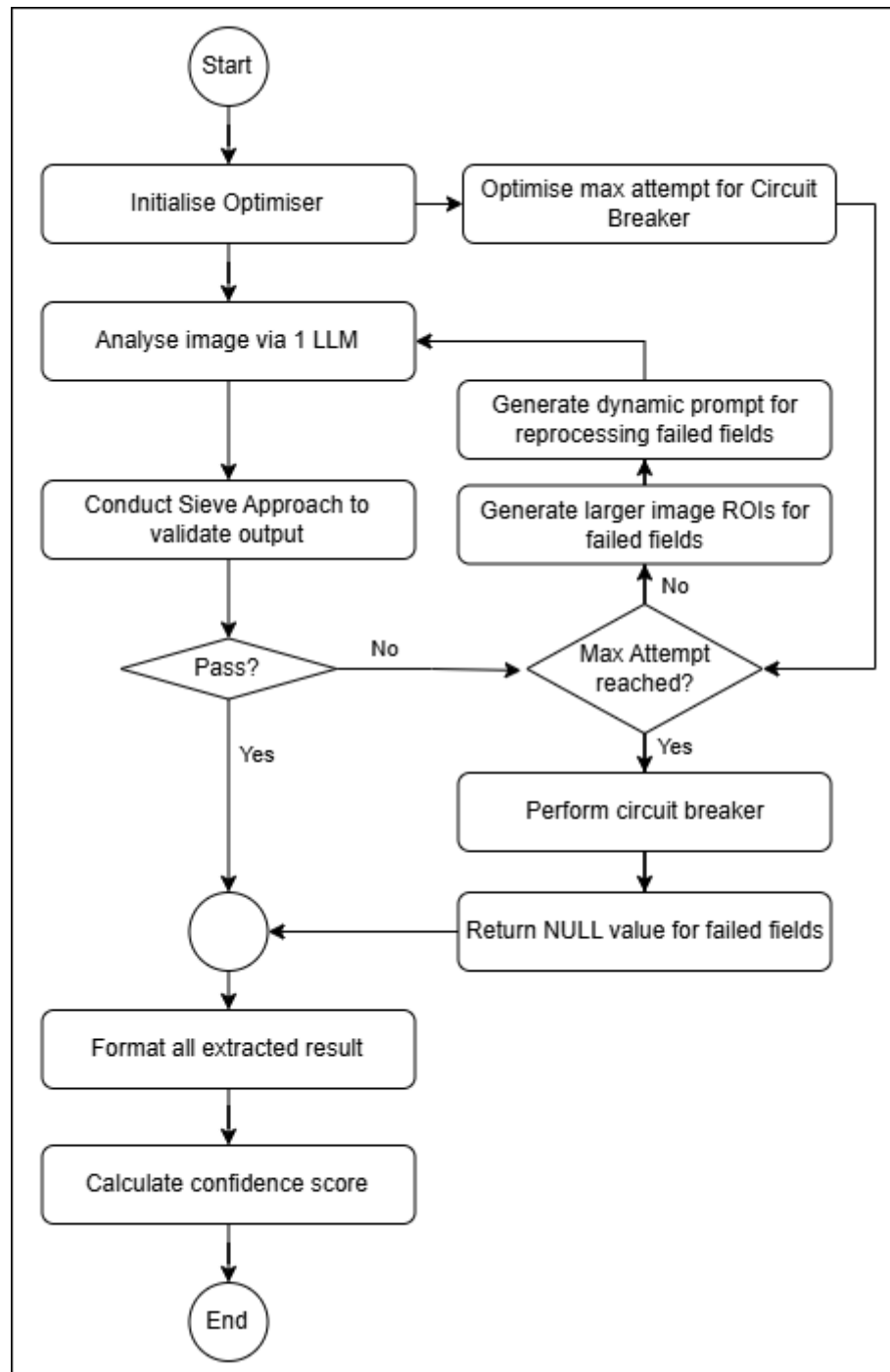


Figure 4.4.3.1 Simplified Sieve Methodology Flowchart

Figure 4.4.3.1 provides an overview flow of how the Sieve approach works. The Sieve approach is a systematic method to integrate LLMs into systems, especially given unpredictability of LLMs. It involves multiple iterations of validation, comparison, and refinement of the results to mitigate the non-deterministic nature of LLMs. It is designed to separate the outputs of the LLM that do not conform to the required format or criteria.

This method is vital for integrating LLMs into systems where consistency and precision are paramount. The purpose of this module is also to optimize the batch processing using Optimizer algorithms. If the inputs are given in batches, this module will track the inputs with corresponding outputs by adding a unique ID for each input.

Step 1: Initial Attempt - Data Validation

It begins with an initial pass where a single large language model (LLM) is tasked with analyzing an image and extracting relevant data fields based on the defined template or format expectations. Once the initial extraction is complete, the output is passed through a validation layer, which applies predefined rules to check for correctness, completeness, and conformance to expected data formats. If the extracted data meets all the criteria, it is accepted and processed for final output formatting, indicating a successful run on the first attempt.

Step 2: Subsequent Iterations – Dynamic Reprocessing and Circuit Breaker

When the extracted data fails to meet validation requirements, the methodology activates a built-in retry mechanism. This process acknowledges that extraction errors can occur due to issues such as tight image crops, ambiguous prompts, or misinterpretation by the LLM. To address this, the system incrementally enhances the input to the LLM for better results. Specifically, it enlarges the Regions of Interest (ROIs) around failed fields, increasing the context visible to the model. In parallel, the system dynamically generates new prompts that are more tailored to the specific failures encountered, providing clearer instructions, incorporating hints, or emphasizing specific formats expected. These adjustments aim to help the LLM produce more accurate outputs on subsequent attempts.

The system allows up to three reprocessing cycles for each image. This controlled loop ensures that the LLM has multiple opportunities to correct earlier mistakes while keeping the process efficient and avoiding infinite repetition. After each iteration, the updated outputs are revalidated using the same Sieve criteria. If the data still fails to meet the validation checks after three retries, the methodology initiates a mechanism referred to as a “**Circuit Breaker**.” This mechanism halts further attempts and flags the document for alternative handling, whether that means sending it for manual review, storing it with warnings, or returning it with an error status.

At the conclusion of this process, whether the data passed validation initially or after one or more retries, the final step involves formatting the output. This ensures consistency and readiness for downstream consumption, whether the data is exported to a system, stored in a database, or returned through an API. In essence, the Sieve Methodology balances intelligent automation with reliable error handling, significantly enhancing the reliability of document processing workflows while minimizing the need for manual intervention.

Step 3: Optimizing the Process - Optimizer

Even with a circuit breaker in place, the system may still risk excessive looping and wasting computational power, particularly in situations where consistent results are unattainable. To address this, the Sieve approach incorporates an Optimizer. It serves to further refine the process by addressing two critical aspects of the system: managing the number of inputs processed by the Large Language Model (LLM) per batch and limiting the number of reprocessing loops for each image.

Dynamic Batch Size Optimization

The first function of the Optimizer focuses on dynamically optimizing the number of submissions (inputs) in each batch sent to the LLM. LLM prompts inherently carry overhead, as each submission requires a prompt that consumes tokens, adding to the overall processing cost. The overhead can be reduced by having more inputs in a batch; such a batch processing approach will also improve the overall processing time. This dynamic adjustment ensures that the system can handle larger data volumes efficiently when conditions are favourable while reducing the batch size when more precise and careful analysis is required.

In this work, the optimizer algorithm uses a “doubling and trimming” strategy: double the inputs per batch if there are no errors, reduce the inputs according to the number of correct outputs from the LLM. In other words, when there are no errors in the outputs from the LLM, the Optimizer doubles the number of inputs processed in the next batch. However, if errors are detected in the outputs, the Optimizer reduces the number of inputs in subsequent batches. The reduction is proportional to the number of correct outputs, allowing the system to focus more precisely on problematic areas that may need more granular attention. Hence, it can dynamically optimize the number of submissions per batch provided to the LLM.

Optimization of Loop Count

The second function of the Optimizer is to manage the maximum number of reprocessing loops allowed for each image. The Optimizer fine-tunes maximum of attempts to extract and validate data from an image by identifying the most efficient loop count that balances thoroughness with efficiency.

The Optimizer algorithm evaluates the average number of submissions needed to achieve consistent and accurate results across multiple images. It then sets this average as the optimized maximum number of loops for future image processing tasks. Once reached, it instructs the Circuit Breaker to break the loop. In practice of our experiment, this often results in this specific system being capped at a maximum of five loops, which the Optimizer has determined to be the most effective limit. Thus, we decide it as our maximum number of loops, which is to allow reanalysis for at most five attempts. In this case, if after five attempts some values still have not been validated and recorded in correct result, the system will adopt a fallback strategy to handle the stubborn cases as illustrated in the last step.

By optimizing the loop count, the system avoids unnecessary repetitions that would otherwise waste computational resources and increase processing time. This approach ensures that the system exits the loop once diminishing returns are identified, where additional loops no longer contribute significantly to improving the results.

Step 4: Final Output - Fallback Strategy to Handle Stubborn Cases

Once the Circuit Breaker is triggered before all outputs get validated result, the system will automatically fill in the remaining missing values with NULL value. In a data extraction system using a LLM), returning a NULL value is a prudent approach when the model is uncertain, or the extracted data fails validation checks. This ensures the integrity and reliability of the output by clearly indicating that a valid answer could not be confidently determined. Rather than risking the inclusion of incorrect or misleading data, which can compromise downstream applications or analytics, a NULL value acts as a transparent placeholder for missing or unverifiable information. It also facilitates easier error handling, auditing, and future reprocessing, either by retraining the model, refining the prompt, or introducing a human-in-the-loop review process. The final results are then organized into a specific format, ensuring that they match the expected structure for further use or storage.

The Sieve methodology, including Circuit Breaker and Optimizer successfully integrates LLMs into existing systems by addressing the inherent unpredictability of LLM outputs. Through a structured process of validation, comparison, and iterative refinement, it ensures that only the most reliable data is ultimately recorded and used. This method not only improves the accuracy of data extraction but also enhances the overall effectiveness and reliability of the system, making it suitable for integration into applications where precision is critical.

Confidence Scoring

Confidence scoring in this system is a vital quality assurance layer, enabling the application to evaluate the reliability of the extracted data before it is used downstream. After the data is extracted from a document, the system analyzes multiple indicators that reflect extraction consistency, accuracy, and overall reliability. These indicators include structural elements such as how many fields were successfully extracted, how many failed validation (e.g., regex checks), how often the system had to retry extractions, and whether field values changed during retries, where each of which suggests instability or uncertainty. Additionally, where supported, the system leverages deeper model-level indicators like average log-probability and entropy derived from the language model's prediction outputs. Lower entropy and higher log-probabilities suggest higher confidence in the generated text.

To quantify this, the system uses two pre-trained predictive models. If advanced LLM metrics (logprob and entropy) are available, model1 is used for greater precision. If these values are missing, model2, which is a simplified version that excludes LLM-specific metrics is employed. These models loaded will process the collected features to produce a final confidence score, typically normalized between 0 and 1. This score is then stored alongside the extracted data and used throughout the application to guide decision-making. For instance, data with low confidence scores might be flagged for manual verification, while high-confidence entries can be processed automatically.

4.5 Components Specification and Approaches for Supporting System

This section provides a detailed explanation of the supporting components of the system, including the input source, data storage and output modules, as well as the template management component.

4.5.1 Input Source Component

The Input Source component in this project serves as the entry point for acquiring images that will be processed by the system. It is designed with flexibility in mind, supporting multiple methods for input, including local folders, cloud storage services, USB scanners, and remote WebDAV-based document scanners. This component is managed through a user interface that allows users to add, configure, and manage these sources seamlessly. Internally, it maintains a set of active connection handlers that manage real-time interaction with each input source, enabling the system to monitor multiple channels in parallel. Through this interface, users can establish connections, initiate scanning, or select folders, ensuring smooth and customizable integration with diverse environments.

Each specific input source is implemented as a separate module, ensuring modularity and scalability. Local folder input is handled using the watchdog library for real-time file monitoring, while cloud storage connections use official APIs to fetch and periodically poll for new image files. Scanning input is handled via the WIA API for USB scanners and the webdav3 library for remote OSS scanners. The architecture is threaded to support asynchronous operation, ensuring the application remains responsive during long-running tasks like scanning or large file downloads.

4.5.2 Data Storage and Output Component

The Data Storage and Output component in this project helps in managing how extracted data is preserved and presented to users or external systems. It supports saving data in multiple formats such as Excel, CSV, JSON, XML, Markdown, and plain text. This flexibility ensures compatibility with a wide range of use cases, from simple text-based exports to structured datasets suitable for further analysis or reporting. The system can either create new output files or append to existing ones, and it intelligently organizes data, for example, by supporting multiple Excel sheets when required. It also ensures that output directories exist and are properly structured. The module encapsulates the file-writing logic necessary for each format, handling tasks like delimiter formatting for CSVs or schema conformity in XML.

In addition to file-based storage, the system supports data delivery through both its API and GUI. The API endpoint allows programmatic access to the extracted data, typically delivered as JSON for easy integration with other software systems. On the GUI side, the screen provides a user-friendly interface to view, browse, search, and download output files.

4.5.3 Template Management Component

Templates act as blueprints that instruct the system where to look for specific data within a document and how to interpret it. Without these templates, the system would lack context about the structure or layout of various document types. This component supports both manual and automated methods for template creation, and provides reliable tools for storing, editing, and organizing templates for reuse.

Template creation begins with a user-driven process through the graphical interface. Users initiate this by supplying key details such as the template name, description, and the language used in the document. A sample image of the document is then uploaded, and the user defines specific Regions of Interest (ROIs), these are visual selections over the document that correspond to data fields to be extracted. For each ROI, the user specifies metadata like the field name, expected data type (e.g., number, date, text), and optional validation rules. This GUI-guided process, managed by screens such as `basic_details_screen.py` and `roi_selection_screen.py`, ensures that even users with limited technical expertise can effectively define templates.

To streamline the template creation process, the system also supports **automatic template generation using OCR and LLM technologies**. The module (`lib/templateCreation.py`) begins by using PaddleOCR to extract both the text and its layout coordinates from the sample document. Then, a language model such as GPT-4o or Gemini is invoked to intelligently infer which text segments represent key-value pairs, and to map those relationships to likely ROIs. The output is a draft template containing suggested field names and regions, which the user can review and refine. This feature is particularly useful for accelerating onboarding of new document types or handling complex layouts that might be time-consuming to annotate manually. Once created, templates are stored in a structured JSON format, with associated sample images saved as PNG files.

4.6 System Optimization Design

This section will focus on system optimisation design, in which the system incorporates a thoughtful set of optimization strategies across its architecture to ensure high performance, responsiveness, and reliability, particularly under concurrent workloads.

4.6.1 Thread Management

Thread management allows critical operations to run independently without freezing or slowing down the main application. For example, the API server, built using FastAPI and Uvicorn, is launched in its own thread, allowing it to serve external requests without interfering with the GUI's responsiveness. Likewise, input monitoring for local directories and remote WebDAV sources is performed in dedicated threads, ensuring that these blocking I/O operations do not hinder the system's core workflow. To maintain data integrity in such a multithreaded environment, thread locks are used when accessing shared resources such as output files, thus preventing race conditions and corruption.

4.6.2 Asynchronous Operations

Another important optimization layer is asynchronous operations, particularly around file I/O. By leveraging asynchronous libraries like `aiofiles`, the system allows API endpoints to handle multiple file uploads concurrently. This means that the server can accept a new request while previous files are still being saved, dramatically improving

throughput and response time during heavy usage. Complementing this is the system's efficient file handling approach, which includes lazy loading mechanisms that process files only when necessary, and clean, error-resistant file path management using best practices like `os.path.join`.

4.6.3 Memory and Resource Management

One key technique used for memory and resource management is the explicit invocation of garbage collection immediately after intensive image processing tasks. Image handling, such as loading large files, performing transformations, or generating Regions of Interest (ROIs), can consume significant memory. By calling Python's garbage collector explicitly at strategic points in the image processing pipeline, the system forces the cleanup of unused objects and memory, maintaining a lean memory footprint even during continuous or batch processing scenarios.

In parallel, the system exhibits disciplined handling of temporary files, particularly during API-based operations. To avoid the intermediate results clutter filesystem and degrade I/O performance, the system ensures that temporary files are only created when absolutely necessary and are tracked meticulously. Once their role is complete, they are deleted immediately. This not only prevents long-term accumulation of unused files but also supports faster and more predictable disk access times.

CHAPTER 5 SYSTEM IMPLEMENTATION

This section presents the design specifications, including hardware specification, software specification, user requirements, system performance criteria, and the project's verification plan to ensure the project meets its objectives.

5.1 Hardware Setup

This section specifically focuses on the design and components of the hardware system being developed or utilized as part of the project. This project involves several hardware components, including a processing unit (laptop) and high-resolution webcam.

Processing Unit

A computer or a dedicated processing unit or laptop receives data from the input sources and runs the modules to analyse the images. As shown in the Table 5.1.1, a laptop Yoga Slim 7 Pro 14ACH5 with 16GB RAM is used in this case for executing the software and algorithms that integrate LLM for data extraction from unstructured data sources, such as images. It then conveys the results back to the desktop applications for the user's reference or be used in the subsequent pipeline. enable higher processor power for data processing, smooth multitasking, and real-time operations.

Description	Specifications
Model	Yoga Slim 7 Pro 14ACH5
Processor (CPU)	AMD Ryzen™ 7 5800H
Operating System	Windows 11
Graphic	Integrated AMD Radeon™ Graphics
Memory	16GB Soldered DDR4-3200
Storage	512GB SSD

Table 5.1.1 Specification of Processing Unit

5.2 Software Setup

This section outlines the software setup process, covering the development environment (IDE), software specifications, application architecture, database and file operations configuration, as well as the setup and functionality of the API server.

5.2.1 Integrated Development Environment (IDE) and Languages

The system proposed can be broken down into several key modules, each responsible for specific functionalities within the overall architecture. Below are the software specifications of the projects, including the Integrated Development Environment (IDE) and programming languages in Table 5.2.1. and Table 5.2.2. Before starting the project development, two software programs need to be installed and downloaded onto the laptop, which are Visual Studio Code (VS Code) and Git, as shown in Table 5.2.1.

Visual Studio Code	Version	1.92.2 (user setup)
	OS	Windows_NT x64 10.0.22631
	Software dependences and external libraries	Refer to Software Specifications section in Chapter 5.2.2.
	Python SDK Version	3.12.3.
	Purpose	For main development and testing platform
Git	Version	2.46.0
	Purpose	For version control, branching for testing and merging to handle projects efficiently.

Table 5.2.1.1 IDE and Git

Python serves as the primary programming language for the entire system, chosen for its versatility, readability, and strong ecosystem of libraries suited for data processing and machine learning tasks. For the development of the graphical user interface (GUI), the project utilizes the **PySide6** library, Qt for Python, which supports the creation of modern, responsive, and cross-platform desktop applications. PySide6 enables efficient integration between the user interface and the system's core logic, ensuring a seamless and interactive user experience across different operating systems.

5.2.2 Software Specification

Table 5.2.2.1 illustrates most of the key libraries and dependencies that are used in this project, where they define the core application logic for image processing, API, data handling, system integration and GUI management.

Libraries and Dependencies	Version	Description
API and Web Framework		
To create the web API, handle requests, and communicate with other services.		
<i>uvicorn</i>	0.34.2	A fast ASGI (Asynchronous Server Gateway Interface) server, used to run the FastAPI application.
<i>requests</i>	2.32.3	A library for making HTTP requests (used for communication with external APIs).
<i>base64</i>	Standard	To encode image into text format to be passed via HTTP requests.
<i>fastapi</i>	0.115.12	A modern, fast (high-performance), web framework for building APIs with Python.
<i>urllib3</i>	2.4.0	A powerful and user-friendly HTTP client library to make HTTP requests.
<i>langchain</i>	0.3.25	To provide tools for prompt management and LLM interaction.
GUI		
To create the graphical user interface.		
<i>PySide6</i>	6.9.0	To create GUI and provide core GUI functionality.
<i>lazy_loader</i>	0.4	A library for lazily loading Python modules.
Image Processing		
For image manipulation, conversion, and analysis.		
<i>cv2 (OpenCV)</i>	4.11.0.86	A library for computer vision tasks (e.g., image reading, writing, processing).
<i>scikit-image</i>	0.25.2	Library for image processing.
<i>Pillow (PIL)</i>	11.2.1	Python Imaging Library for working with images.
<i>pdf2image</i>	1.17.0	Library to convert PDF pages to images.
<i>paddleocr/paddlepaddle</i>	2.10.0/ 3.0.0	For text detection and recognition with coordinates.

<i>fuzzywuzzy</i>	0.18.0	To calculate string similarity and differences, enabling fuzzy string matching.
Data Handling To work with Excel files, JSON data, and other data formats.		
<i>json</i>	Standard	For encoding and decoding JSON data (used for API communication, data storage, and configuration files).
<i>openpyxl</i>	3.1.5	A library for working with Excel files (reading and writing).
<i>tifffile</i>	2025.3.30	A library for reading and writing TIFF files, which are a common image format.
<i>xml.etree.ElementTree</i>	Standard	Python library to parse and create XML documents.
<i>pyzbar</i>	0.1.9	A library for reading barcodes and QR codes from images.
<i>python-dotenv</i>	1.1.0	Library to read key-value pairs from a .env file and set them as environment variables.
Data Analysis To analyse and manipulate data.		
<i>pandas</i>	2.2.3	A powerful library for data analysis and manipulation. It provides data structures like DataFrames and Series.
<i>numpy</i>	2.2.5	The fundamental package for numerical computation in Python. It provides support for arrays, matrices, and mathematical functions.
File System Interaction To manage files and directories, perform asynchronous file operations, and monitor file system changes.		
<i>aiofiles</i>	24.1.0	Asynchronous file operations for aiohttp
<i>watchdog</i>	6.0.0	A library to monitor file system events.
<i>os</i>	Standard	Provides functions for interacting with the operating system (e.g., file path manipulation, directory creation).
<i>sys</i>	Standard	Provides access to system-specific parameters and functions (e.g., modifying the Python path).
<i>shutil</i>	Standard	To enable high-level file operations such as copying, moving, archiving, and removing files and directories
<i>pywin32</i>	310	Used to interact with Windows Image Acquisition (WIA) for scanner control.
Data Validation To validate data formats and ensure data integrity.		
<i>email_validator</i>	2.2.0	Library to validate email addresses.
<i>re</i>	Standard	Regular expression operations for pattern matching in strings (e.g., data validation, string manipulation).
Machine Learning		

To load pre-trained machine learning models.		
<i>joblib</i>	1.4.2	Tools for saving and loading Python objects efficiently.
<i>scikit-learn</i>	1.6.1	A machine learning library that provides tools for classification, regression, clustering, and other machine learning tasks.
Concurrency		
To handle multiple tasks concurrently, improving performance.		
<i>asyncio</i>	Standard	Library for writing concurrent code using the async/await syntax (used for asynchronous file operations in the API).
<i>threading</i>	Standard	Support for creating and managing threads, enabling concurrent execution (used for running the API server in a separate thread).
<i>threadpoolctl</i>	3.6.0	A library to limit the number of threads used by other libraries.
General Utility Operation		
<i>numpy</i>	2.2.5	The fundamental package for numerical computation in Python. It provides support for arrays, matrices, and mathematical functions.
<i>datetime</i>	Standard	For manipulation, formatting, and calculations involving temporal data.
<i>typing</i>	Standard	Support for type hints to improve code readability and help with static analysis.
<i>pyinstaller</i>	6.13.0	To bundle Python applications and their dependencies into standalone executables. This is a key tool for distributing the application, making it run without requiring a Python installation on the user's system.
<i>uv pip</i>	0.6.14	A package manager to manage python package
File Optimization		
<i>gc</i>	Standard	Enable automatic garbage collection for process optimisation.
<i>tempfile</i>	Standard	To provide generic, low- and high-level interfaces for creating temporary files and directories.

Table 5.2.2.1 External Libraries/ Packages

Aside from the libraires and frameworks shown in Table 5.2.2.1, this project mainly leverages many LLMs, specifically gemini-2.0-flash, gpt-4o-2024-08-06, gpt-4.1-2025-04-14, qwen-vl-max and qwen-vl-plus, as a critical component in the data extraction process. These models have been used for their multimodal advanced capabilities in understanding and processing complex language tasks, making them

well-suited for the nuanced challenges of extracting unstructured data from images, including handwritten information. They can also understand context, allowing it to extract unstructured information that is inferred but not explicitly written on the image.

5.2.3 Input Source Setup

Setting up an input source in this system is designed to accommodate a variety of image sources such as local folders, cloud storage services, USB scanners, and networked WebDAV scanners. Available input sources are listed in Table 5.2.3.1.

Input Source	Configuration
Local Folder	Connect the folder directly in the application.
Google Drive	<ol style="list-style-type: none"> 1. In Google Developer Console (https://console.developers.google.com/), create a new project. 2. Enable Google Drive API. 3. Create credentials via OAuth client ID.
One Drive	<ol style="list-style-type: none"> 1. In Azure Portal, (https://www.google.com/search?q=https://portal.azure.com/), register an application. 2. Obtain a credentials.
Dropbox	<ol style="list-style-type: none"> 1. In Dropbox App Console (https://www.dropbox.com/developers/apps), create a new app. 2. Choose Dropbox API and access type as App Folder. 3. Obtain access token.
Webcam	<ol style="list-style-type: none"> 1. Use webcam or download “Droid Cam” app on phone and computer device. 2. Connect the webcam to the device.
USB Scanner	<ol style="list-style-type: none"> 1. Connect USB scanner physically to the computer and turn on. 2. Install any necessary drivers for scanner.
Networked WebDAV Scanner	<ol style="list-style-type: none"> 1. (Optional) Download “OSS Document Scanner” from the mobile app store. 2. Create a WebDAV account ‘https://infini-cloud.net/en/’ or any other provider. 3. Connect the OSS Document Scanner and WebDAV. 4. Obtain hostname, username, WebDAV path and password.

Table 5.2.3.1 Input Sources

Users can add a new input source by selecting the desired source type (e.g., “Local Folder”, “Google Drive”, or “USB Scanner”) and entering relevant details such as a folder path or authentication credentials. Once configured, each source is listed with options to connect, disconnect, or remove the configuration, providing a centralized

hub for managing image inputs. For example, connecting to a local folder or cloud storage can either monitor a folder for new images in real time or periodically poll cloud directories for new files.

5.2.4 Database and File Operation Setup

This application employs a strategic approach to data storage and file operations, balancing the need for efficient access to application resources with the requirements for persistent, user-specific data. The file structure is organized to distinguish between **application-critical, read-only resources** and **user-configurable, modifiable data**.

The use of a dedicated *resources* directory, kept read-only during execution, ensures that all **critical static files** such as icons, animations, and AI models remain untouched and are consistently available regardless of the user's system configuration. This includes the inclusion of pre-trained models and metadata files. For data that varies per user (user configuration and modifiable data), such as API credentials, scanner configurations, and extraction templates, the application uses separate JSON files saved in **user-specific directories** like AppData.

In this project, storing configuration data such as API keys, template definitions, and image source settings in JSON files is preferred due to the lightweight, local, and read-heavy nature of the application. The data structures are relatively small, static, and do not require complex querying or relational integrity, making a full-scale database like PostgreSQL unnecessary and overly complex.

Temporary data created during the **image analysis process**, such as **intermediate** images, such as cropped images or format conversions, is stored in short-lived directories using the tempfile module. File operations are managed using standard Python libraries like `os` and `tempfile`, which enable the application to create and clean up temporary data with specific garbage collector features. This ensures that memory usage is optimized, and leftover files do not clutter the user's system or risk data persistence where it is not needed.

Finally, the application **outputs user-relevant results** into a dedicated and easily accessible location, typically within the user's "Documents" directory. Extracted data is written to output files in Documents, while user-defined configurations are saved to

AppData, as mentioned. This strategy ensures data persistence and traceability of extracted results while maintaining a clear separation between configuration data and output artifacts.

5.2.5 API Server Setup

In this project, an API server is set up using the **FastAPI framework** to enable external applications to interact with the data extraction pipeline. FastAPI is chosen as the framework for its speed, simplicity, and native support for asynchronous operations. This server plays a critical role in bridging the backend processing with potential frontend interfaces or third-party integrations, allowing image data to be submitted, processed, and returned through a structured and scalable approach.

The core functionality is encapsulated in a class that manages the entire API lifecycle, from setting up routes to managing the server status. The server exposes a key endpoint, `/extract_data/`, which accepts image files along with model selection and template details. This endpoint is designed to be flexible: it can work with predefined templates or accept custom template configurations directly. Once the images are uploaded, they are temporarily saved, processed through the application's image processing pipeline, and the extracted results are sent back as a structured JSON response.

A major strength of this setup is its modular and user-friendly design. The server can be started and stopped independently, with internal logic to manage it in a separate thread. This prevents it from interfering with other parts of the application, especially the GUI, which may be running concurrently. The asynchronous nature of file handling using `aiofiles` ensures that large uploads or multiple requests don't cause slowdowns.

From a practical setup perspective, once FastAPI is installed and the server class is initialized in the code, the server can be started with a method call, which internally launches Uvicorn, a lightweight ASGI server that powers FastAPI. Developers or testers can then access the API by sending POST requests to the defined endpoint using tools like Postman or any HTTP client. The clear separation between the API interface and the core processing logic also means that the system is easy to maintain, debug, and expand upon.

5.3 Settings and Configuration

This section outlines the technical settings, packaging methods, and deployment configuration used in the development and delivery of the EXTSCAN+ desktop application, which is the main application for illustrating the image processing pipeline using LLM.

5.3.1 Installation and Setup Steps

The installation process for the application differs based on the user type, developers involved in further development and testing, and end users utilizing the packaged software. Due to the nature of this project being developed under sponsorship from **GDEX Berhad**, public access to the source code and software is currently restricted (as of 9th May 2025).

These instructions are intended for end users who will install and use the compiled desktop application.

1. Download the Installer or Executable File

The application installer can be provided directly to stakeholders via secure links such as:

- <https://github.com/wthislifehuh/extscan/releases/tag/v1.0.0> (*Access restricted to authorized parties*)
- https://drive.google.com/drive/folders/1su_c9mBQujQSCkbRRq8_yJiGQLyPaSdW (*Access restricted to authorized parties*)

2. Run the Installer (.exe File)

- Double-click the .exe file to initiate the installation process (extscan_installer.exe).
- Follow the on-screen prompts to complete the installation, including selecting the installation directory and optionally creating a desktop shortcut, as illustrated in Figure 5.3.1.1, Figure 5.3.1.2, Figure 5.3.1.3, Figure 5.3.1.4, Figure 5.3.1.5, and Figure 5.3.1.6.

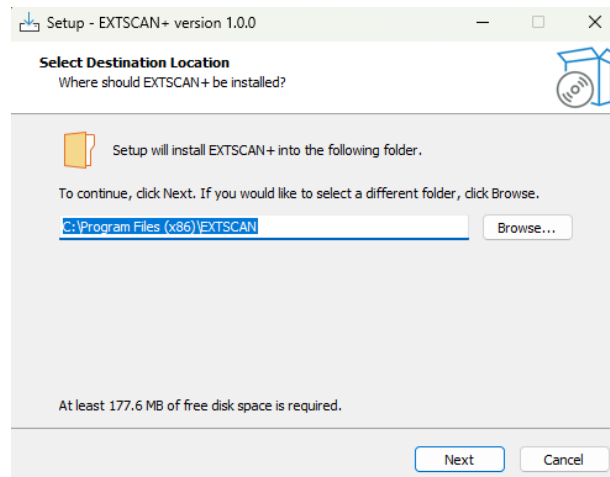


Figure 5.3.1.1 Installer Destination Location Setup

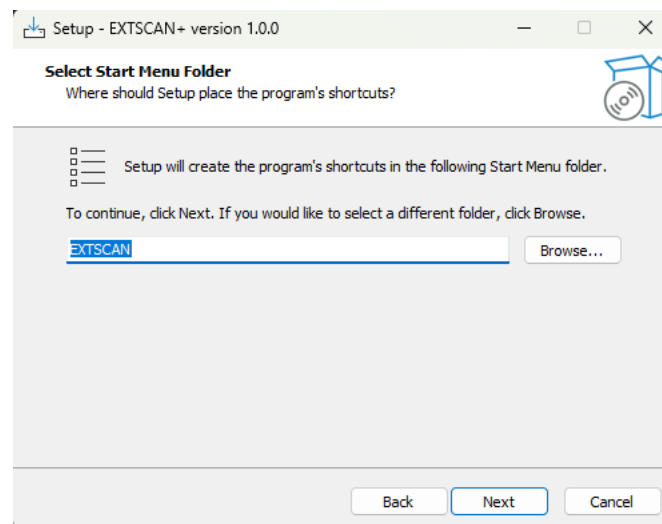


Figure 5.3.1.2 Start Menu Folder Setup

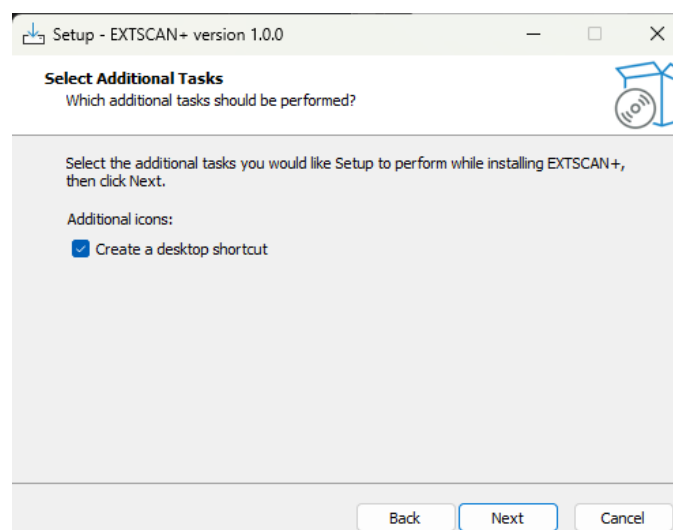


Figure 5.3.1.3 Additional Tasks Setup

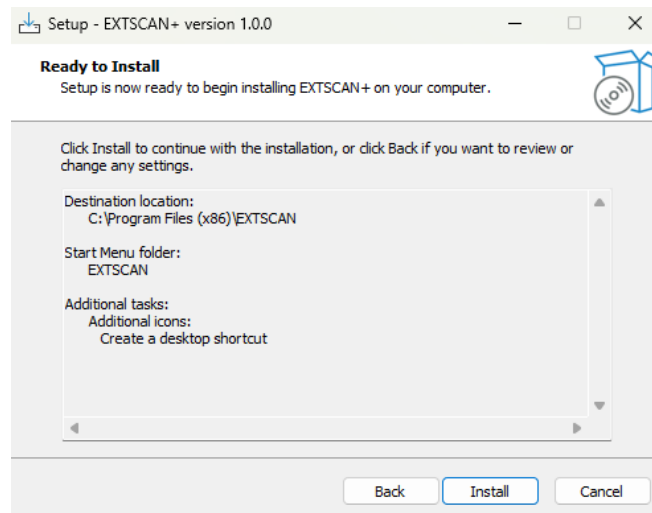


Figure 5.3.1.4 Installation Permission

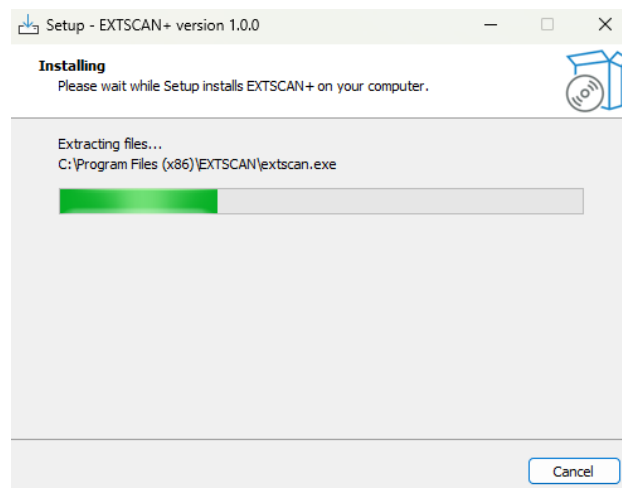


Figure 5.3.1.5 Installing Application

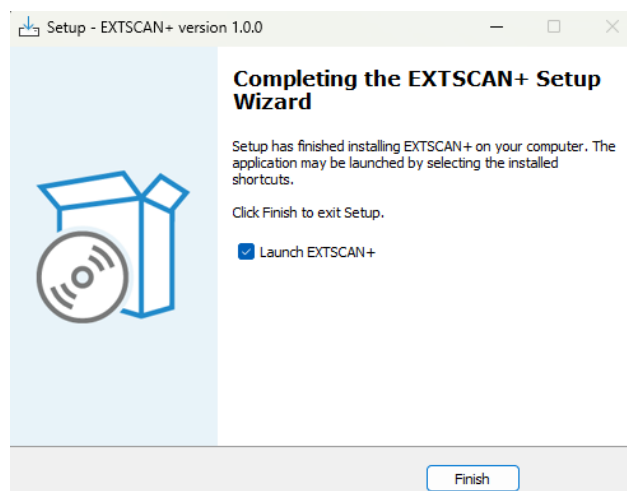


Figure 5.3.1.6 Installation Completed

3. Launch the Application

After installation, EXTSCAN+ (application of this project) can be launched:

- From the Start Menu or desktop shortcut (if created), as illustrated in Figure 5.3.1.7 and Figure 5.3.1.8.
- Or by navigating to the installed directory and running extscan.exe

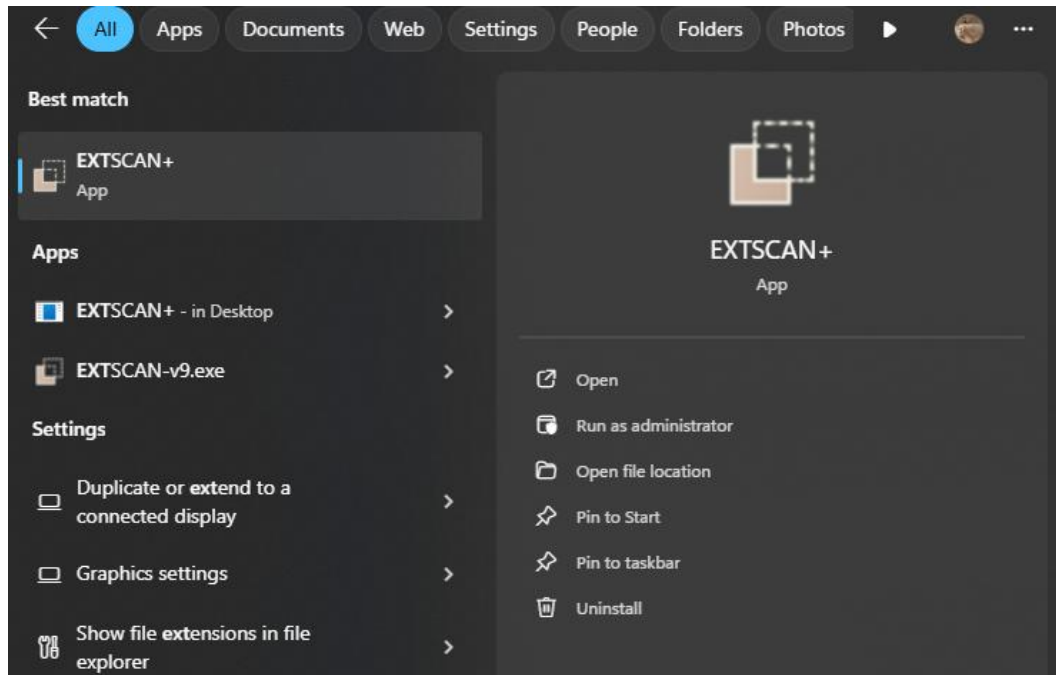


Figure 5.3.1.7 App in Start Menu

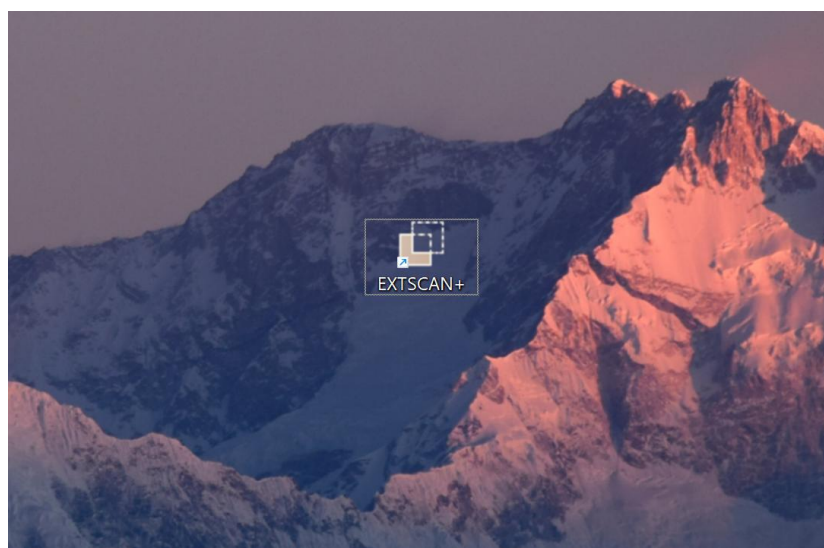


Figure 5.3.1.8 Desktop Shortcut in Window Home Page

To uninstall the application:

1. Go to Settings

- Click on Apps > Installed apps > Search “EXTSCAN”
- Click uninstall button, as shown in Figure 5.3.1.9, Figure 5.3.1.10, Figure 5.3.1.11 and Figure 5.3.1.12.

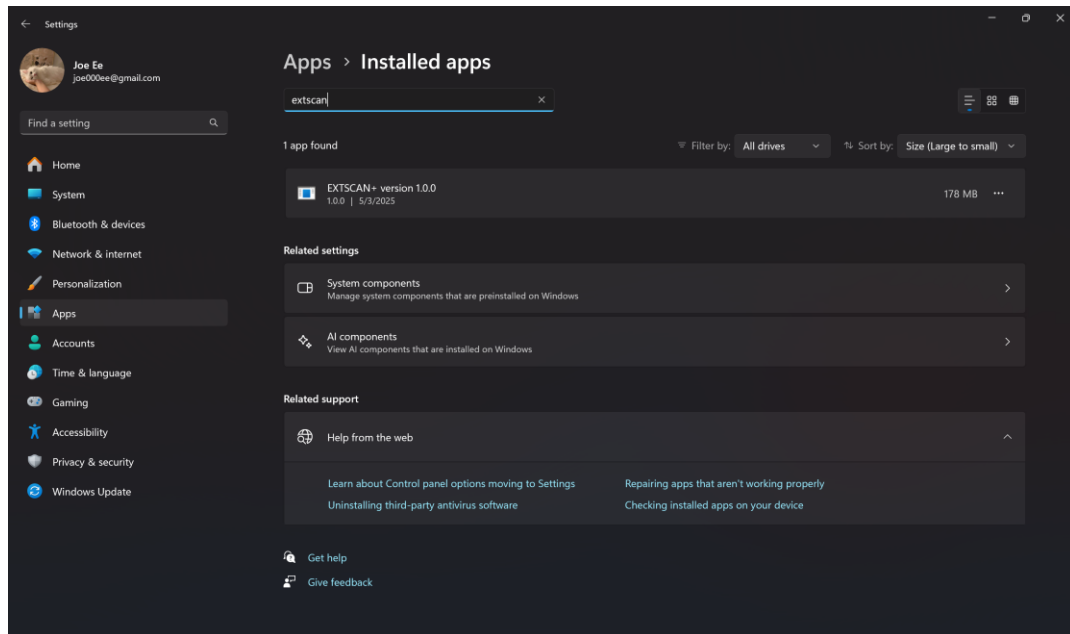


Figure 5.3.1.9 App in Installed Apps settings

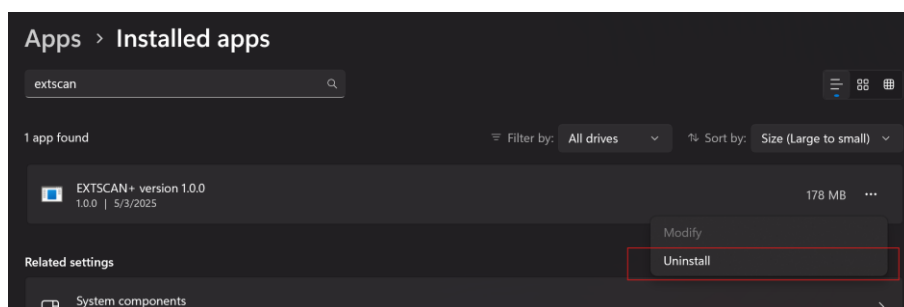


Figure 5.3.1.10 Uninstall App

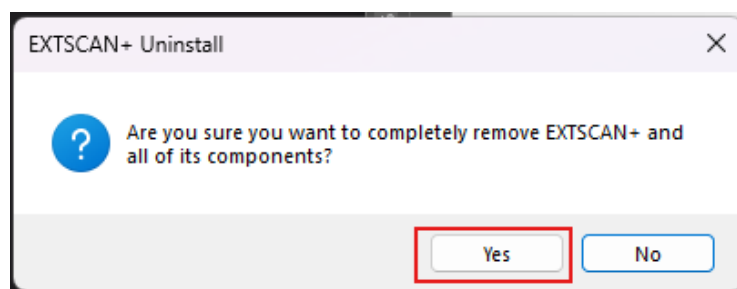


Figure 5.3.1.11 Permission dialog box to remove application

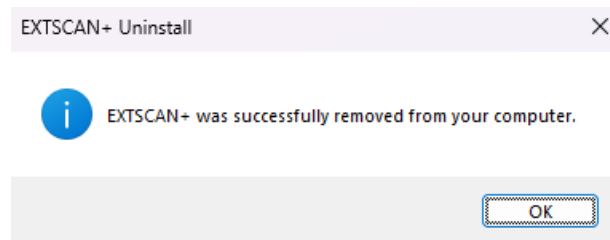


Figure 5.3.1.12 Uninstallation completed

5.3.2 Deployment Details

Packaging Tool Used

The application was packaged into a standalone executable using PyInstaller, a widely used tool for converting Python applications into distributable .exe files. The following command was used: `pyinstaller --noconfirm --clean main.spec`. This command rebuilds the application using a clean slate, with instructions from `main.spec`, and doesn't prompt for user confirmation when overwriting files. The output will be in `dist/main.exe`. This file served as the base for installer creation.

Installer Creation Tool

To provide a professional and user-friendly installation experience, **Inno Setup** was used to create a GUI-based installer. The installer includes a custom installation path, a desktop shortcut creation, embedded application icon and license file.

This configuration script (.iss) is compiled to generate an installer file (.exe). This configuration ensures that users receive a seamless installation experience with options to create desktop shortcuts and immediately run the application upon completion.

5.3.3 Distribution Method

The final .exe and installer files were distributed privately via GitHub Releases (Private Repository) and Google Drive (Restricted Access).

This project was developed in partnership with GDEX Berhad as part of a sponsored Final Year Project. As of 9th May 2025, the software is not publicly accessible and is restricted for internal evaluation and academic assessment only.

5.3.4 License

This software was developed under an academic-industry collaboration with **GDEX Berhad Sdn. Bhd.** All intellectual property, including source code, design, and branding, is owned by GDEX Berhad Sdn. Bhd. Use of the software is granted solely for academic evaluation purposes. Redistribution or reuse of this software in any form is **prohibited without written consent** from GDEX Berhad.

5.3.5 Versioning

The project follows Semantic Versioning, using the format: `MAJOR.MINOR.PATCH`

Current version: v1.0.0

This version indicates the first complete, stable release of the EXTSCAN+ application, ready for internal deployment and review.

5.4 System Operations as Graphical User Interface

This section describes the system’s operational aspects regarding the graphical user interface (GUI) design, and the functionality of various screens. It details how the interface responds to user input and how different components interact to generate the corresponding output.

5.4.1 System Navigation

The collapsible sidebar in the interface, as depicted in Figure 5.4.1.1, serves as the primary navigation mechanism, allowing users to switch between the application’s main functional areas such as “Extract Data,” “Output Documents,” “Template Manager,” “AI Model Manager,” and “System Settings.” When collapsed, the sidebar conserves valuable screen space, which is particularly beneficial on smaller displays or when users need to focus on detailed content in the central workspace, such as during data extraction or template configuration.

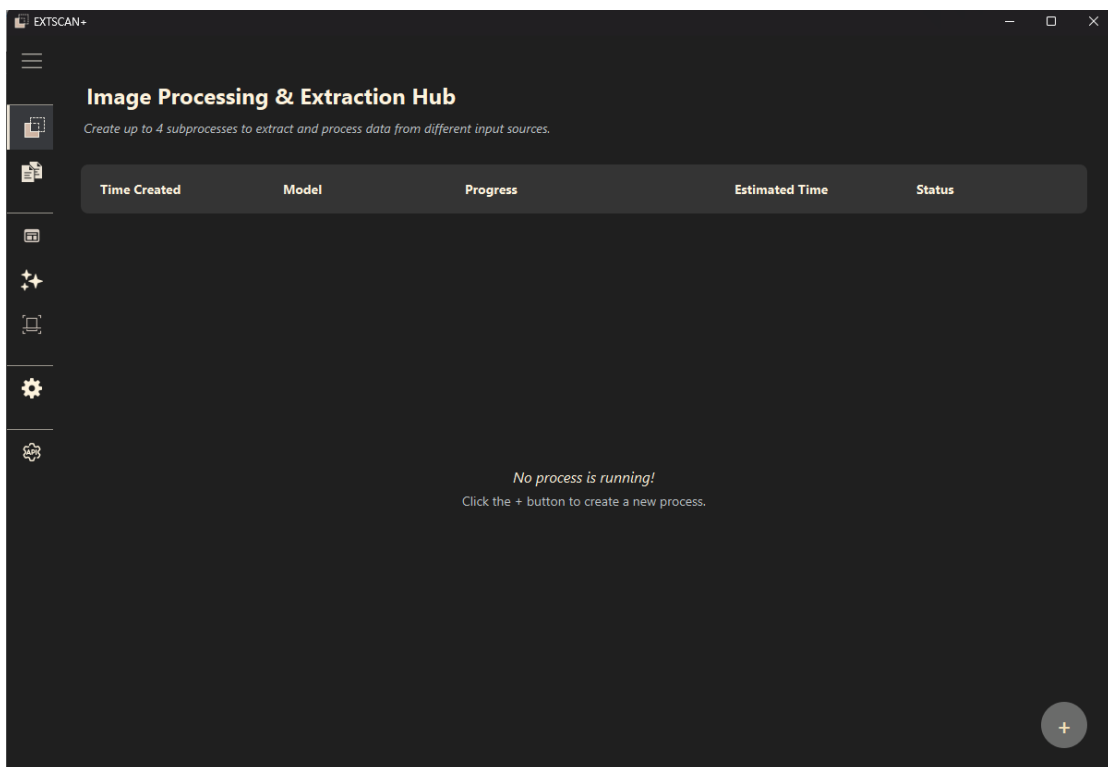


Figure 5.4.1.1 Collapsed Navigation Sidebar

5.4.2 Template Manager

The template management interface, as shown in Figure 5.4.2.1, allows users to view, edit, and organize templates efficiently. They can update ROI definitions, add or remove fields, and change metadata as needed. Templates can also be exported for backup or sharing with other users or imported into another system instance. The user interface presents a list of templates with relevant metadata like names and last edited dates, making it easy to locate and manage existing templates.

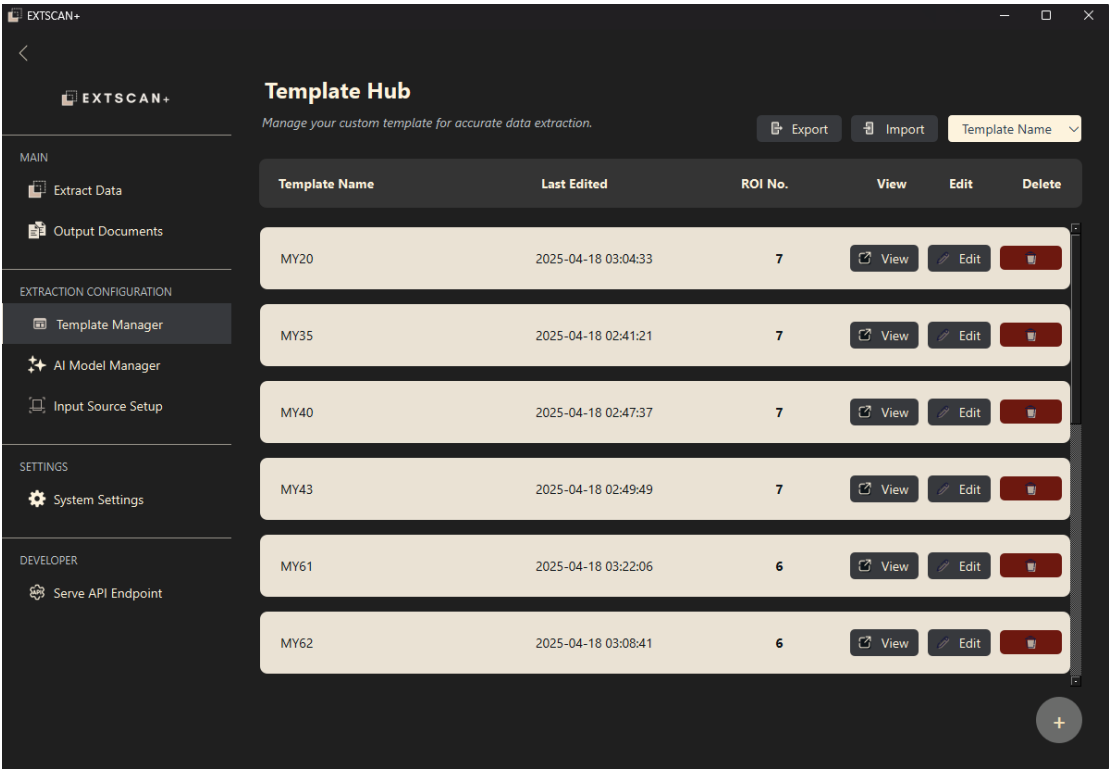


Figure 5.4.2.1 Template Manager Screen

To create a new template, user click on the “+” button at the bottom right of the screen. Figure 5.4.2.2 shows the starting point for template creation. Users input essential metadata for the template, including the template name, a short description, and the primary language of the document. Additional checkboxes let users indicate whether the template will include tables, barcodes, or checkboxes. These settings help the system prepare for more advanced extraction logic based on document structure. This information will help in LLM prompting and validation, where will be directly contributing to the overall accuracy of system.

Figure 5.4.2.2 Template Manager Screen - Create New Extraction Template

After providing the template metadata, the user is prompted to upload a sample document image, as shown in 5.4.2.3. which becomes the reference layout.

Figure 5.4.2.3 Template Manager Screen – Upload Template Image

Figure 5.4.2.4 shows an interactive interface displays the uploaded document and allows users to draw rectangular boxes over specific areas to define ROIs. Each box corresponds to a data field the user wants to extract. Users can zoom in and drag around the image for precision while defining ROIs. This visual mapping aligns the template with the actual document layout, enabling precise and structured data extraction.

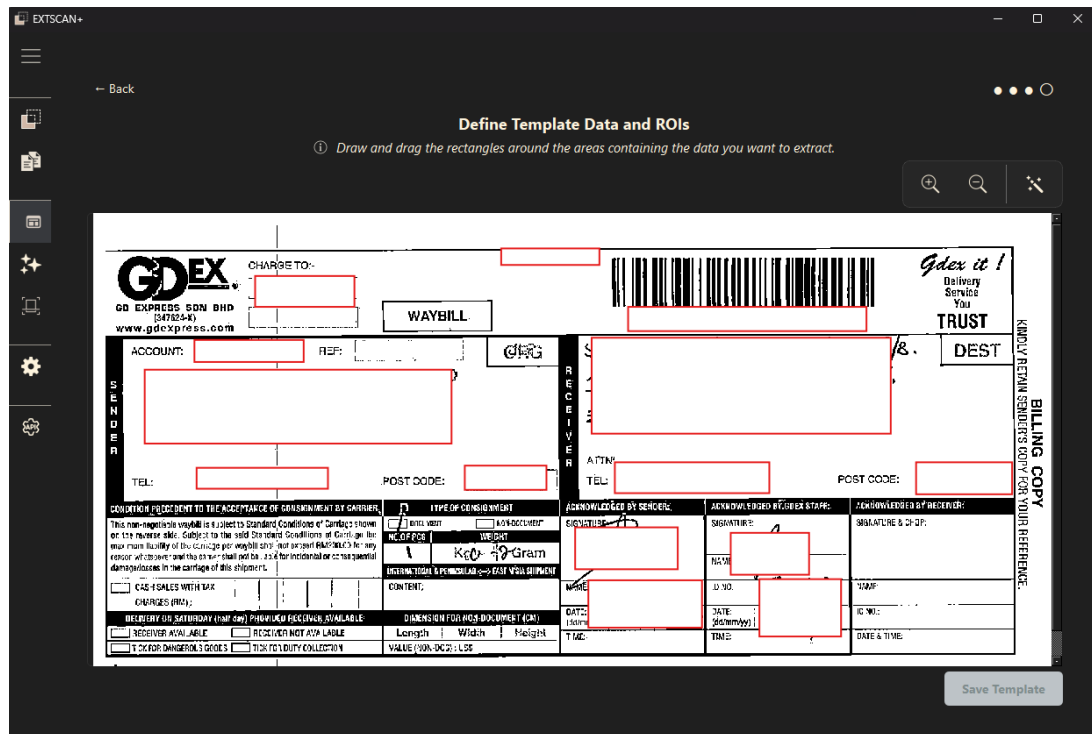


Figure 5.4.2.4 Template Manager Screen – Define Template Data and ROIs

Once an ROI is created, users define its properties in a pop-up form, as shown in Figure 5.4.2.5. These fields serve as guideline for LLM prompting and validation in Sieve Methodology. Fields include:

- **Field Key:** A unique identifier (no spaces allowed).
- **Data Type:** Such as string, number, or date.
- **Input Field Type:** Optional classification (e.g., barcode, checkbox). This is important for regex check in Sieve methodology.
- **Validation Rules:** Constraints like character count.
- **Description and Example:** Optional notes to clarify field purpose.

This metadata enhances both validation and downstream processing accuracy. The user can choose to create more fields within the same ROI. For example, user can create postcode and country, which can be inferred from address, in the address ROI.

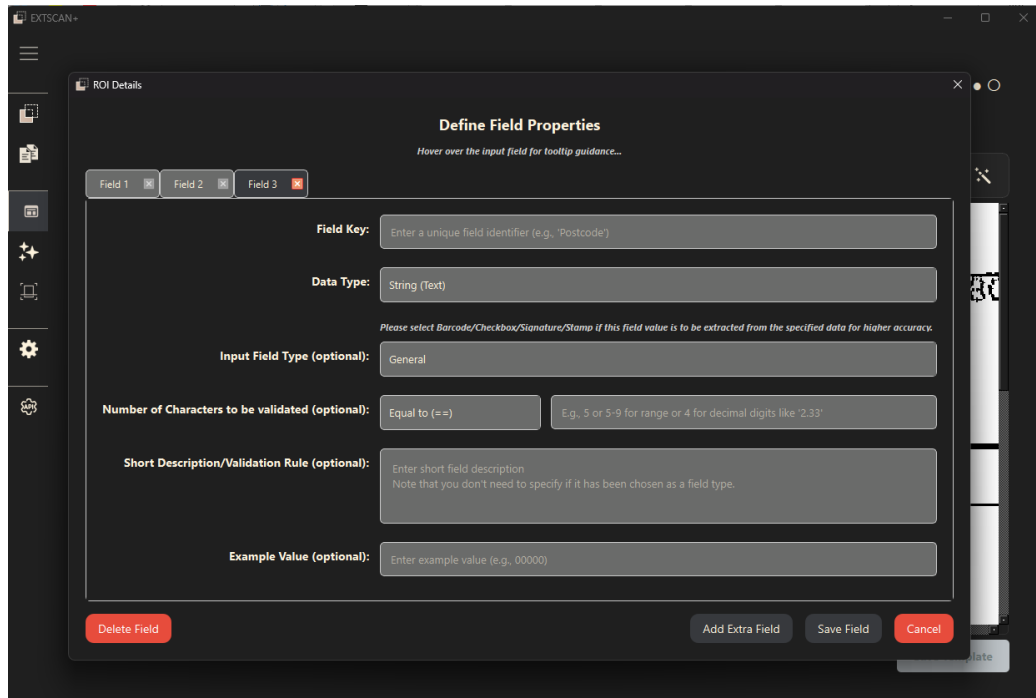


Figure 5.4.2.5 Template Manager Screen – Define Field Properties

If users input an invalid field key, such as one with spaces, they are immediately alerted, as showcased in Figure 5.4.2.6.

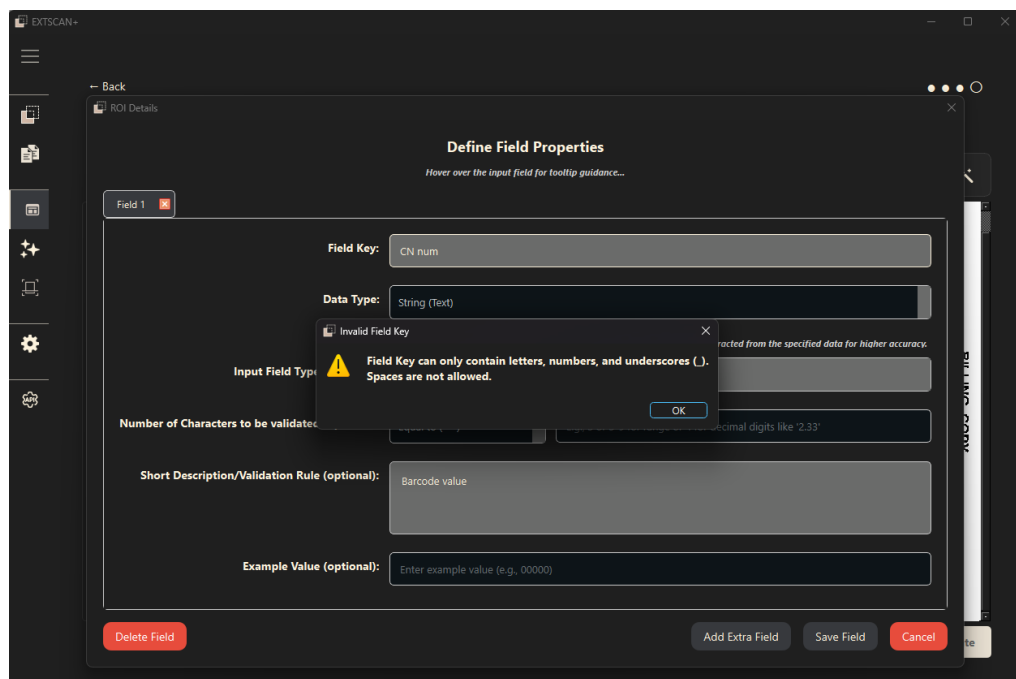


Figure 5.4.2.6 Template Manager Screen – Error Message

Figure 5.4.2.7 shows a document image with ROIs fully defined and labelled. Edit icons (pencil-like icon) allow users to modify or delete any ROI. This final review step ensures all required fields are covered before saving the template.

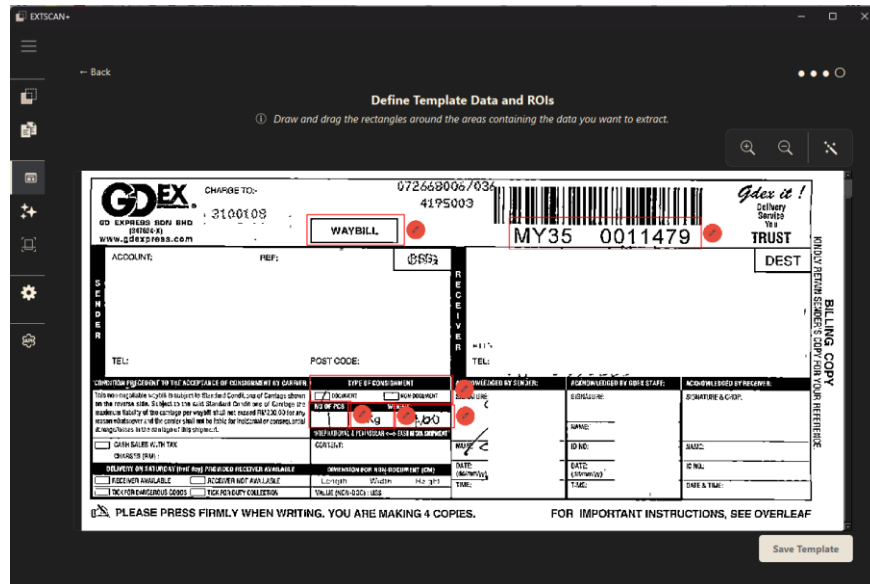


Figure 5.4.2.7 Template Manager Screen – Defined ROIs and Edit Buttons

For efficiency, the system supports automatic template generation. Here, OCR and a large language model work together to extract text and intelligently identify ROIs without user input. This reduces setup time significantly and is especially useful for recurring or standardized forms. An example is shown in Figure 5.4.2.8, note that sensitive data has been covered by red box.

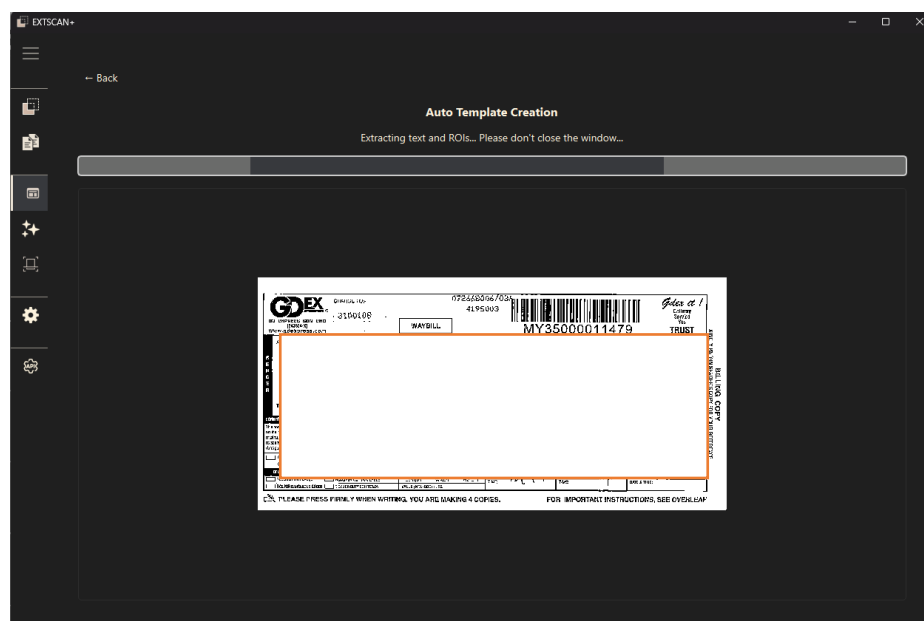


Figure 5.4.2.8 Template Manager Screen – Auto Template Creation

The last screen in Figure 5.4.2.9 confirms that the template has been saved and is ready for use. It shows a visual preview of the mapped ROIs and offers navigation options: return to the template list or proceed directly to data extraction using the new template.

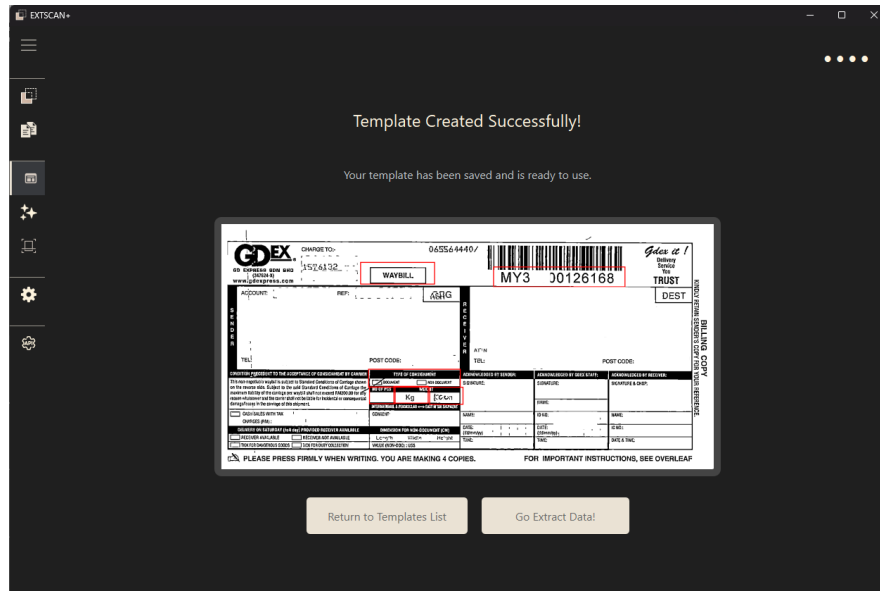


Figure 5.4.2.9 Template Manager Screen – Successfully Created Template

5.4.3 AI Model Manager

The screen shown in Figure 5.4.3.1 is displayed when no AI models have been added to the system. It shows an empty model list and a red warning message prompting the user to add a model and API key before proceeding.

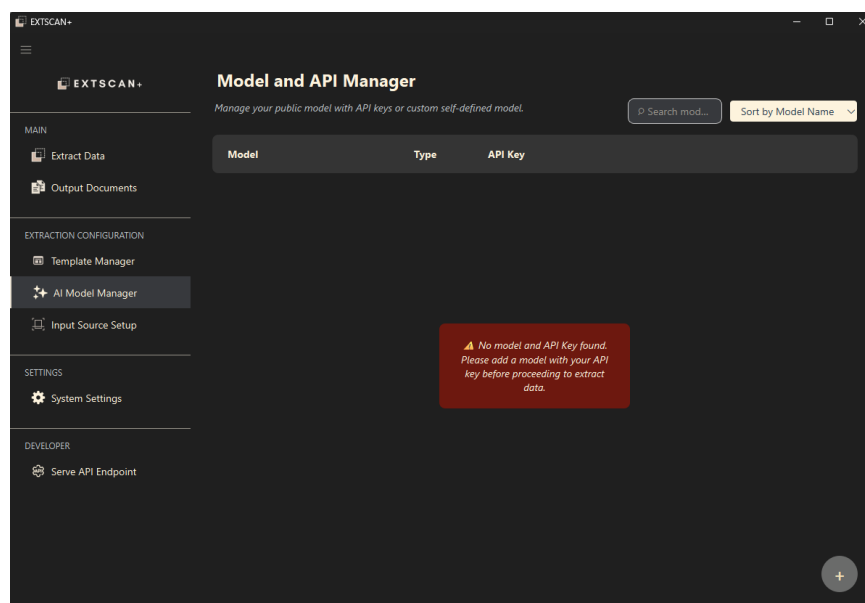


Figure 5.4.3.1 AI Model Manager Screen – No API Key found

Once models are added, the Model and API Manager screen lists them in a clean, scrollable format, such as in Figure 5.4.3.2. Each model entry displays its name, type (“Public” or “Custom”), and a hidden API key. The three-dot menu (toggle menu) next to each model allows users to interact with that model via options such as View API Key, Copy API Key, Edit, or Delete. This design promotes organized management of multiple models and offers quick actions for maintenance.

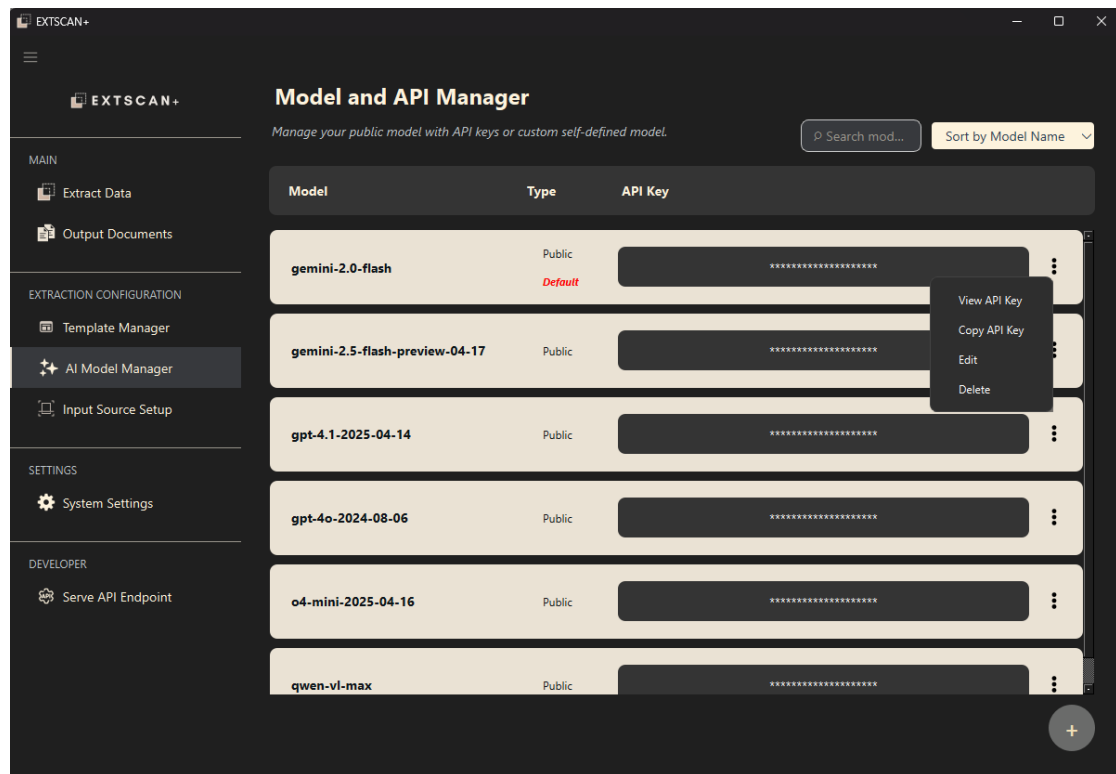


Figure 5.4.3.2 AI Model Manager Screen – Toggle Menu Option

This screen in Figure 5.4.3.3. appears when the user clicks the “+” button to add a new model and chooses the “Public Model” tab. The interface allows users to select a pre-listed public model (e.g., GPT-4.1 or Gemini), enter an associated API key, and optionally set it as the default model. The default model is used for extraction if no other model is specified in a particular template or extraction task. Other model configuration for extraction task, for example, choice of model for fallback setting, can be managed through system settings.

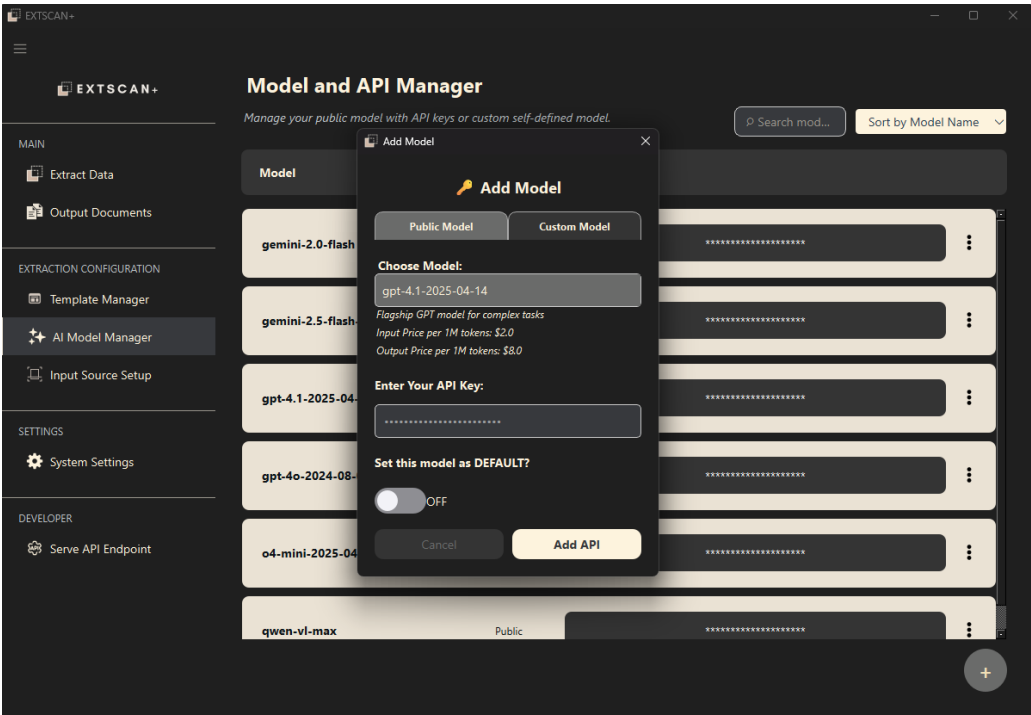


Figure 5.4.3.3 AI Model Manager Screen – Add Public Model

Figure 5.4.3.4 displays a screen that is for users who want to add a Custom Model, such as a private model hosted on Hugging Face, OpenRouter, or a proprietary endpoint. It includes fields for the model’s name, provider name, custom API URL, and an API key. Like the public model setup, users can also set a custom model as the default.

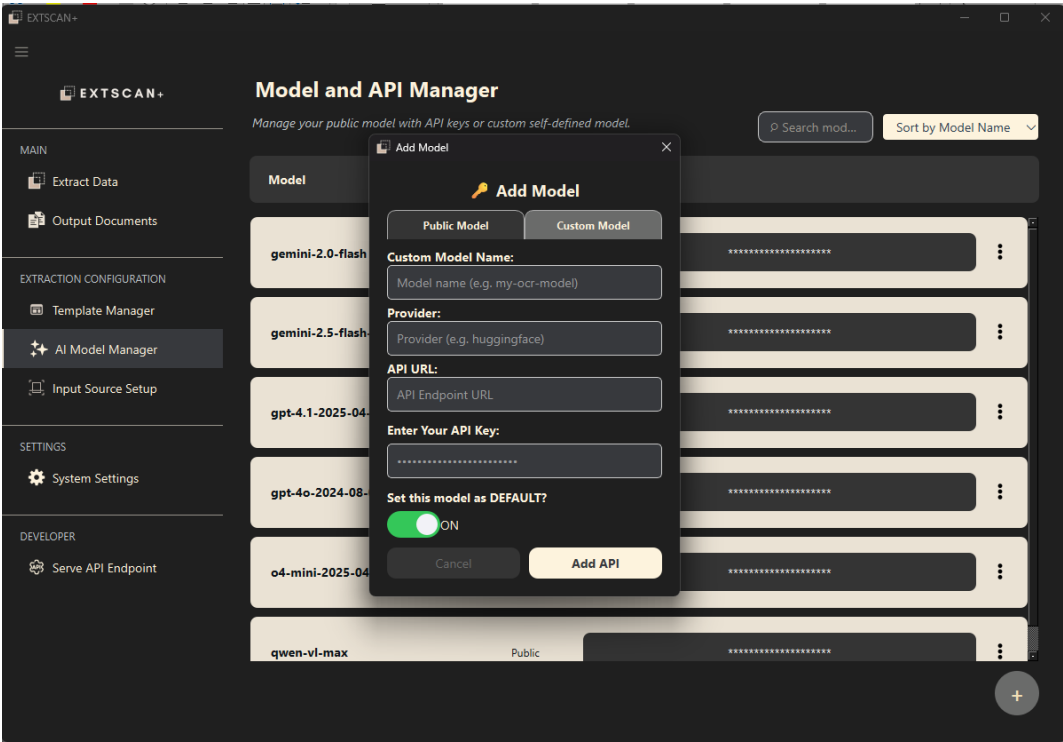


Figure 5.4.3.4 AI Model Manager Screen – Add Custom Model

5.4.4 Input Source Setup

The screen in Figure 5.4.4.1 serves as the central hub for managing all image input sources and scanner devices. It allows users to add, configure, and manage multiple types of sources such as Local Folder, Google Drive, OneDrive, Dropbox, Webcam, Scanner Device, and OSS Document Scanners (via WebDAV). When no input sources are configured, the list is empty and users are prompted to click the “+” button to add a new source. A dropdown menu appears, listing all supported source types.

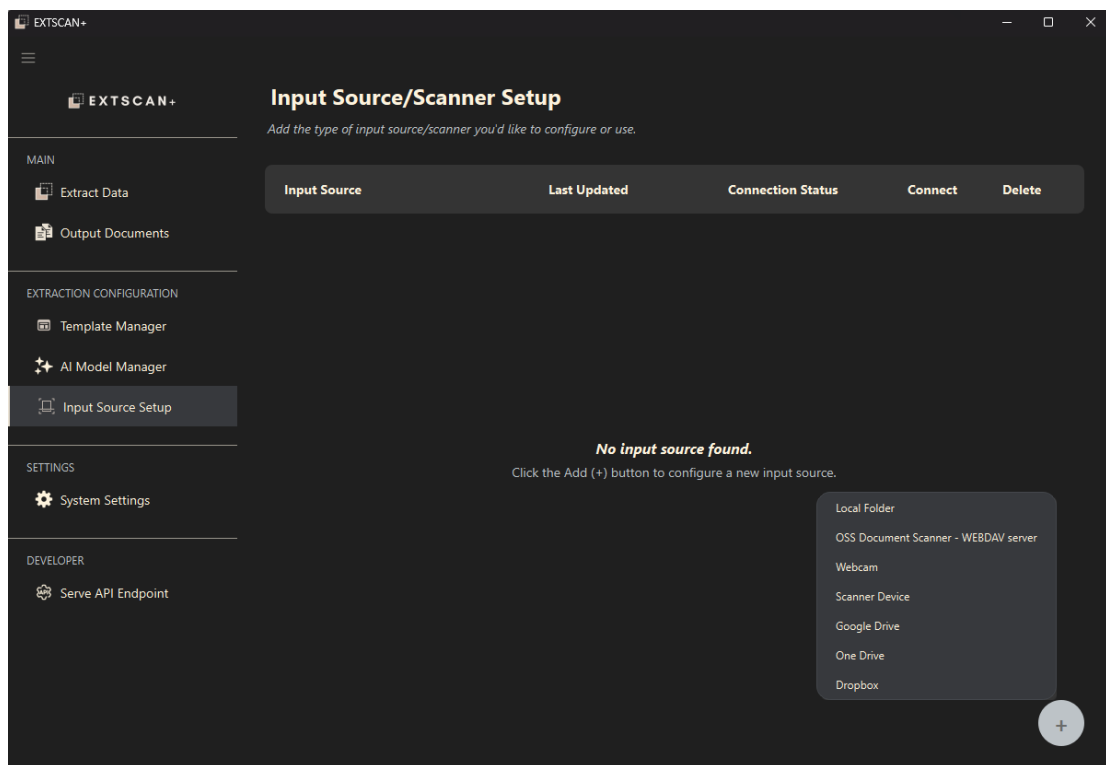


Figure 5.4.4.1 Input Source Setup Screen

When a user selects a source type such as OSS Document Scanner, the screen transitions to a configuration form specific to that input type, just as in Figure 5.4.4.2. In this case, users are instructed to configure a connection by entering details such as Host URL, Username, Password, and Remote Directory. Additional instructional text provides step-by-step guidance, including how to set up the source and sync scanned documents with the system.

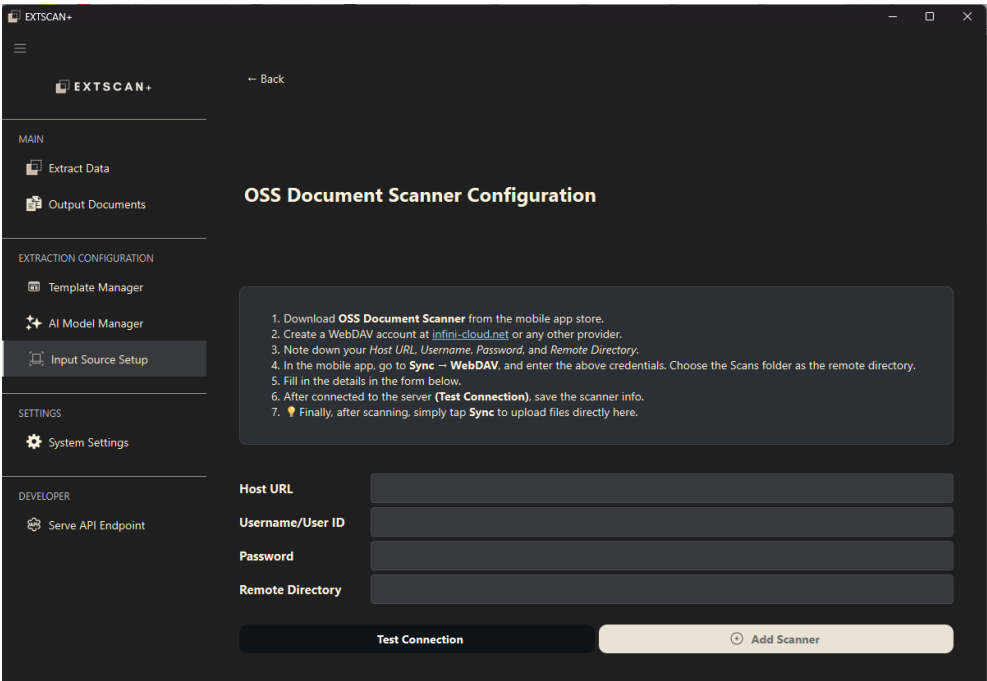


Figure 5.4.4.2 Input Source Setup Screen – Example Configuration

Once input sources have been added, they are listed in a table format showing their name (e.g., “Google Storage”, “OSS Scanner”), last updated timestamp, connection status, and available actions. Example screen is shown in Figure 5.4.4.3. Users can click “**Connect**” to activate a source or “**Delete**” to remove it from the system. This overview provides users with immediate insight into which sources are available and which ones need attention (e.g., disconnected status).

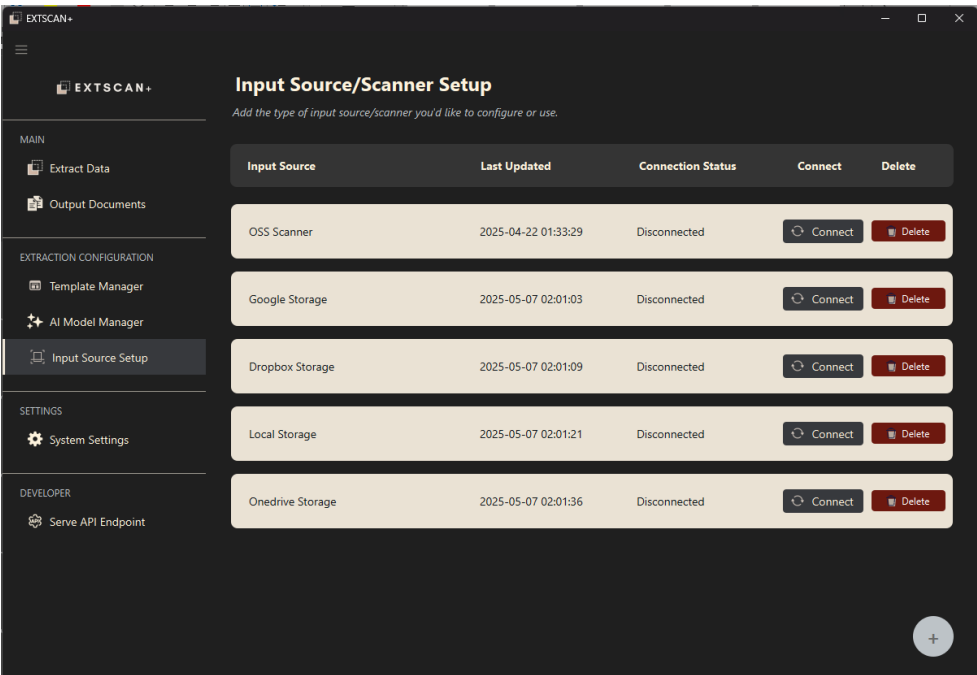


Figure 5.4.4.3 Input Source Setup Screen – Populated Source

5.4.5 System Settings

The System Settings Screen as in Figure 5.4.5.1 provides users with centralized control over key system behaviours related to data extraction, template handling, model selection, and logging. They can manage the default auto-template matching mode, either is content-based which uses OCR and LLM for template detection, or barcode-based which is more suitable for large volume of input with barcode value. They can also configure the fallback behaviour of the system when no template is match, and enable LLM Challenger and Referee mode. This screen is primarily used to configure how the system behaves when processing documents, ensuring consistent operation aligned with user preferences or organizational policies.

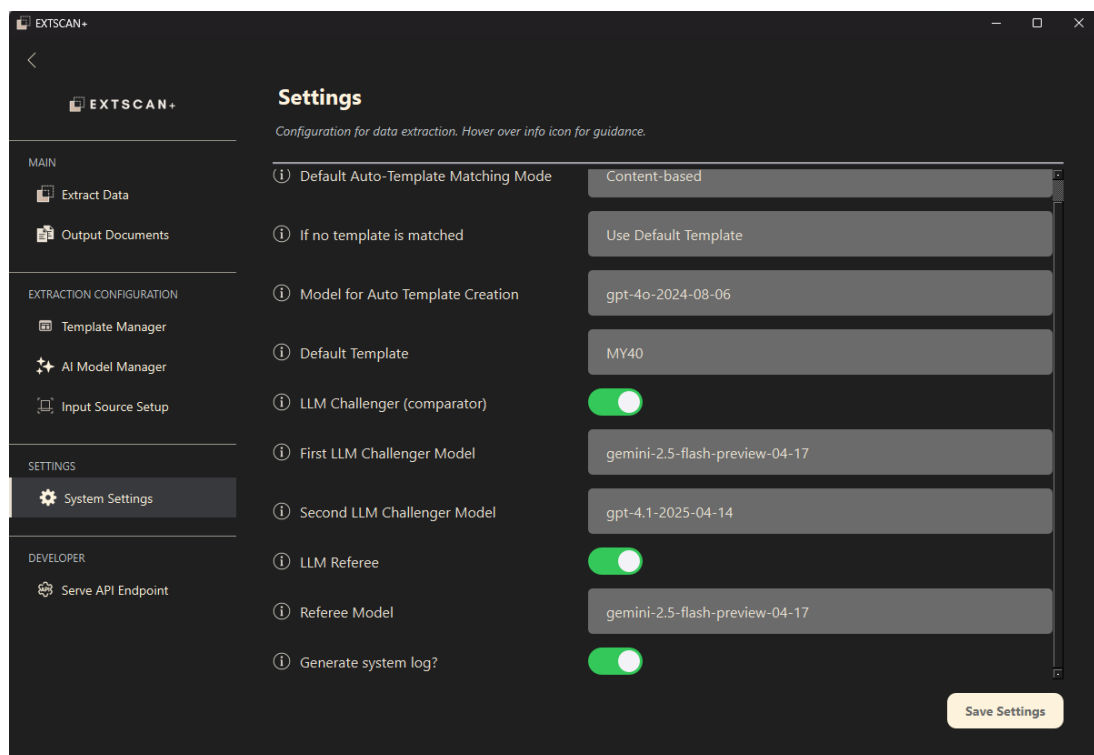


Figure 5.4.5.1 System Settings Screen

5.4.6 Data Processing

Figure 5.4.6.1 shows the central dashboard for managing all ongoing or queued data extraction tasks. Each process is listed with details such as time created, the AI model selected, current progress, estimated time to completion, and overall status. Users can view the process details or delete a process. The interface supports parallel processing of up to four subprocesses, allowing efficient batch operations across multiple document sets.

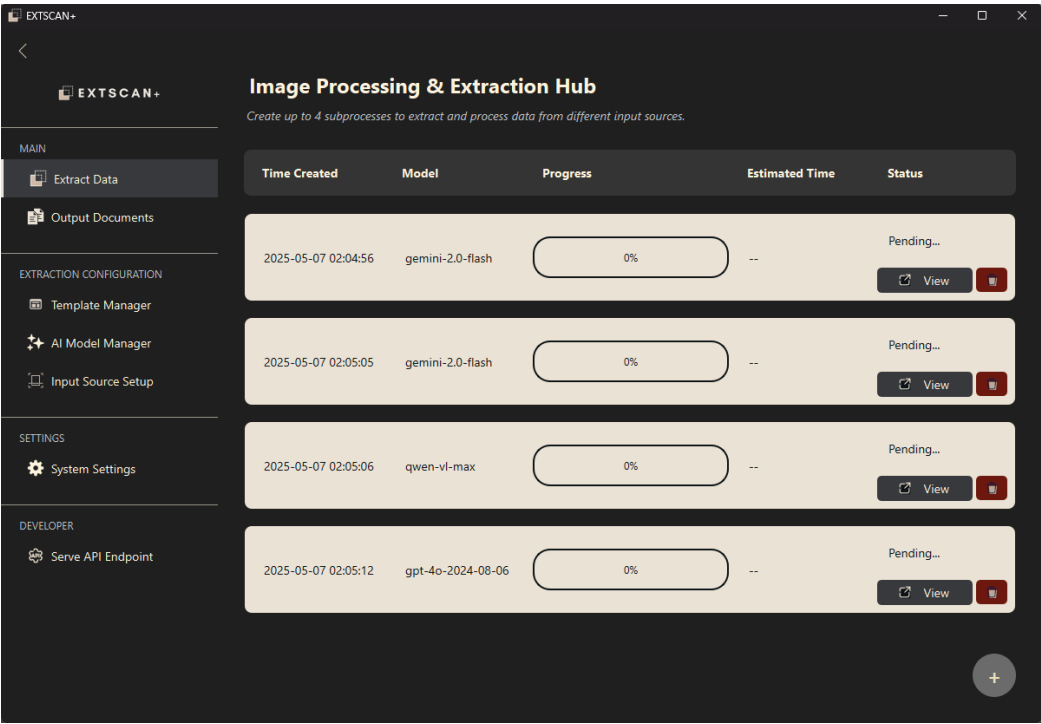


Figure 5.4.6.1 Image Processing and Extraction Hub

In Figure 5.4.6.2, a screen is displayed to allow users to initiate a new data extraction process. Images can be uploaded via drag-and-drop, and users configure the extraction by selecting an AI model (e.g., gemini-2.0-flash), choosing a template (manual or Auto), and choosing an output format (e.g., Excel .xlsx). The interface also shows warnings for low-quality images, readiness status, and estimated processing time.

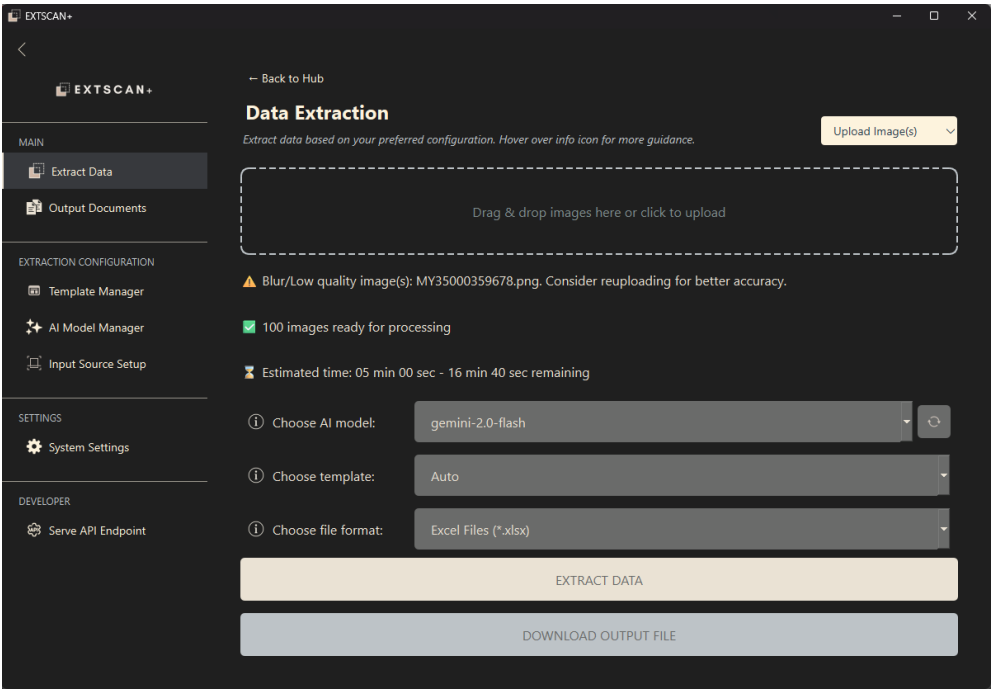


Figure 5.4.6.2 Data Extraction Screen

Once the extraction begins, the screen shown in Figure 5.4.6.3 displays real-time progress of the task. It shows how many images have been processed so far, a progress bar with percentage completed and estimated time remaining for the batch.

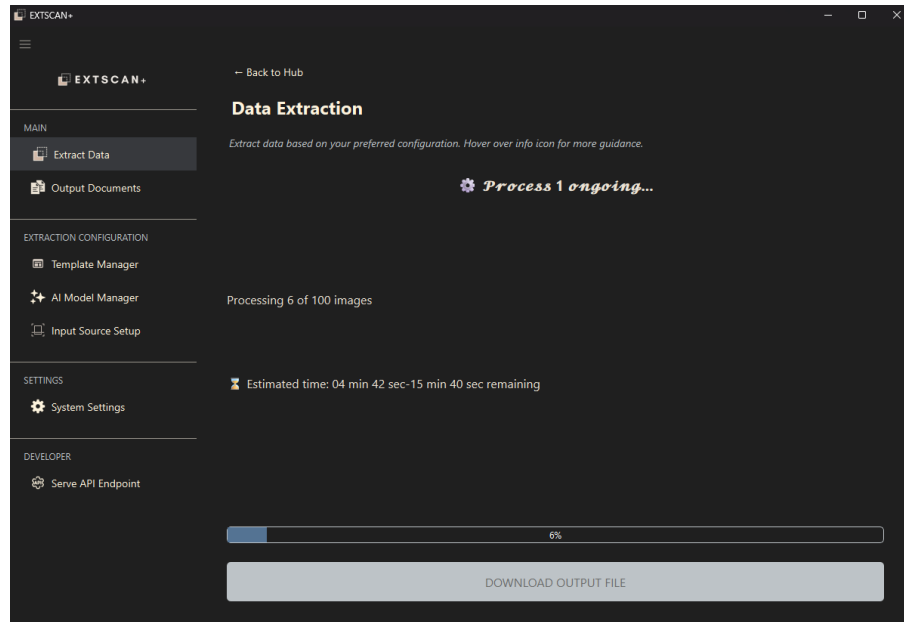


Figure 5.4.6.3 Data Extraction Screen – Start Processing

Once extraction is complete, this result screen depicted in Figure 5.4.6.4 shows a summary of the process, including the number of images processed, total processing time and the model used for extraction. A toggle labeled “Show Extracted Data” allows users to view the actual data within the interface.

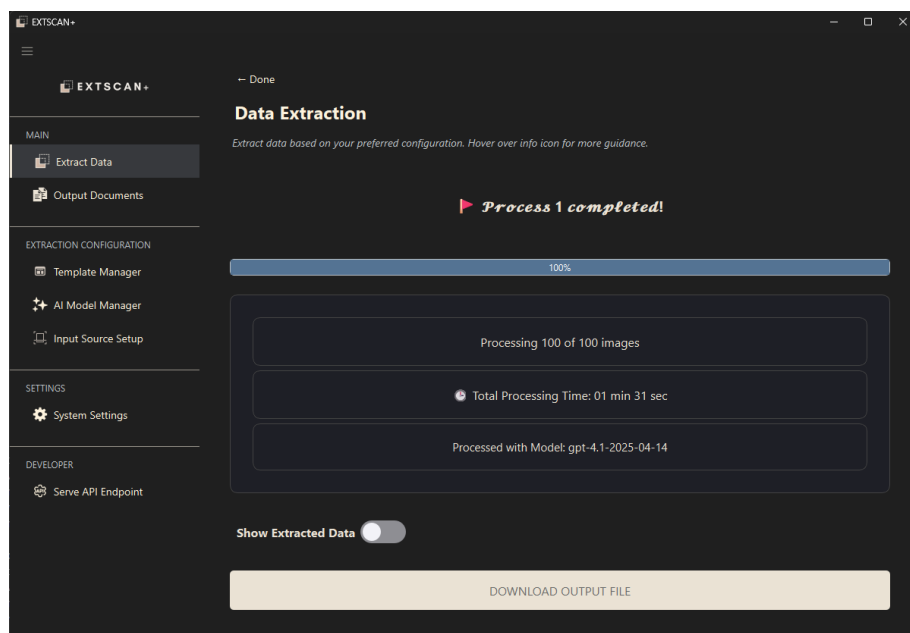


Figure 5.4.6.4 Data Extraction Screen – Result Screen

When the Show Extracted Data toggle is enabled, a table displays the extracted content in a structured format. An example is shown in Figure 5.4.6.6. Each row corresponds to a processed document, and columns represent extracted fields. This preview helps users verify results before downloading the final output file.

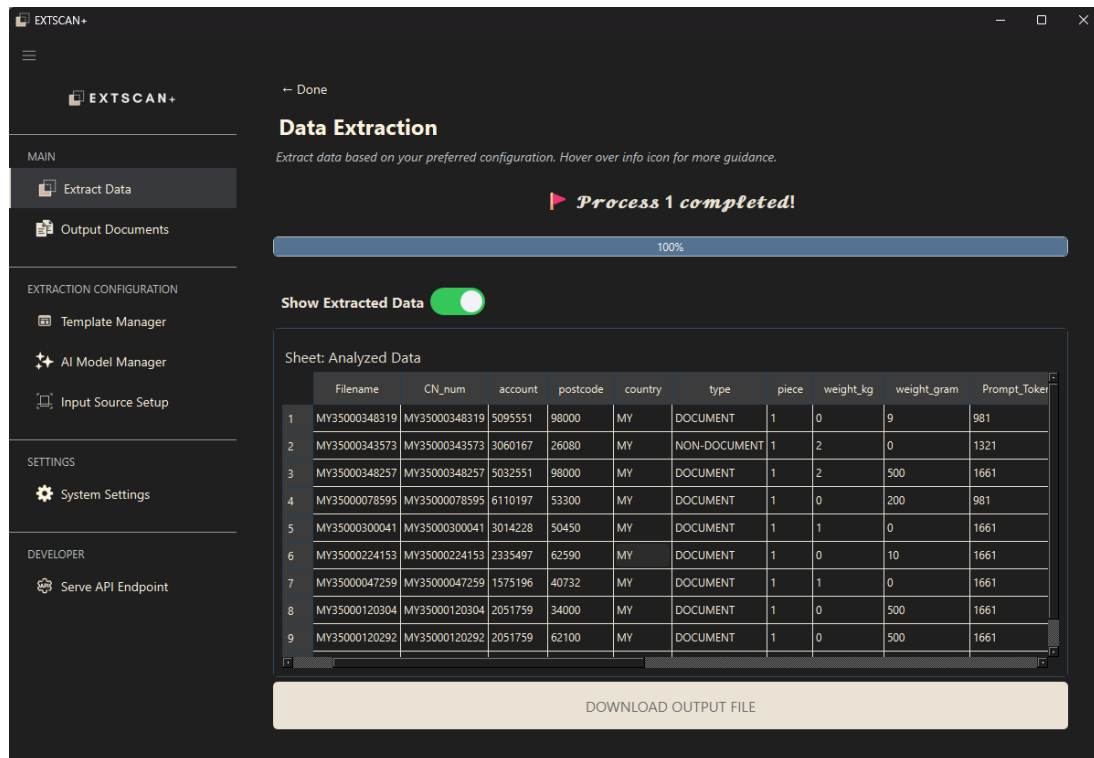


Figure 5.4.6.5 Data Extraction Screen – Display Output

This screen returns to the central hub after processes have completed, just as in Figure 5.4.6.7. It now shows completed processes (with 100% progress) and any leftovers (unprocessed files). Users can resume interrupted jobs or review completed ones using the “Resume” and “View” buttons respectively. The hub simplifies task management by giving users control over pending, completed, or failed extractions.

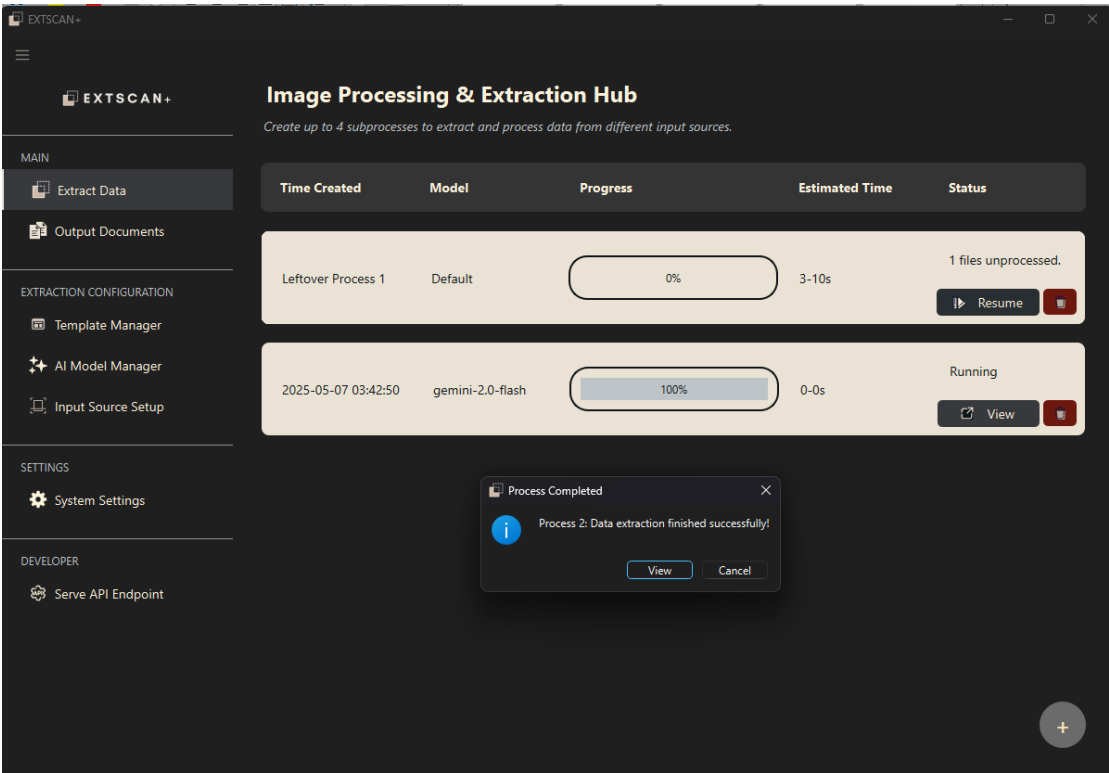


Figure 5.4.6.6 Image Processing and Extraction Hub – Manage Process

5.4.7 Past Output Documents

The Output Documents section of the system provides a streamlined interface for managing files generated from the data extraction process. It allows users to view, search, sort, download, and delete past output files. The system supports multiple file formats such as XLSX, JSON, CSV, XML, TXT, and MD. As shown in Figure 5.4.7.1, the interface populates a table with metadata including the file name, type, timestamp, and actions for downloading or deleting. There are also filter dropdown for user to filter the file by latest file creation time, file type and file name. These features improve navigability, especially when dealing with extensive archives of output files.

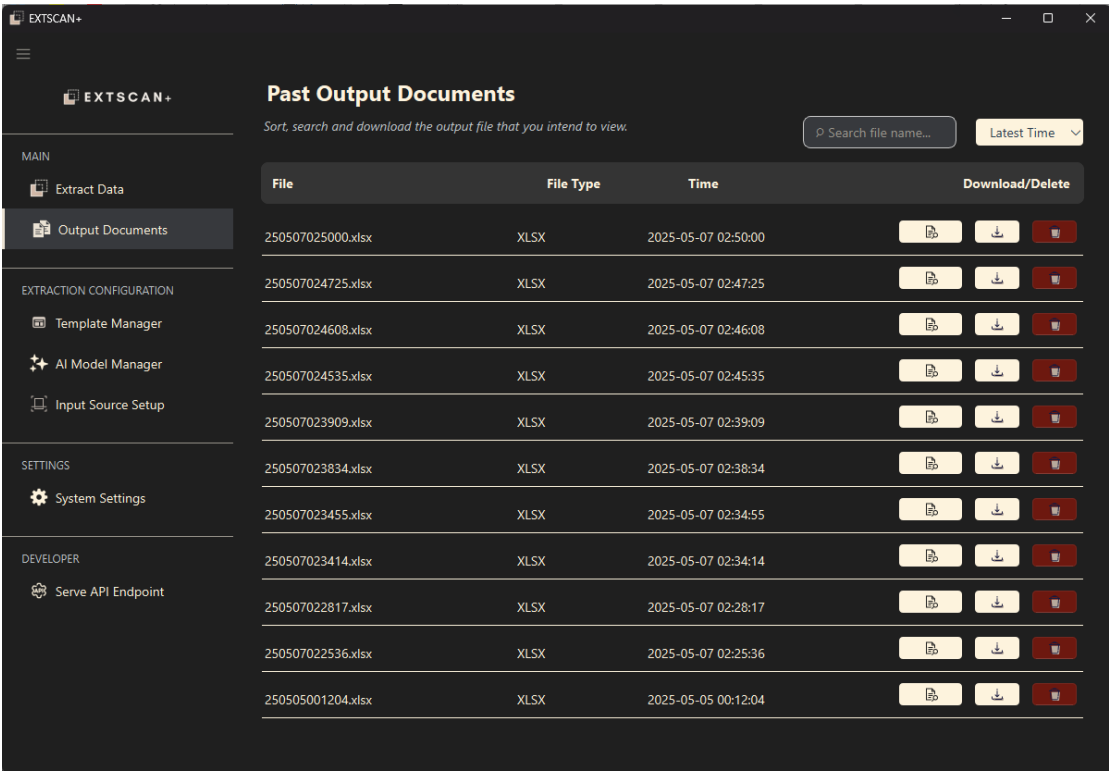


Figure 5.4.7.1 Files populated in Output Documents Screen

In Figure 5.4.7.2, the search bar is used to filter files by filename. Matching files are displayed instantly.

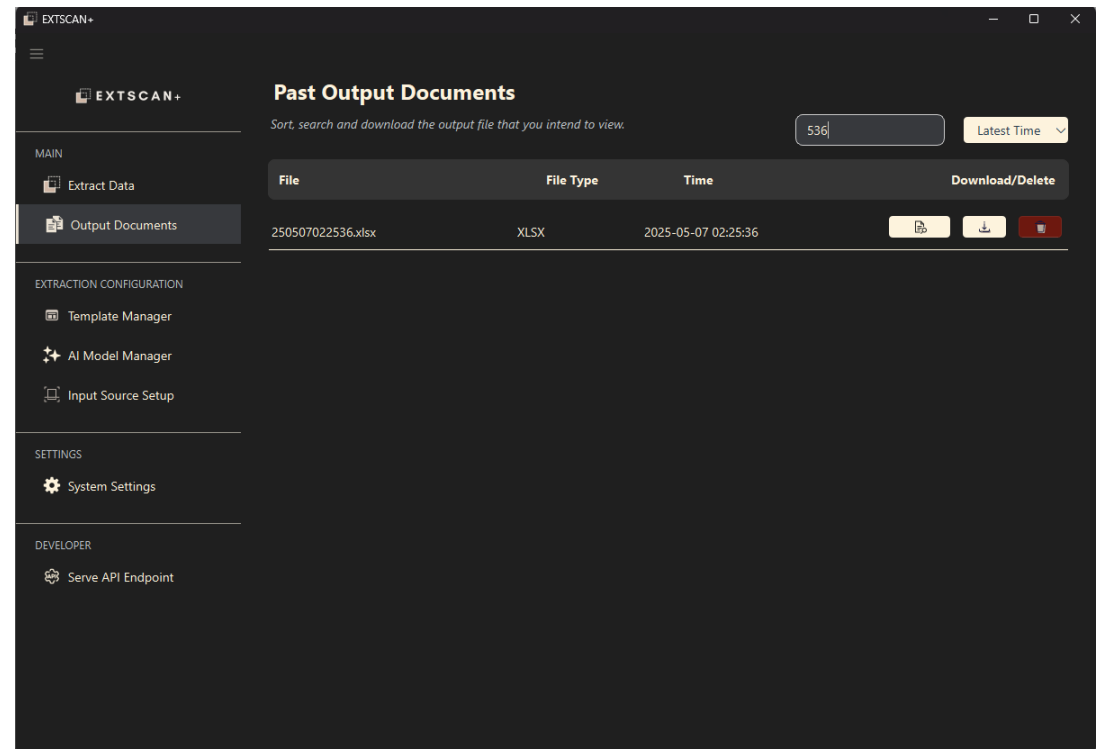


Figure 5.4.7.2 Past Output Document Screen – File Search

When user click on view file button, the content of the file will be displayed in table form, as shown in Figure 5.4.7.3.

The screenshot shows the EXTSCAN+ application window. On the left is a sidebar with navigation options: MAIN (Extract Data, Output Documents), EXTRACTION CONFIGURATION (Template Manager, AI Model Manager, Input Source Setup), SETTINGS (System Settings), and DEVELOPER (Serve API Endpoint). The main area displays a file named '250507023909.xlsx' with a 'Back to Documents' link. Below the file name, it says 'Sheet: Analyzed Data'. A table with 10 rows and 10 columns is shown. The columns are: Index, CN_num, account, postcode, country, type, piece, weight_kg, weight_gram, and Prompt_Tokens. The table contains 10 rows of data, with the last row being a non-document.

	CN_num	account	postcode	country	type	piece	weight_kg	weight_gram	Prompt_Tokens
1	MY35000348319	5095551	98000	MY	DOCUMENT	1	0	9	981
2	MY35000343573	3060167	26080	MY	NON-DOCUMENT	1	2	0	1321
3	MY35000348257	5032551	98000	MY	DOCUMENT	1	2	500	1661
4	MY35000078595	6110197	53300	MY	DOCUMENT	1	0	200	981
5	MY35000300041	3014228	50450	MY	DOCUMENT	1	1	0	1661
6	MY35000224153	2335497	62590	MY	DOCUMENT	1	0	10	1661
7	MY35000047259	1575196	40732	MY	DOCUMENT	1	1	0	1661
8	MY35000120304	2051759	34000	MY	DOCUMENT	1	0	500	1661
9	MY35000120292	2051759	62100	MY	DOCUMENT	1	0	500	1661
10	MY35000111863	2313432	31100	MY	NON-DOCUMENT	1	1	0	1661

Figure 5.4.7.3 Past Output Document Screen – View File Content

5.4.8 System Output as Subsequent Input of Data Pipeline (API)

The API Endpoint screen in this application enables the system to function as a service-oriented component within a broader data pipeline. This interface allows users to start and stop an internal API server that listens for HTTP requests and responds with processed data extracted from images. This functionality is crucial for integrating this application into automated workflows, where output from the system can be used as input for other services or data processing stages, such as data validation, storage, analytics, or visualization platforms.

The guideline, example request, example response format is displayed in the screen, as shown in Figure 5.4.8.1 and Figure 5.4.8.2. The system also displays the local IP (Internet Protocol) address and port (typically `http://192.168.x.x:8000`) through which the API is accessible for easier establishment of API connection.

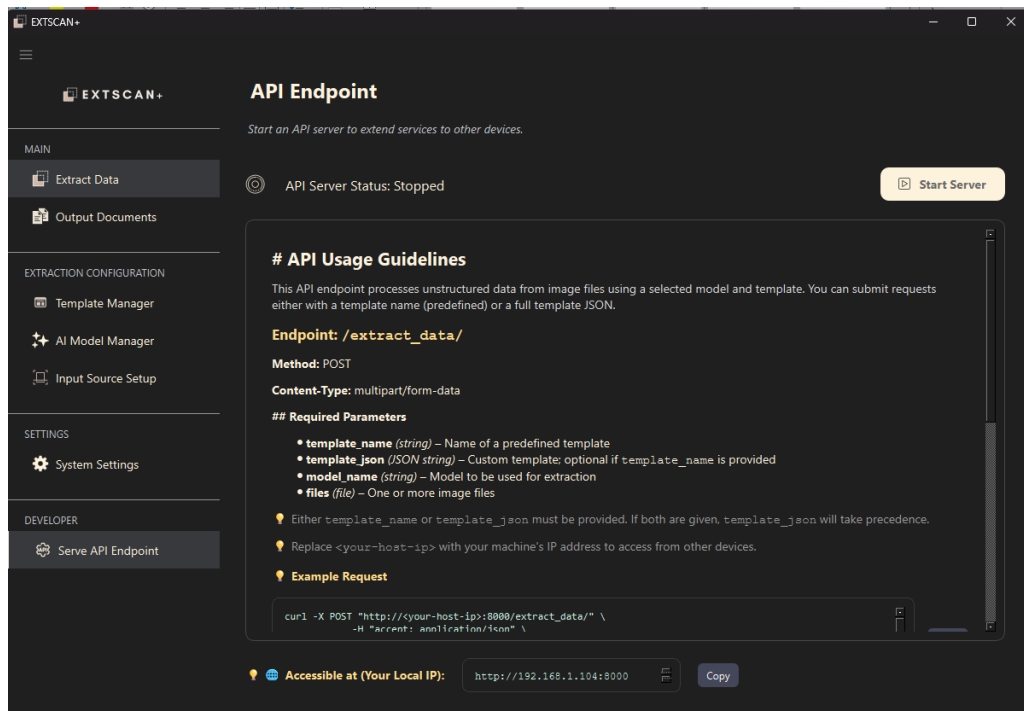


Figure 5.4.8.1 API Endpoint Screen – Guideline Part I

The “How to Access from Other Devices” section in the API Endpoint screen as illustrated in Figure 5.4.8.2 provides instructions for enabling external devices on the same local network to interact with the API. To make the API accessible beyond the local network, for example, to allow cloud services, remote developers, or external systems to send data, additional configuration is required.

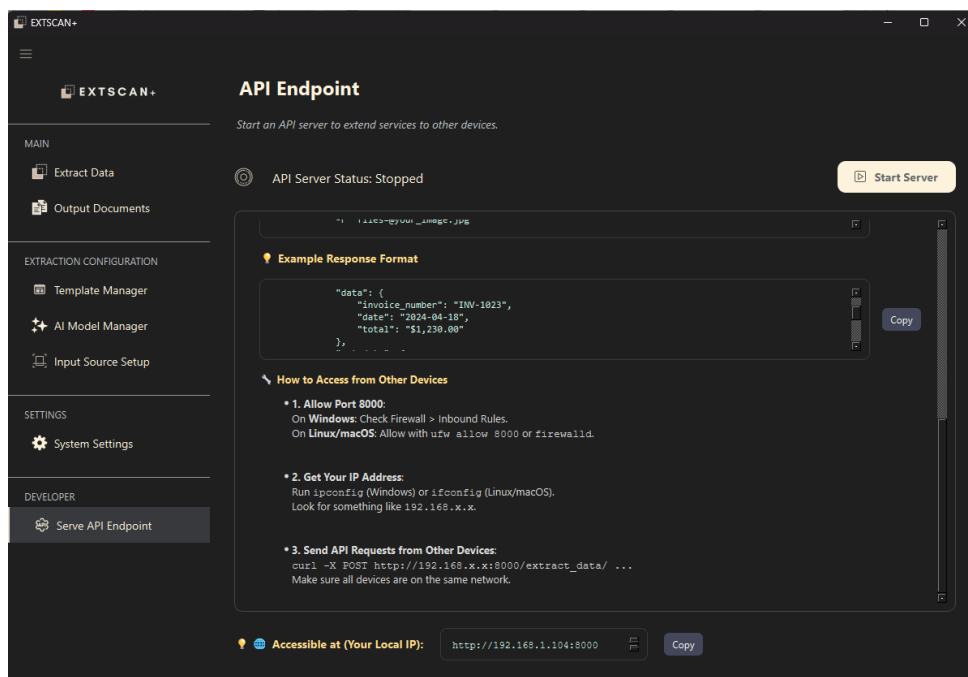


Figure 5.4.8.2 API Endpoint Screen – Guideline Part II

To use the API Endpoint feature, the user first initiates the server by clicking “Start Server”. Once active, the system will show a success message to inform the user. The user can then submit a POST request to the /extract_data/ endpoint with multipart/form-data that includes a selected AI model, a predefined or custom template, and one or more image files. The server then processes the images using the specified parameters and returns the structured output in JSON format, which can be automatically forwarded or consumed by another component in the pipeline.

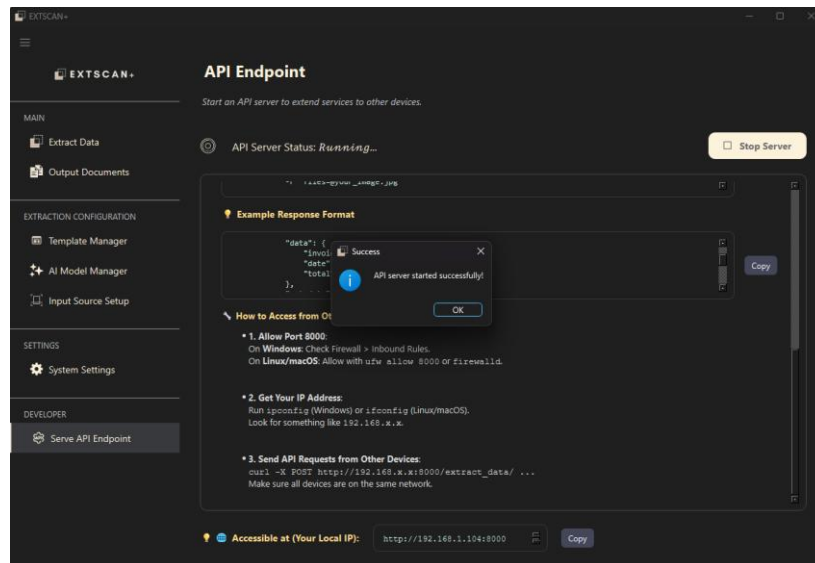


Figure 5.4.8.3 Start API Endpoint Connection

When the server is no longer needed, it can be cleanly shut down using the “Stop Server” button, as shown in Figure 5.4.8.4.

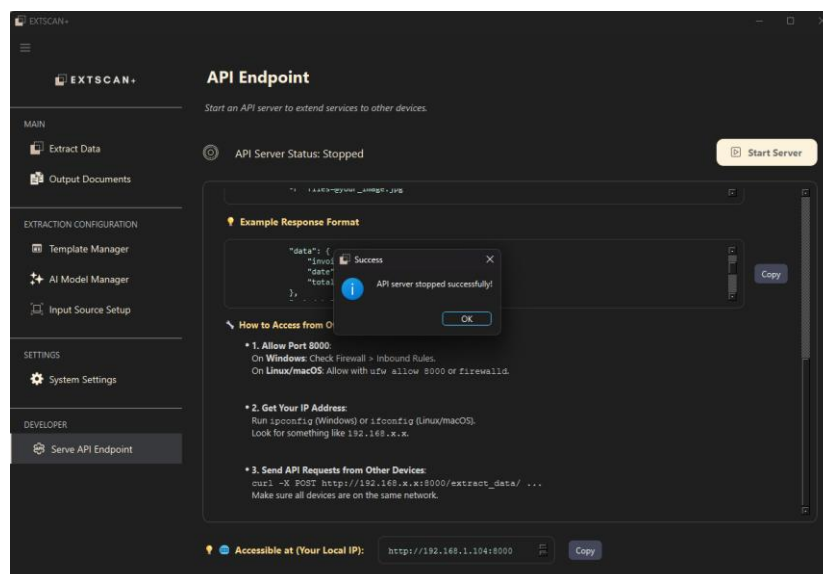


Figure 5.4.8.4 Stop API Server

5.5 Implementation Issues and Challenges

One of the core technical challenges in this project lies in efficiently managing concurrency through threads and asynchronous operations. The system is designed to handle multiple image input sources simultaneously while remaining responsive through a GUI and managing different threads for different image processing processes. This requires various background threads to monitor sources and process image in real-time. However, concurrency can introduce complications such as race conditions, thread contention, and data corruption, especially when shared resources like output files or logs are accessed concurrently. To mitigate this, thread locks are employed, but their improper use can lead to bottlenecks or deadlocks, potentially degrading performance and responsiveness.

Another significant challenge is maintaining high data accuracy and reliability, particularly when working with noisy or low-quality document images. Input images often vary in lighting, skew, resolution, or contain artifacts such as handwriting, stamps, or shadows. These inconsistencies can significantly affect the results of OCR and downstream LLM-based analysis. Moreover, due to the layered nature of the pipeline, where each component (preprocessing, template matching, prompt generation, etc.) influences the final outcome, evaluating the true accuracy against ground truth becomes complex. Even small tweaks to a single stage can ripple across the system, altering outputs and making it difficult to isolate which part of the pipeline caused an improvement or degradation in performance. This presents a challenge for testing and validation, as continuous tuning is required without guaranteed reproducibility of results.

Finally, building a data extraction pipeline that is flexible enough to support diverse document types and dynamic input is inherently complex. Different forms, such as invoices, waybills, and forms, have varied layouts, field structures, and content types. Ensuring this flexibility without overcomplicating the architecture demands a modular design, where components can be extended or swapped with minimal disruption. However, this extensibility must be balanced with maintaining consistency in output format and behavior, which can be particularly challenging as new document types or data structures are introduced into the system.

5.6 Concluding Remark

The system implementation outlined in this chapter reflects a well-structured and thoughtfully engineered solution, meticulously developed to meet the project's performance, usability, and integration goals. From hardware provisioning and software architecture to modular API design and GUI responsiveness, the implementation demonstrates a strong alignment with practical deployment considerations. Key strengths include the extensibility of the template and model management systems, the seamless setup of diverse input sources, and the intelligent use of multimodal LLMs for robust data extraction. The integration of image preprocessing, concurrent task handling, and structured data output has positioned the system as both flexible and production ready. While implementation challenges, particularly around concurrency and input variability, posed complexity, these were met with sound design practices and iterative refinement.

All in all, this implementation serves as a solid foundation for enterprise-level automation in document processing. It demonstrates the viability of combining classical software engineering principles with advanced AI models to deliver a user-friendly, high-performing, and scalable solution tailored for real-world use cases.

CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION

In this chapter, we present a comprehensive evaluation of the system, detailing the verification and validation strategies employed. It covers functional and non-functional testing, presents a general discussion on how the system performed during testing, highlighting major achievements, benchmarks, and areas of concern. The chapter culminates in an overall performance evaluation and a discussion of project challenges. This section

6.1 System Verification and Validation Strategy Overview

For this project, a system developed to automate structured data extraction from document images, verification and validation were carried out. Verification focuses on ensuring the system is developed according to specifications, while validation confirms that the final product fulfils its intended purpose and meets user needs. This system processes sensitive and variable data, making accuracy, reliability, and usability essential. The chosen V&V strategy ensures both technical soundness and real-world applicability of the solution.

The verification and validation approach for this project comprises a comprehensive mix of:

- **Functional Testing:** To ensure the core features work as intended under various conditions. It includes use case testing and input handling testing.
- **Non-Functional Testing:** To assess performance, reliability, usability, and effectiveness, especially for image processing and data extraction accuracy.
- **Acceptance Testing:** To validate the system against user expectations and real-world usage scenarios involving diverse document types and formats.

These validation efforts confirmed that this project delivers meaningful value and performs reliably in its target environments.

6.2 System Performance Metrics

The system performance for this project will be defined by a set of key metrics that target improvements in areas such as accuracy, processing speed, efficiency, and usability. These metrics shown in Table 3.3.4.1 ensure that the system meets the operational needs of the users and performs reliably in various use cases, including high-volume processing scenarios. This project has been tested for **deployment** at **GDEX Berhad** from **February 2025**, during which user feedback will be collected to evaluate the system's performance.

Metrics	Measurement	Goal
Data Extraction Accuracy The system shall achieve a data extraction accuracy of at least 95% for all supported data types, including handwritten text, barcodes, and metadata.	Accuracy will be measured by comparing 200 extracted data against a ground truth dataset (provided by GDEX sdn bhd), assessing the percentage of correctly extracted information (e.g., correct text recognition for postcode, correct barcode values).	Continuous improvement to exceed 98% accuracy by preprocessing techniques and incorporating user feedback for error correction.
Single Image Processing Time The system shall process individual images in less than 5 seconds, from upload to final data output, including all preprocessing, quality checks, and data extraction steps.	Time will be measured from the moment an image is uploaded until the processed data is available for download or passed to the next module.	Reduce the processing time to 3 seconds per image by optimizing algorithms, improving server-side processing power, and streamlining data flow.
Batch Processing Speed The system shall process a batch of 1,000 images within 10 minutes, ensuring consistent performance even under high load.	Total processing time will be measured from the start of the batch upload to the completion of data extraction and output for the entire batch.	Enhance parallel processing capabilities and optimize the distribution of workload across servers to achieve a target of processing 1,000 images in under 8 minutes
Token Usage in LLM Operations The system shall minimize token usage during LLM operations, keeping the average token count	Token usage will be monitored per text extraction task, with adjustments made to the processing algorithms to stay within target limits.	Further optimize LLM operations to reduce token usage by 10-15%, contributing to cost savings in processing.

REFERENCES

per processed text extraction under 1,000 tokens.		
Error Handling and Recovery The system shall automatically detect and recover from errors, with less than 1% of processing tasks requiring manual intervention.	The frequency and type of errors encountered during processing will be tracked, with a focus on automated recovery procedures.	Enhance error detection algorithms and recovery protocols to reduce manual intervention to below 0.5%.
User Interface Responsiveness The web application interface shall respond to user inputs (e.g., image uploads, data extraction criteria selection) in under 1 second.	Interface responsiveness will be tested across different devices and network conditions, ensuring consistent performance.	Optimize front-end performance to reduce latency, achieving a target response time of under 0.5 seconds for most user actions.

Table 6.2.1 System Performance Metrics

By defining and targeting these performance improvements, the system will not only meet current user requirements but also adapt to future needs, ensuring long-term usability and efficiency.

6.3 Functional Testing Setup and Result

This section presents detailed breakdown of dynamic testing procedures and outcomes. Functional testing is a type of dynamic software testing that validates the system against its functional requirements. It is used in this project to ensure that the application behaves as expected by testing each feature and function based on input and output. The goal is to confirm that the software performs the intended operations correctly and consistently. **Most of the test cases** will be performed on GDEX's CN billing copy, as in Figure 6.3.1.

Figure 6.3.1 GDEX Berhad CN Billing Copy Template

6.3.1 Use Case Testing

In this section, it depicts how use case testing was carried out to validate system behaviour against predefined use cases to ensure coverage of functional requirements. The tests cover the end-to-end system workflow, giving confidence that this project can behave correctly across its major user journeys. The use cases selected for testing are based on those defined in Chapter 3.4.

Use Case Test 01

Test Design Item	TD01 – Manage Template Testing
Related Use Case ID	UC01
Objective	To ensure all functionalities of the Template Manager work correctly , including creating, viewing, editing, deleting, importing, and

REFERENCES

		exporting. This test ensures that templates can be managed effectively for data extraction.		
Test Coverage and Outcome				
Test Case ID	Test Condition/ Data	Expected Result	Actual Result	Pass/ Fail
TC-01-001	Main Flow – Launch Template Manager screen and clicks on (+) button	System navigates to respective sections.	System navigates correctly.	Pass
TC-01-002	Alternative Flow – <i>Create Template</i> User clicks (+), fills details, uploads image, creates ROIs, saves.	Template is saved and appears in hub	Template saved and displayed	Pass
TC-01-003	Alternate Flow – <i>Edit Template</i> User clicks edit, modifies fields, and saves	Template updated and saved	Changes reflected in hub	Pass
TC-01-004	Alternate Flow – <i>Delete Template</i> User clicks delete and confirms	System deletes template	Template removed from list	Pass
TC-01-005	Alternate Flow – <i>View Template</i> User clicks view button	System displays template with sorting	Details shown with sorting options	Pass
TC-01-006	Alternate Flow – <i>Export Template</i> User clicks export, chooses location	System saves file to location	Exported file found in location	Pass
TC-01-007	Alternate Flow – <i>Import Template</i> User uploads template and image	Template imported and displayed	New template appears in hub	Pass

Table 6.3.1.1 Use Case Test 01

Use Case Test 02

Test Design Item	TD02 – Manage AI Model Testing
Related Use Case ID	UC02
Objective	To verify the system's ability to correctly create, view, edit, and delete AI models used for document processing. It confirms that AI models can be configured and managed in compliance with operational needs.
Test Coverage and Outcome	

REFERENCES

Test Case ID	Test Condition/ Data	Expected Result	Actual Result	Pass/Fail
TC-02-001	Main Flow – <i>Launch AI Model Config</i> User launches AI Model Config and clicks a button	System navigates to section	Section displayed	Pass
TC-02-002	Alternate Flow – <i>Add AI Model</i> User clicks (+), fills details, selects model, toggles default, saves	AI model is saved	Model appears in list	Pass
TC-02-003	Alternate Flow – <i>View AI Model</i> User clicks view, searches/sorts	System displays sorted/search results	Correct results shown	Pass
TC-02-004	Alternate Flow – <i>Edit AI Model</i> User edits and saves AI model	Updated AI model is saved	Changes reflected in list	Pass
TC-02-005	Alternate Flow – <i>Delete AI Model</i> User deletes a model and confirms	Model is removed	Deleted model no longer in list	Pass

Table 6.3.1.2 Use Case Test 02

Use Case Test 03

Test Design Item	TD03 – Manage Input Source Testing			
Related Use Case ID	UC02			
Objective	To validate the system’s capability to add, connect, and delete various input sources such as scanners, webcams, or cloud storage. It ensures seamless integration and availability of input sources for document ingestion.			
Test Coverage and Outcome				
Test Case ID	Test Condition/ Data	Expected Result	Actual Result	Pass/Fail
TC-03-001	Main Flow – <i>Launch Input Source Setup</i> User opens Input Source Setup	System displays Input Source Setup screen	Screen displayed	Pass
TC-03-002	Alternate Flow – <i>Add Input Source</i> User adds source, tests connection, and saves	Input source is saved	Source appears in list	Pass

REFERENCES

TC-03-003	Alternate Flow – <i>Connect Input Source</i> User clicks connect	System connects and shows status	Status shown correctly	Pass
TC-03-004	Alternate Flow – <i>Delete Input Source</i> User deletes source and confirms	Source is removed	Source no longer in list	Pass

Table 6.3.1.3 Use Case Test 03

Use Case Test 04

Test Design Item		TD04 – Process Image and Extract Data Testing		
Related Use Case ID		UC04		
Objective		To test the complete image processing and data extraction pipeline, including template and model selection, image pre-processing, data recognition, sieve approach, challenger, referee and structured output It ensures accurate and reliable data extraction from images.		
Test Coverage and Outcome				
Test Case ID	Test Condition/ Data	Expected Result	Actual Result	Pass/ Fail
TC-04-001	Main Flow – <i>Extract Data</i> User uploads image, selects model/template, clicks extract	Data extracted and displayed	Output shown with data	Pass
TC-04-002	Alternate Flow – <i>Template Not Found</i>	Prompt and button to create template	Message and button shown	Pass
TC-04-003	Alternate Flow – <i>API Model Not Found</i>	Prompt and button to create model	Message and button shown	Pass
TC-04-004	Alternate Flow <i>Low Image Quality</i> Image quality fails	Image processed by system after retry	Rectified and reprocessed	Pass
TC-04-005	Alternate Flow – <i>Low Quality Persists</i>	Image rejected	Rejected with error	Pass
TC-04-006	Alternate Flow – <i>No Template Matched</i>	System uses fallback or returns null	Fallback triggered or skipped	Pass
TC-04-007	Alternate Flow – <i>Job interrupted</i>	System retries, prompts resume	Auto retry or manual resume	Pass

Table 6.3.1.4 Use Case Test 04

Use Case Test 05

Test Design Item		TD05 – Access Past Output Documents & Results Testing		
Related Use Case ID		UC05		
Objective		To ensure users can effectively access, search, sort, download, and delete previously extracted output documents and results. This supports audit trails, result review, and data management.		
Test Coverage and Outcome				
Test Case ID	Test Condition/ Data	Expected Result	Actual Result	Pass/Fail
TC-05-001	Main Flow – <i>View Output Documents</i> User views, sorts, and downloads files	Files displayed, sorted, and downloaded	Files listed and downloadable	Pass
TC-05-002	Alternate Flow – <i>File Not Found</i> File missing	Error message shown	Proper error displayed	Pass

*Table 6.3.1.5 Use Case Test 05***Use Case Test 06**

Test Design Item		TD06 – Serve API Endpoint Testing		
Related Use Case ID		UC06		
Objective		To verify that the API server for external clients is correctly initialized, serves data extraction requests, and handles server failures gracefully. This ensures interoperability with third-party systems via the API.		
Test Coverage and Outcome				
Test Case ID	Test Condition/ Data	Expected Result	Actual Result	Pass/ Fail
TC-06-001	Main Flow – <i>Start API Server</i> User launches screen and clicks Start Server	API endpoint is active	Server running and status shown	Pass
TC-06-002	Alternate Flow – <i>Server Failure</i>	Error and troubleshooting shown	Error displayed correctly	Pass

Table 6.3.1.6 Use Case Test 06

6.3.2 Input Validation Testing

This section demonstrates how various types of input are validated to assess the system's capability to accurately process and extract meaningful data from each format. Input validation testing plays a critical role in verifying that the system can handle diverse input scenarios reliably and maintain data integrity throughout the extraction process. To ensure comprehensive coverage and resilience, the system was tested against a wide range of input types. Table 6.3.2.1 summarizes the tested input types along with their respective outcomes, providing insight into the system's performance across different data conditions, while Table 6.3.2.2 listed all the test data used for the testing.

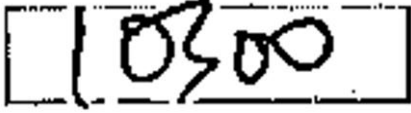





Test Case ID and Types of Inputs	Test Condition/ Data	Expected Output	Actual Output	Pass/Fail
IVTC-001 Printed Text	Upload a GDEX billing copy with printed text – Figure 6.3.2.1	Values of detected text – “MY35000371293”	MY35000371293	Pass
IVTC-002 Typed Text	Upload an image with type text – Figure 6.3.2.2	Values of detected text – “Test Case ID”	Test Case ID	Pass
IVTC-003 Handwritten Text	Upload a GDEX billing copy with handwritten notes – Figure 6.3.2.3	Values of detected text - “10300”	10300	Pass
IVTC-004 1D Barcode	Upload a GDEX billing copy with a 1D barcode – Figure 6.3.2.4	Values of extracted barcode – MY35000371293	MY35000371293	Pass
IVTC-005 2D Barcode	Upload a QR code image – Figure 6.3.2.5	Values of extracted barcode - 12345678	12345678	Pass
IVTC-006 Checkboxes and Form Elements	Upload a GDEX billing copy with checkboxes in various states (checked, unchecked) – Figure 6.3.2.6	Ticked values – “NON-DOCUMENT”	NON-DOCUMENT	Pass

REFERENCES

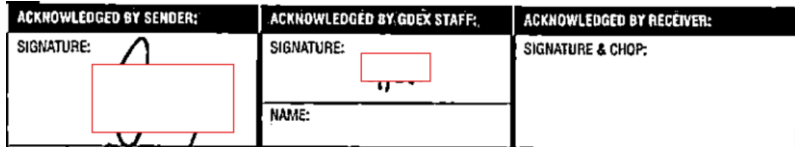
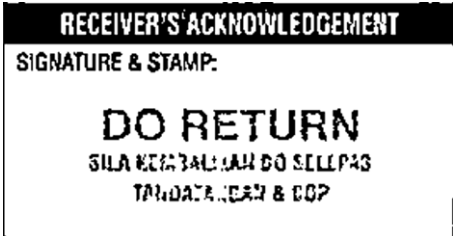

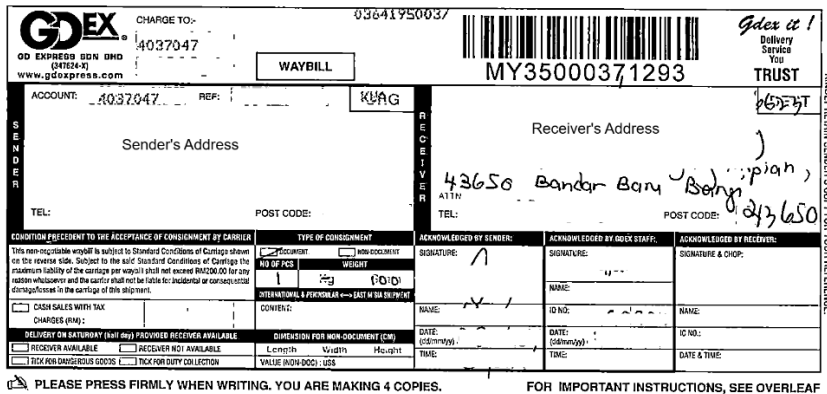
IVTC-007 Logos	Upload a GDEX billing copy with GDEX logo– <i>Figure 6.3.2.7</i>	Organization name – GDEX	GDEX	
IVTC-008 Symbols	Upload an image with a hand and pen symbol – <i>Figure 6.3.2.8</i>	Symbol meaning – “Sign here”	“Sign Here” or “Signature Required”	Pass
IVTC-009 Existence of Signature	Upload GDEX billing copy with several signatures– <i>Figure 6.3.2.9</i>	Presence of signatures – Yes, Yes, No	Yes, Yes, No	Pass
IVTC-010 Existence of Stamp/ Chop	Upload GDEX billing copy with one stamp – <i>Figure 6.3.2.10</i>	Presence of signatures – Yes	Yes	Pass
IVTC-011 Value of character-based Stamp	Upload GDEX billing copy with one stamp – <i>Figure 6.3.2.11</i>	Value of the stamp – “DO RETURN SILA KEMBALIKAN DO SELEPAS TANDATANGAN & COP”	DO RETURN SILA KEMBALIKAN DO SELEPAS TANDATANGAN & COP	Pass
IVTC-012 Short description of picture content	Upload GDEX billing copy – <i>Figure 6.3.2.12</i>	A short description of what the document is – “GDEX waybill document”	GDex waybill document	Pass

Table 6.3.2.1 Key Input Validation Test

Test Case ID	Test Data (Image)
IVTC-001 Printed Text	<p style="text-align: center;">MY35000371293</p> <p style="text-align: center;"><i>Figure 6.3.2.1 Test Data for IVTC-001</i></p>
IVTC-002 Typed Text	<p style="text-align: center;">Test Case ID</p> <p style="text-align: center;"><i>Figure 6.3.2.2 Test Data for IVTC-002</i></p>

IVTC-003 Handwritten Text	 <i>Figure 6.3.2.3 Test Data for IVTC-003</i>
IVTC-004 1D Barcode	 <i>Figure 6.3.2.4 Test Data for IVTC-004</i>
IVTC-005 2D Barcode	 <i>Figure 6.3.2.5 Test Data for IVTC-005</i>
IVTC-006 Checkboxes and Form Elements	 <i>Figure 6.3.2.6 Test Data for IVTC-006</i>
IVTC-007 Logos	 <i>Figure 6.3.2.7 Test Data for IVTC-007</i>
IVTC-008 Symbols	

REFERENCES

	<i>Figure 6.3.2.8 Test Data for IVTC-008</i>
IVTC-009 Existence of Signature	<p>The signature contains sensitive and private information. For confidentiality purposes, it has been obscured with boxes in Figure 6.3.2.9 as part of this report.</p>  <p><i>Figure 6.3.2.9 Test Data for IVTC-009</i></p>
IVTC-010 Existence of Stamp/ Chop	 <p><i>Figure 6.3.2.10 Test Data for IVTC-010</i></p>
IVTC-011 Value of character-based Stamp	 <p><i>Figure 6.3.2.11 Test Data for IVTC-011</i></p>
IVTC-012 Short description of picture content	 <p><i>Figure 6.3.2.12 Test Data for IVTC-012</i></p>

	<i>Figure 6.3.2.12 Test Data for IVTC-012</i>
--	--

Table 6.3.2.2 Input Validation Test Data

The input validation test cases listed under **IVTC-001 to IVTC-012** in Table 6.3.2.1 demonstrate the versatility of the system’s image processing and data extraction capabilities across a wide range of input types. These tests assess how accurately the system interprets and extracts textual, symbolic, and visual elements from documents such as GDEX billing copies. Each test case targets a unique type of data, from printed and handwritten text to barcodes, checkboxes, stamps, and even visual content understanding.

For instance, the system correctly identified values from various sources like printed tracking numbers (IVTC-001), typed text (IVTC-002), and handwritten notes (IVTC-003). This indicates that the LLM engine is well-calibrated for mixed input quality. Furthermore, the successful decoding of 1D and 2D barcodes (IVTC-004 and IVTC-005) confirms that barcode scanners are integrated and functioning accurately within the workflow.

Tests like IVTC-006 through IVTC-012 demonstrate the system’s advanced capabilities beyond basic OCR. It correctly interpreted checkbox states, identified the presence and value of stamps and signatures, and could describe the overall document content, a feature indicative of higher-level AI tasks such as image classification and semantic understanding. The system also showed success in symbol interpretation (e.g., recognizing a pen-and-hand icon as a signature prompt), which adds value in automating document handling workflows.

Overall, the test outcomes show that the system achieves a high degree of input versatility and content accuracy. Its ability to handle different text types, visual cues, and document structures makes it suitable for real-world deployment in logistics, business documentation, and compliance workflows, where variability in input format is common. The **Pass** results across nearly all test cases suggest the system is performing as intended, enhancing automation and reducing manual review needs.

Types of input that cannot be extracted:

- Watermark
- Hidden Text
- Seals
- Value of Signature
- Value of Stamp

While the system demonstrates strong capabilities in extracting structured data from various visual and textual inputs, there are still certain types of content that it cannot reliably extract. These include watermarks, hidden text, seals, and the actual values of signatures and stamps.

Watermarks, by design, are faint background elements meant to avoid interference with the main content; however, this subtlety makes them difficult for OCR engines to detect or extract without confusing them with noise. Similarly, hidden text, text embedded invisibly within files or obscured in images, falls outside the scope of standard visual recognition and cannot be extracted unless made explicitly visible. Seals, often embossed or faintly imprinted on paper, present a challenge due to their low contrast and absence of clear contours, which standard image processing techniques cannot easily decipher.

Additionally, while the system can detect the presence of a signature or stamp (i.e., whether it exists), it lacks the ability to interpret their semantic or textual values. This means it can flag that a stamp or signature is present but cannot determine the name signed or the message within the stamp. This limitation highlights the challenges of extracting detailed information from non-standardized and highly variable visual inputs, and it may require advanced AI models or manual review in workflows that depend on these elements.

6.4 Non-Functional Testing Setup and Result

In this section, non-functional testing, which is an assessment of system characteristics that define quality attributes, was carried out. It was conducted using 200 sample billing forms from GDEX Berhad, providing critical insights into the accuracy and performance of the system designed to extract unstructured data from images.

6.4.1 Testing Setup

To prove the effectiveness of the approach, three LLMs, including *qwen-vl-max*, *gemini-2.0-flash* and *gpt-4.1-2025-04-14* are used. We chose the following three multimodal LLMs for their leading performance on vision-language benchmarks and broad availability:

- qwen-2.5-vl: a 2.5 billion-parameter vision-language model known for strong numeric and text-layout understanding
- gemini-2.0-flash: Google's multimodal model optimized for document tasks
- gpt-4.1-2025-04-14: the latest GPT-4 variant with enhanced image-understanding capabilities

Figure 6.4.1 shows the example billing copy that contains the data field that needs to be extracted.

GDEX CHARGE TO: 03641950037
 DD EXPRESS SDN BHD (347624-X) 4037047
 www.gdexpress.com

WAYBILL CN number 4037047

ACCOUNT 4037 REF: KUAG

Postcode, Country 43650 Bandar Baru Bera, Pahang

TEL: POST CODE: Type

RECEIVER: 43650 Bandar Baru Bera, Pahang

TEL: POST CODE: 43650

CONDITION PRECEDENT TO THE ACCEPTANCE OF CONSIGNMENT BY CARRIER

TYPE OF CONSIGNMENT: ☒ DOCUMENT ☐ NON-DOCUMENT

NO OF PCS 1 WEIGHT 10.00

CONTENTS: 10.00

DELIVERY ON SATURDAY (ball day) PROVIDED RECEIVER AVAILABLE

☐ RECEIVER AVAILABLE ☐ RECEIVER NOT AVAILABLE

☐ TICK FOR DANGEROUS GOODS ☐ TICK FOR DUTY COLLECTION

VALUE (NON-DOC): US\$

ACKNOWLEDGED BY SENDER: SIGNATURE: NAME: DATE: TIME:

ACKNOWLEDGED BY GDEX STAFF: SIGNATURE: NAME: ID NO: DATE: TIME:

ACKNOWLEDGED BY RECEIVER: SIGNATURE & CHOP: NAME: ID NO: DATE & TIME:

PLEASE PRESS FIRMLY WHEN WRITING. YOU ARE MAKING 4 COPIES. FOR IMPORTANT INSTRUCTIONS, SEE OVERLEAF

Signature/Chop

Figure 6.4.1 Data that needs to be extracted from billing copy

Table 6.4.1. lists all the types of data extraction and description for each field. Note that most of the data is handwritten data.

Fields	Type of Extraction	Description
CN Number	Printed Text/ Barcode	13 digits as billing copy ID
Destination Postcode	Handwritten Text	5 – 9 digits for postcode number
Country	Handwritten Text (Implied from Address)	ISO 3166-1 alpha-2 format, e.g. MY
Type	Checkbox	Value that is checked/ ticked for type of consignment
Pieces	Handwritten Text	Number of pieces of the parcel
Account Number	Blurry printed/ handwritten text	7 digits for account number
Signature/ Chop	Presence of signature/chop	Existence of sign/ chop

Table 6.4.1 Type of data extraction and description for each field

Note that all non-functional testing described in this chapter was conducted using the Sieve Methodology without the inclusion of the LLM Challenger and LLM Referee components, due to their high token consumption. These components are intended for scenarios where absolute accuracy is prioritized over resource efficiency, making them suitable for use cases where cost is not a primary concern.

6.4.2 Data Extraction Accuracy

Data extraction accuracy is an evaluation of the correctness and precision of system outputs. The system's accuracy varied significantly depending on the data fields being extracted and the methods applied. The tables provided present the accuracy results for different data fields when processed through two approaches: directly feeding the images into the **LLM without processing (direct approach)** and passing the images through our **system (Sieve processing pipeline)**. The comparison between these two approaches highlights the impact of our system framework on data extraction accuracy.

In the **direct approach**, the raw sample images were passed straight into each LLM for data extraction with prompts requesting seven target fields. No image pre-processing (e.g., OCR or layout analysis) was applied. Table 6.4.2.1. depicts the accuracy of different data fields if the sample images are directly fed into LLM without processing.

Model Accuracy Fields	Accuracy (%)		
	qwen-2.5-vl	gemini-2.0-flash	gpt-4.1-2025-04-14
CN Number	91.00	91.00	80.50
Destination Postcode	77.00	79.00	82.50
Country	93.50	92.00	89.50
Type	55.50	81.50	57.00
Pieces	90.50	57.50	86.50
Account Number	82.50	86.50	66.00
Signature	81.50	83.00	80.50
Average	81.64	81.50	77.50

Table 6.4.2.1 Accuracy of different data fields with direct approach

Based on Table 6.4.2.1, the accuracy varies significantly across different fields, with some fields like “CN number” and “Country” having relatively high accuracy, while others like “Account Number” and “Type” show poor performance. This suggests that the direct approach may work well for certain types of information that are consistently presented but struggles with others that are more variable or complex. It reveals that LLMs alone struggle to consistently extract structured information from raw images, particularly when the layout is complex, or the text is not well isolated.

Additionally, a notable problem is that even when the LLM is prompted with “Return output in JSON format,” the output may vary in format or include extra details beyond the JSON structure, or the values does not conform to the required output criteria. It introduces extra output token usage. Besides, these inconsistencies can cause issues for subsequent modules that may not be able to process the non-standard output. This potential issue highlights the importance of having to implement strategies to mitigate hallucinations, such as incorporating additional validation steps like Sieve Approach. Overall, the LLMs – gemini-2.0-flash, qwen-2.5-vl and gpt-4.1-2025-04-14 – perform reasonably good in extracting the data from images.

In this project, which leveraged **the full data processing framework**, the images underwent several processing steps, including image preprocessing, image quality checks, barcode scanning, cropping, prompt engineering, and the application of the sieve approach after being processed by the LLM.

Table 6.4.2.2. shows the accuracy of different data fields passing through our system after applying our full approach, with three LLMs.

Model Accuracy Fields	Accuracy (%)		
	qwen-2.5-vl	gemini-2.0-flash	gpt-4.1-2025-04-14
CN Number	100.00	100.00	100.00
Destination Postcode	92.50	95.50	94.50
Country	99.50	98.50	99.00
Type	94.00	95.50	89.00
Pieces	99.00	97.00	99.00
Account Number	83.50	94.50	69.50
Signature	97.00	97.50	95.50
Average	95.07	96.93	92.36

Table 6.4.2.2 Accuracy of different data fields using system pipeline

Model Accuracy Approach	Average Accuracy (%)		
	qwen-2.5-vl	gemini-2.0-flash	gpt-4.1-2025-04-14
Direct Approach	81.64	81.50	77.50
System Pipeline	95.07	96.93	92.36
Increased Accuracy	13.43	15.43	14.86

Table 6.4.2.3 Comparison of average increased accuracy for 3 LLMs

Notably, Table 6.4.2.2 shows that after implementing our system pipeline, the accuracy of the data extraction can go up to **96.93%** with the use of gemini-2.0-flash. The results in Table 6.4.2.3 also highlight the substantial improvement in average data extraction accuracy achieved through the implementation of the system pipeline across three different LLMs, qwen-2.5-vl, gemini-2.0-flash, and gpt-4.1-2025-04-14. For the qwen-2.5-vl model, accuracy increased from 81.64% to 95.07%, resulting in a gain of 13.43 percentage points. Similarly, gemini-2.0-flash showed an even more pronounced improvement, rising from 81.50% to 96.93%, yielding the highest boost of **15.43 percentage points** among the models tested. gpt-4.1-2025-04-14 also saw a significant jump, improving from 77.50% to 92.36%, an increase of 14.86 percentage points. These figures consistently affirm that the pipeline not only enhances raw LLM output but also contributes to the reliability of information extracted from complex document formats.

REFERENCES

For better visualization, Figure 6.4.2.1 presents a bar chart comparing the model accuracy between the direct approach and the proposed system pipeline across three different LLMs.

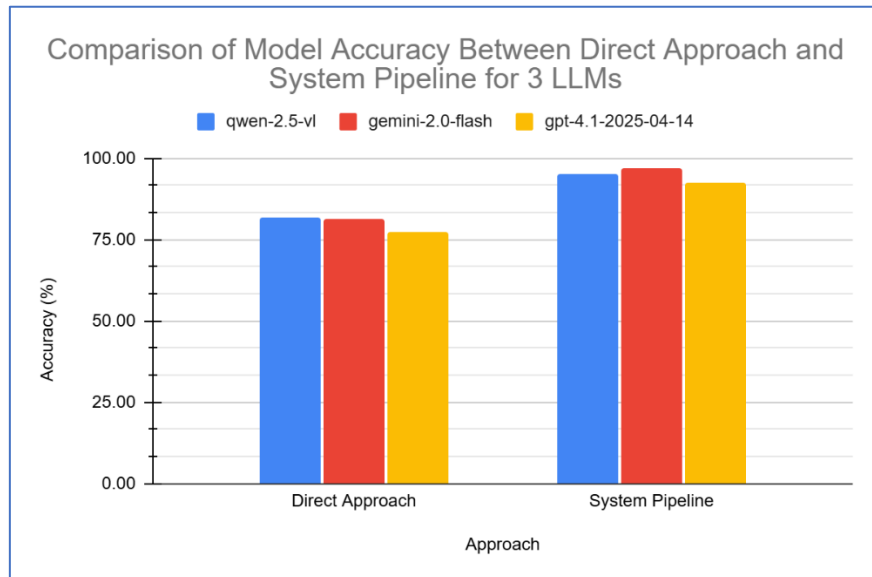


Figure 6.4.2.1 Bar chart to compare accuracy of approaches across 3 LLMs

Table 6.4.2.4 presents a comparison of the average accuracy achieved by the direct approach versus the system pipeline.

Model Accuracy Fields	Average Accuracy (%)	
	Direct Approach	System Pipeline
CN Number	87.50	100.00
Destination Postcode	79.50	94.17
Country	91.67	99.00
Type	64.67	92.83
Pieces	78.17	98.33
Account Number	78.33	82.50
Signature	81.67	96.67
Average	80.21	94.79

Table 6.4.2.4 Average accuracy of different data fields using both approaches

Most data field accuracies yielded dramatically improved result after applying the full approach, according to Table 6.4.2.3 and Table 6.4.2.4. The **direct approach** had the accuracy at 71.54%. **After applying the Sieve approach**, accuracy further increased

to 86.15%. The high initial accuracy highlights that the LLM handled numerical data well, and the Sieve approach further refined these results, leading to near-perfect accuracy. The average accuracy for fields like “CN Number,” “Country,” and “Pieces” reached or neared 100%, while even more challenging fields such as “Type” and “Postcode” saw considerable improvements. This suggests that the proposed pipeline effectively identified and corrected inaccuracies or ambiguities that the initial processing may have missed, thereby enhancing the data extraction process. It is especially effective for handwritten data such as postcode.

The increase in accuracy also indicates that the Sieve approach was successful in resolving ambiguities or correcting misinterpretations, such as distinguishing between similar data types or improving the accuracy of checkbox selections (type), where the accuracy jumped from 64.67% in the direct approach to 92.83% in the processed one. Notably, the accuracy for the destination postcode field improved from 79.50% to 94.17 % after applying the Sieve approach. This improvement suggests that the Sieve method fine-tuned the extraction process by verifying numerical data and correcting minor errors missed in the initial pass. It also indicates that the combination of layout awareness, targeted image regions, and smart validation (retries via the Sieve process) greatly enhances the LLMs’ ability to interpret and extract precise field data.

For better visualization, a bar chart for field-level accuracy comparison between the approaches is shown in Figure 6.4.2.2.

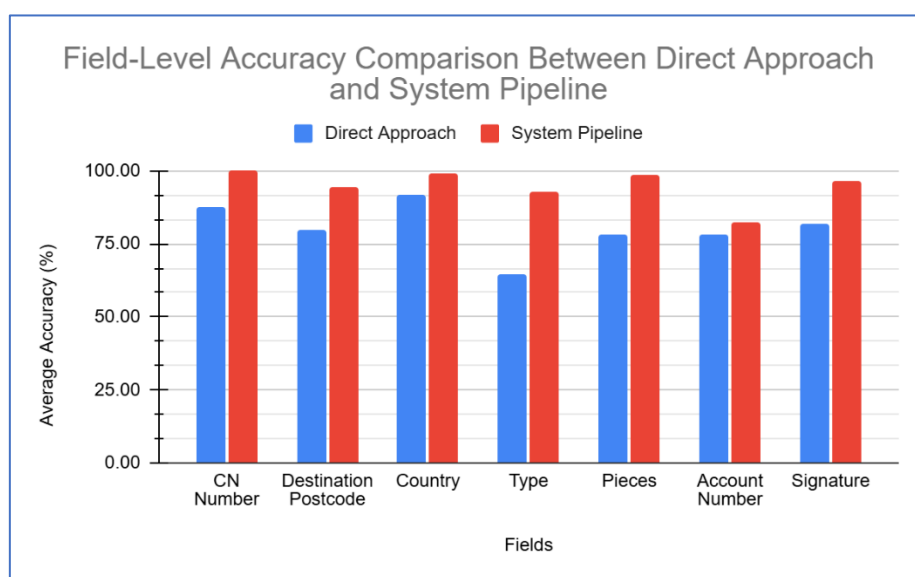


Figure 6.4.2.2 Bar chart to compare field-level accuracy of approaches

REFERENCES

Based on the differences in accuracy illustrated, the increased accuracy and performance improvements after applying the system framework is shown in Table 6.4.2.5. These improvements are calculated using the simple formula provided below:

Increased Accuracy (Percentage Increase) =

$$\text{Final Accuracy} - \text{Starting Accuracy}$$

Fields	Average Increased Accuracy (%)
CN Number	12.50
Destination Postcode	14.67
Country	7.33
Type	28.17
Pieces	20.17
Account Number	4.17
Signature	15.00
Average	14.57

Table 6.4.2.5 Overall Accuracy Improvement

The experimental results from both the direct and processed approaches as shown in Table 6.4.2.5 clearly demonstrate the effectiveness of the proposed system pipeline in improving data extraction accuracy across various fields.

The **CN number's** accuracy improved from 87.5% to 100% after processing, with 12.50% increased accuracy. This improvement can be attributed to the implementation of the barcode scanning method, which is highly reliable for reading standardized barcodes. The direct approach, relying solely on the LLM, likely struggled with image noise or inconsistencies, leading to lower accuracy.

The **destination postcode** accuracy improved from 79.50% to 94.17%, which has 14.67% increased accuracy. The preprocessing steps, particularly cropping to focus on relevant areas and reducing the complexity of the input, helped the LLM better identify and extract the postcode, which is often a critical piece of information. The improvement reflects the effectiveness of focusing the LLM's attention on specific image regions. However, there are some cases where the postcode extraction was more variable. For postcodes directly presented in the designated field, the cropping approach coupled with LLM analysis yielded good results.

REFERENCES

However, accuracy dropped when postcodes were embedded within larger address blocks, especially if more handwriting was involved. Some postcodes were incorrectly identified or missed entirely, leading to a drop in accuracy.

Country is usually implied from the postcode or the address, using LLM context understanding ability. It works surprisingly well, where the accuracy has increased from 91.67% to 99.00%, contributing to a near 100% accuracy.

The **account number**'s accuracy saw a marginal improvement from 78.33% to 82.50%. The total increased accuracy is 4.17%. This small increase indicates that while preprocessing aids in reducing some errors, the system still faces challenges with this field, potentially due to variability or the LLM's difficulty in distinguishing between similar-looking digits. Errors included misreading account numbers, particularly when the quality of the image was poor.

On the other hand, the accuracy of **signature** recognition improved significantly, increasing from 81.67% to 96.67%. This marks a 15.00% gain in accuracy improvement. This enhancement is largely attributed to the image preprocessing steps, particularly the precise labeling and cropping of signature regions, which enabled the LLM to more effectively identify and interpret the presence of signatures within the designated areas.

The extracted fields can be grouped to calculate the average accuracy for different types of data extraction. CN number and account number are considered as printed text, while postcode, country (derivable), and pieces are handwritten text. Table 6.4.2.6 presents the average accuracy for each data type, derived from the detailed results in Table 6.4.2.4.

Accuracy Difference (Percentage Increase %) =

$$\text{Final Accuracy} - \text{Starting Accuracy}$$

Model Accuracy Type of Data	Average Accuracy (%)		
	Direct Approach	System Pipeline	Difference
Printed Text	82.92	91.25	8.33
Handwritten Text	83.11	97.17	14.06
Checkbox	64.67	92.83	28.16

REFERENCES

Presence of Signature	81.67	95.50	13.83
-----------------------	-------	-------	-------

Table 6.4.2.6 Average accuracy for each type of data for extraction

For improved clarity and comparison, a bar chart is presented in Figure 6.4.2.3, illustrating the average accuracy across different types of data extraction.

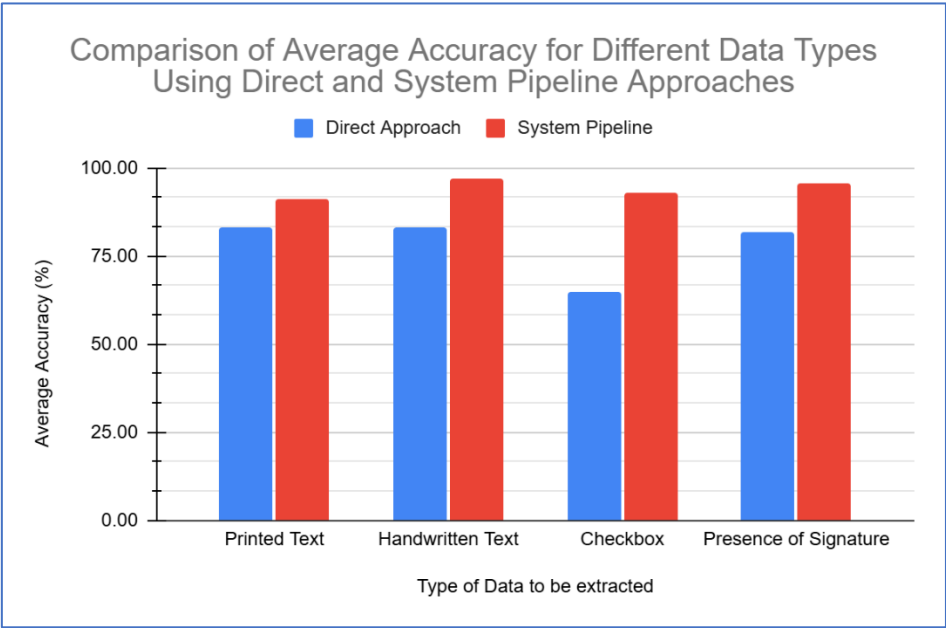


Figure 6.4.2.3 Bar Chart of Average Accuracy for Different Types of Data Extraction

The results presented in Table 6.4.2.5 demonstrate the comparative accuracy improvements in extracting various types of data when using the proposed System Pipeline versus a Direct Approach. This analysis is crucial in evaluating the system’s real-world performance across different content categories, such as printed text, handwriting, checkboxes, and signature detection.

The accuracy improved from **82.92% to 91.25%**, a gain of **8.33%**. This indicates that even for relatively well-structured and machine-friendly data like printed text, the pipeline (which includes preprocessing, OCR refinement, and prompt engineering) enhances the LLM’s extraction capability by providing cleaner, more context-aware inputs.

Accuracy of handwritten text recognition jumped from **83.11% to 97.17%**, showing a **14.06% improvement**. This category is traditionally challenging due to

inconsistencies in handwriting, but the system pipeline, likely through better region targeting and preprocessing, helps the LLM interpret these fields more reliably.

The most dramatic gain is observed here: from **64.67% to 92.67%**, marking a **28.00% improvement**. Checkboxes often have subtle visual cues that are not text-based, so preprocessing (e.g., ROI cropping, binarization) and specialized classification prompts likely made a huge difference in enabling the LLM to distinguish marked versus unmarked boxes.

Accuracy improved from **81.67% to 96.67%**, reflecting a **15.00% increase**. Signatures often appear in specific zones with highly varied visual patterns. The pipeline's ability to segment these areas and apply tailored prompts or detection logic enables the LLM to recognize the presence or absence of a signature with much higher precision.

All in all, the analysis shows that preprocessing steps significantly improve the accuracy of data extraction across most fields, particularly for structured data like CN numbers, postcodes, and types. It successfully achieves up to **96.93% accuracy** and increased overall accuracy by approximately **13.43 – 15.43%**, depending on the LLMs. The improvements underscore the importance of preprocessing in helping the LLM focus on relevant parts of the image, reducing noise, and increasing clarity. It might be owing to the sieve approach that was introduced to mitigate these issues. However, challenges remain for fields involving complex or ambiguous data particularly with handwritten entries and overlapping text, such as account numbers, where further refinement is needed. Overall, the results highlight the effectiveness of the current system and suggest areas for future development to achieve even higher accuracy and reliability.

6.4.3 Output Delivery Time

In this section, testing of output delivery time was conducted for measurement of system responsiveness and latency. The output delivery time assessment in the project evaluates how long each approach, direct versus pipeline, takes to process and return results for a fixed dataset (200 images).

Table 6.4.3.1 and Table 6.4.3.2 show the system performance efficiency across different models and strategies. The difference between the processing time over 3 models are compared, where the time difference it is calculated as below:

$$\text{Time Difference} = \text{Pipeline Time} - \text{Direct Time}$$

Time Approach	Processing Time per image (s)		
	qwen-2.5-vl	gemini-2.0-flash	gpt-4.1-2025-04-14
Direct Approach	4.36	3.43	5.16
System Pipeline	3.91	3.82	3.60
Time Difference (s)	-0.45	0.39	-1.56
Average Time Improvement (%)	-10.32	11.37	-30.23

Table 6.4.3.1 Processing time of 3 LLMs using different approaches

Approach	Average Processing Time (s)
Direct Approach	4.32
System Pipeline	3.78
Time Difference (s)	-0.53

Table 6.4.3.2 Average processing time and difference

$$\text{Time Efficiency Ratio} = \left(\frac{\text{Pipeline Time}}{\text{Direct Time}} \right) \times 100 = \frac{3.83}{4.32} \times 100 = \mathbf{88.66}$$

$$\begin{aligned} \text{Time Efficiency Improvement (\%)} &= \frac{(\text{Pipeline Time} - \text{Direct Time})}{\text{Direct Time}} \times 100 \\ &= \frac{(3.78 - 4.32)}{4.32} \times 100 = \mathbf{-12.50\%} \end{aligned}$$

According to Table 6.4.3.1, the results show mixed performance across the models. While models like qwen-2.5-vl and gpt-4.1-2025-04-14 experience significant improvements in processing time with the pipeline (time saved: 0.45s and 1.56s

REFERENCES

respectively), gemini-2.0-flash exhibits a slight increase (0.39s slower in pipeline mode). Notably, gpt-4.1-2025-04-14 has achieved a 30.23% reduction in processing time relative to the direct execution path, which means it completes tasks significantly faster than previous baseline method, despite Sieve Methodology likely takes more time to be conducted due to extra reprocessing needed. On average, the system pipeline achieves a faster processing time of 3.78 seconds, compared to 4.32 seconds in the direct approach. This equates to a time difference of 0.53 seconds saved per 200 images, showing that the pipeline improves processing speed overall despite the extra processing steps.

Table 6.4.3.2 further quantifies this finding. The **time efficiency ratio** is calculated to be **88.66%**, suggesting that the system pipeline requires only ~89% of the time needed by the direct method. However, the time efficiency improvement, computed as a percentage decrease in time, stands at -12.50%. This indicates a 12.50% reduction in processing time, meaning the system pipeline delivers results faster on average, while also offering improvements in accuracy (as shown in previous analyses).

However, in the system, after we adopted thread management, asynchronous processing and rate limit for batch processing, we successfully reduce the time for processing 200 images to total 108.87 seconds. Table 6.4.3.3 shows the total average processing time and average processing time per image after applying batch processing, while Table 6.4.3.4 shows the average processing time per image for different approaches.

Model	qwen-2.5-vl	gemini-2.0-flash	gpt-4.1-2025-04-14
Total Processing Time (s)	258.24	252.91	108.87
Average Processing Time per image (s)	1.2912	1.2996	0.5439

Table 6.4.3.3 Total and average time per image for batch processing

Processing Time Approach	Processing Time per image (s)		
	qwen-2.5-vl	gemini-2.0-flash	gpt-4.1-2025-04-14
Direct Approach	4.36	3.43	5.16
System Pipeline	1.29	1.30	0.54

Time Difference (s)	-3.07	-2.13	-4.62
Time Efficiency Improvement (%)	-70.39	-62.11	-89.46

Table 6.4.3.4 Average time per image for batch processing for different approaches

For clearer visualization, Figure 6.4.3.1 presents a bar chart comparing the average processing time per image during batch processing between the direct approach and the system pipeline across three different LLMs.

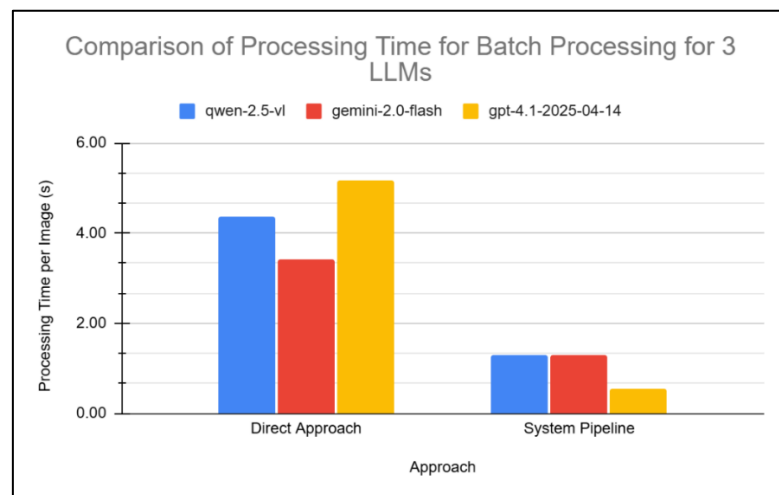


Figure 6.4.3.1 Bar chart of processing time for batch processing

Approach	Average Processing Time (s)
Direct Approach	4.32
Batch Processing Approach	1.04
Time Difference	-3.28

Table 6.4.3.5 Average processing time for difference approaches

Overall Time Efficiency Improvement (%)

$$\begin{aligned}
 &= \frac{(\text{Batch Processing Time} - \text{Direct Approach Time})}{\text{Direct Approach Time}} \times 100 \\
 &= \frac{(1.04 - 4.32)}{4.32} \times 100 = -75.96\%
 \end{aligned}$$

The results shown in Table 6.4.3.3, Table 6.4.3.4. and Table 6.4.3.5 highlight the substantial improvements achieved through the implementation of a batch processing technique, specifically involving thread management, asynchronous processing, and rate-limiting within the system. These optimizations are designed to process multiple

images in parallel without overwhelming the system or the LLM API, leading to better throughput and more efficient use of compute resources.

Before batch optimization, the average processing time per image was between 3.60 to 5.16 seconds depending on the model (as noted in Table 6.4.3.4). After the batch enhancements, qwen-2.5-vl achieved an average processing time of 1.2912 seconds per image. It is due to rate limitations; it reaches the limit before completing the entire processing task. However, it still achieves reduction of 2.13 seconds and time efficiency improvements of 62.11% compared to its previous result.

gemini-2.0-flash reached 1.2996 seconds per image. Similarly, rate constraints prevent it from completing processing before the limit is reached. Despite the rate limit, it process the image 3.07 seconds faster than the direct approach, with a time efficiency improvement of 70.39%.

gpt-4.1-2025-04-14 significantly outperformed the others with only 0.5439 seconds per image, processing 200 images in just 108.87 seconds total. It is because it had a higher rate limit in the testing environment, meaning it could handle more concurrent or rapid requests without being throttled. This allowed it to complete the task (200 images in 108.87 seconds) much faster, more in line with its true processing capability. Thus, it has the highest time difference of 4.62 seconds with **time efficiency improvement of 89.46%**, which is considered very efficient.

This demonstrates a drastic **reduction in processing time**, which achieved an average of **75.96% time savings** compared to its original pipeline runtime (~4.32s per image previously). For qwen-2.5-vl and gemini-2.0-flash, their performance is bottlenecked not by speed, but by access restrictions. Once they reach a set threshold of allowed requests per second or minute (set by the LLM provider), further processing is delayed or throttled. So even if the model could process faster, it gets paused or slowed down artificially.

The effectiveness of this batch strategy suggests that concurrent and asynchronous execution is critical in high-throughput scenarios. Threading enables the system to utilize CPU resources more effectively, while asynchronous calls prevent I/O wait times from blocking the pipeline. The use of rate limiting also avoids overloading the LLM API and keeps processing within allowable request thresholds.

6.4.4 Resource Utilization

The resource utilization evaluation of this project focuses on two key metrics: token usage and associated cost. These are critical for understanding the operational efficiency when using different LLMs in both a direct approach and a pipeline-based approach. Table 6.4.4.1, Table 6.4.4.2 and Table 6.4.4.3 shows the token usage of 3 LLMs using different approaches.

<div> <div>Approach</div> <div>Token</div> </div>	Direct Approach			System pipeline		
	qwen-2.5-vl	gemini-2.0-flash	gpt-4.1-2025-04-14	qwen-2.5-vl	gemini-2.0-flash	gpt-4.1-2025-04-14
Prompt Token	1359.69	2362.97	1479.13	830.2	1998.99	998.2
Completion Token	91.97	77.69	77.91	74.23	97.98	42.38
Total Token	1451.66	2440.66	1557.04	904.43	2096.97	1040.58

Table 6.4.4.1 Token of 3 LLMs using different approaches

$$\text{Token Difference} = \text{Pipeline tokens per image} - \text{Direct tokens per image}$$

$$\text{Token Efficiency Improvement (\%)} = \frac{(\text{Pipeline Token} - \text{Direct Token})}{\text{Direct Token}} \times 100$$

<div> <div>Token</div> <div>Approach</div> </div>	Total Token Usage per Image		
	qwen-2.5-vl	gemini-2.0-flash	gpt-4.1-2025-04-14
Direct Approach	1451.66	2440.66	1557.04
System Pipeline	904.43	2096.97	1040.58
Token Difference	-547.23	-343.69	-516.46
Token Efficiency Improvement (%)	-37.70%	-14.08%	-33.17%

Table 6.4.4.2 Comparison of token usage of 3 LLMs using different approaches

Approach	Average Token Used
Direct Approach	1816.45
System Pipeline	1347.33
Token Difference	-469.13

Table 6.4.4.3 Average total token used for different approaches

$$\text{Token Efficiency Ratio} = \left(\frac{\text{Pipeline Tokens}}{\text{Direct Tokens}} \right) \times 100 = \frac{1347.33}{1816.45} \times 100 = \mathbf{74.17}$$

REFERENCES

Token usage of LLM often comes with a cost. Table 6.4.4.4 shows the cost of different models and the reduction after applying the pipeline, and the formula to calculate average cost per image and cost difference is also detailed below.

Average cost per image (\$)

$$= \text{Average prompt token} \times \text{Model input price} \\ + \text{Average output token} \times \text{Model output price}$$

Cost Difference = Pipeline cost per image – Direct cost per image

Cost Approach	Average cost per image (\$)		
	qwen-2.5-vl	gemini-2.0-flash	gpt-4.1-2025-04-14
Direct Approach	0.001117	0.000267	0.003582
System Pipeline	0.000688	0.000239	0.002334
Cost Difference	-0.000429	-0.000028	-0.001248
Cost Efficiency Improvement (%)	-38.43	-10.58	-34.84

Table 6.4.4.4 Comparison of cost for 3 LLMs using different approaches

The average cost per image across 3 LLM is illustrated in table 6.4.4.5.

Approach	Average cost per image across 3 models (\$)
Direct Approach	0.001655
System Pipeline	0.001087
Token Difference	-0.000568

Table 6.4.4.5 Average total cost used for different approaches

$$\text{Cost Efficiency Ratio} = \left(\frac{\text{Pipeline Cost}}{\text{Direct Cost}} \right) \times 100 = \frac{0.001087}{0.001655} \times 100 = \mathbf{65.68}$$

$$\text{Overall Cost Efficiency Improvement (\%)} = \frac{(\text{Pipeline Cost} - \text{Direct Cost})}{\text{Direct Cost}} \times 100 \\ = \frac{(0.001087 - 0.001655)}{0.001655} \times 100 = \mathbf{-34.32\%}$$

The comparison in Table 6.4.4.3 between the direct and system pipelines demonstrates that the pipeline strategy significantly reduces total token consumption across all three LLMs. When comparing the approaches, the **direct approach** resulted in the higher token usage and, consequently, the **highest cost**. For example, according to Table

6.4.4.1, in the case of qwen-2.5-vl, the total token count per image dropped from **1451.66 to 904.43** tokens, indicating a reduction of over 500 tokens. Similarly, gpt-4.1-2025-04-14 showed a reduction from 1557.04 to 1040.58 tokens. On average, across all three models, the direct approach consumed 1816.45 tokens per image, whereas the system pipeline used only 1347.33. The direct approach consumes higher token usage due to the larger image size and lack of preprocessing, which caused the LLM to process more data, thus increasing the total token count.

Despite variations across models, by using the system, it translates to a **token efficiency improvement** of approximately **37.70% with qwen-2.5-vl model and 34.84% with gpt-4.1-2025-04-14 model**, as shown in Table 6.4.4.2. This improvement is largely attributed to pre-processing mechanisms such as prompt refinement, ROI targeting, and intelligent prompt generation, which help limit the context sent to the model and thus reduce the prompt size. This substantial decrease highlights the economic benefits of the preprocessing and Sieve approach. It is important to note, however, that token usage and calculation differ between models. For instance, **gemini-2.0-Flash** typically consumes more tokens when processing images and exhibited a more modest **10.58% reduction**, which is still notable given its inherently high token usage.

From a cost perspective, the reductions in token usage naturally led to lower processing costs. Table 6.4.4.5 shows that, on average, the cost per image decreased from **\$0.001655 (direct approach) to \$0.001087 (system pipeline)**, yielding an average saving of \$0.000568 per image. This represents a cost efficiency improvement of approximately **34.31% on average**, almost directly in line with the token efficiency gains. Among the three models, **qwen-2.5-vl model** benefitted the most in terms of absolute cost reduction, due to its higher saving of token usage. In fact, it has a **cost efficiency improvement of around 38.43%**, hence proving that it is especially important for LLM that has higher pricing or higher token usage to adopt the measures. These savings become substantial when scaled across thousands or millions of images, validating the system pipeline as a more resource-conscious strategy.

In conclusion, the project's pipeline implementation not only boosts accuracy, as demonstrated in earlier tables, but also proves to be far more efficient in terms of resource consumption—both in token usage and monetary cost. This efficiency directly supports scalability and cost-effectiveness in production environments.

6.5 User Acceptance Testing (UAT)

User Acceptance Testing (UAT) was conducted in collaboration with a logistics company to validate the end-to-end integration of the data extraction system into their billing and documentation workflow. The testing was designed in two distinct phases, aligning with their operational migration strategy from manual data entry to full automation, as illustrated in the integration plan in Figure 6.5.1.

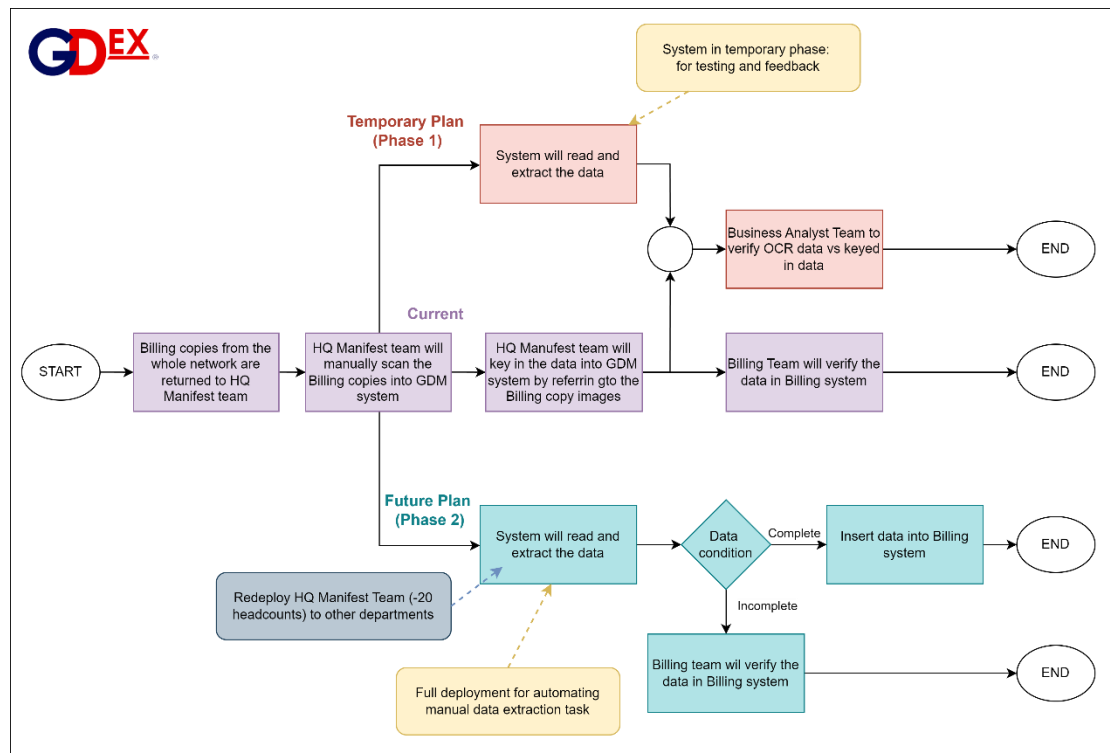


Figure 6.5.1 GDEX Berhad Process Flow Map with System Integration

In Phase 1 (Temporary Plan), the system was deployed in a controlled testing environment where the OCR engine extracted data from billing copies returned to the HQ Manifest team. The Business Analyst Team was responsible for validating the accuracy of the OCR output by cross-referencing it against manually keyed-in data. This allowed real-time benchmarking of the automated system's extraction capabilities while collecting feedback on its performance. The goal of this phase was not full automation but to verify that the system could reliably extract core data fields and meet predefined accuracy thresholds. UAT during this stage revealed critical insights into field recognition consistency, especially for handwritten entries and low-resolution documents, allowing for targeted refinement of template settings and prompt engineering.

Phase 1 has been successfully tested, and feedback was gathered from the development team. Based on testing with thousands of data samples, the system achieved approximately **92–95%** accuracy in postcode detection and **99%** accuracy in country recognition. These results are considered stable and reliable, especially given the variability and complexity of handwritten inputs.

In Phase 2 (Future Plan), following improvements from the initial phase, the system entered a broader rollout aimed at operational integration. The system was configured to automatically extract data from incoming scanned billing copies and evaluate data completeness. If the extracted data met accuracy conditions (i.e., passed validation using the Sieve Approach), it will be automatically inserted into the Billing System. For any incomplete or questionable outputs, the system routed these to the Billing Team for manual verification. This ensures that the automation process did not compromise billing integrity. UAT in this phase involved real billing records and live system connections, validating the full pipeline, from data ingestion and extraction to billing system updates. Additionally, the future plan includes redeploying ~20 HQ Manifest staff to other departments, reinforcing the system's long-term impact in reducing manual labor and operational costs.

Overall, UAT confirmed the system's readiness for production deployment. It validated not only technical accuracy but also usability and integration with existing business processes, setting the foundation for full-scale digital transformation of document handling in the logistics workflow.

6.6 Project Challenges

Despite the promising results, several challenges and limitations were identified during the development and testing phase, including handling handwritten data, balancing accuracy and processing time, token usage and cost management and the complexity of data structure.

One of the most significant challenges encountered during the development was **accurately processing handwritten data**, particularly in fields like account numbers. Handwriting can be highly variable, with differences in style, legibility, and alignment, making it difficult for the LLM and even traditional OCR techniques to interpret correctly. Despite the preprocessing and Sieve approach, the accuracy in these fields remained relatively low, highlighting the limitations of current methods in dealing with unstructured, handwritten inputs.

Balancing accuracy and processing time was hard in implementing the framework. The Sieve approach, while effective in improving accuracy, introduced additional processing time due to its iterative nature. This posed a challenge in weighing the need for high accuracy with the requirement for timely data processing. In time-sensitive applications, this trade-off could become a significant hurdle, potentially limiting the system's effectiveness in scenarios requiring rapid processing.

Another challenge was **managing token usage and associated costs, with risk to scalability**. The testing showed that while preprocessing reduced token usage, the Sieve approach added to the total token count, slightly increasing the cost per image. This presents a challenge in scaling the system for large-scale operations, where even small increases in cost per image could accumulate into substantial expenses.

The system struggled with data fields that contained **complex or ambiguous structures**, such as overlapping text or inconsistent data formatting. For example, the system had difficulty accurately interpreting account number when the label overlapped with column line or when the format varied between images. These complexities require more sophisticated parsing and context understanding, which current methods may not fully support.

6.7 Overall Performance and Objective Evaluation

The system evaluation in this chapter demonstrates a successful realization of the project's primary and sub-objectives, particularly in ensuring consistent and accurate data extraction from unstructured document images. The overarching aim was to develop a scalable and reliable LLM-based framework that mitigates the inherent variability of language models and enables seamless integration into enterprise-level logistics and data processing workflows. The outcomes of the system verification, functional and non-functional testing, and real-world user acceptance testing collectively affirm the project's effectiveness in delivering on this promise.

From a performance perspective, the framework achieved a significant improvement in data extraction accuracy. The average accuracy increased from 80.21% using the direct LLM approach to 94.79% when processed through the complete pipeline, which includes the Sieve methodology. This improvement of approximately **14.57 percentage points**, representing a **34.83% performance gain**, aligns well with the target of at least a **10% improvement in accuracy**. High accuracy was achieved particularly for fields such as CN Number (100%), Country (99%), and Signature Presence (96.67%), affirming the soundness of the framework in high-stakes scenarios where reliability is critical.

In terms of integration and output consistency, the system successfully addressed the variability of LLM outputs. Through mechanisms like prompt engineering, targeted cropping, and structured formatting enforcement via the Sieve method, the framework ensured that outputs conformed to predefined schemas suitable for downstream systems. These safeguards not only mitigated hallucinations and formatting inconsistencies but also ensured that the system outputs could be directly consumed by applications in logistics without further manual intervention, meeting the goal of effortless incorporation of LLM capabilities.

Efficiency was another area of substantial progress. The system's average processing time per image improved from 4.32 seconds (direct) to just 1.04 seconds with optimized batch processing, achieving an **average of 75.96% time efficiency improvement and highest of 89.46%**. This was made possible through innovations like asynchronous processing, thread management, and API rate control, which enhanced throughput and

scalability, thus fulfilling the objective of reducing processing time and resource consumption by at least 25%. Additionally, **token usage was reduced by highest of 37.70%** on average across models, resulting in proportional **cost savings of 38.43%**, making the solution more economically viable for enterprise-scale deployment.

Real-world applicability was confirmed through User Acceptance Testing (UAT) conducted in collaboration with a logistics firm. During controlled and operational rollout phases, the system consistently delivered high-accuracy outputs (up to 99% for certain fields) and supported integration into live billing systems. It facilitated partial and then full automation of previously manual workflows, enabling significant reductions in human labor and error rates. This not only validated the technical capabilities but also confirmed the usability and business value of the system.

In conclusion, the project met and, in several cases, exceeded its defined objectives. The system reliably extracted structured data from complex, variable inputs such as handwritten notes and form elements, reduced reliance on manual intervention, and optimized efficiency both in processing time and resource consumption. While some challenges remain for specific fields (e.g., account number recognition from poorly printed text), the framework provides a solid foundation for further refinement.

6.8 Concluding Remark

In summary, the system evaluation confirms that the proposed solution successfully fulfils its design objectives, demonstrating strong performance in both accuracy and efficiency. Through rigorous functional, non-functional, and user acceptance testing, the system has proven its ability to reliably extract structured data from a variety of unstructured sources, significantly outperforming baseline LLM approaches. The implementation of preprocessing techniques, the Sieve methodology, and batch optimization collectively contributed to enhanced accuracy, reduced processing time, and lower token usage. These advancements establish the framework as a scalable, and production-ready solution for real-world integration in logistics and similar data-intensive domains. Overall, the project represents a compelling blueprint for enterprise-grade LLM integration, especially in logistics and document-heavy industries. The evaluation highlights both the achievements and remaining challenges, offering valuable insights for continued refinement and future deployments.

CHAPTER 7 CONCLUSION AND RECOMMENDATION

This section concludes the report by summarizing the key achievements of the project, reflecting on its limitations, and identifying opportunities for future improvements. It highlights the practical contributions of the system, particularly its applicability to real-world challenges in the logistics sector and other industries that rely on accurate and efficient data extraction from unstructured sources.

7.1 Conclusion

The project “Large Language Model Application Integration for Extracting Unstructured Data” addresses the critical challenge of LLM integration in automating the extraction of meaningful and structured information from unstructured data sources, particularly images, which are widely used across various industries such as logistics, healthcare, and finance. The problem revolves around the difficulty of accurately and efficiently extracting meaningful information from unstructured sources via LLM integration. This challenge is compounded by the non-deterministic nature of LLMs, which can produce inconsistent outputs, making their integration into existing systems particularly difficult. Besides, the existing LLM-based data extraction system are often resource-heavy, especially when applied to image inputs.

The motivation behind this project is the need to bridge the gap between the advanced capabilities of LLMs and the practical demands of industries that require precise and reliable data processing, with lower cost. To achieve this, the project proposed the development of a framework that integrates LLMs into traditional data processing pipelines, ensuring that the output is both accurate and consistent.

The proposed solution involves integrating Large Language Models (LLMs) into data processing frameworks to enhance the accuracy and consistency of data extraction. The methodology includes a full pipeline of data preparation, data extraction and analysis and data post-processing. One of the novel contributions of this project is the introduction of the “**Sieve Methodology**,” a new approach designed to address the inherent unpredictability of LLMs. This methodology involves multiple iterations of validation, comparison, and refinement of the data extracted by LLMs, ensuring that outputs are reliable and suitable for downstream processing. This innovation is

particularly valuable for industries that concern about precision. It allows the integration of LLMs within the proposed framework, providing a solution for the automated extraction of unstructured data, particularly in high-stakes environments.

The project achieved significant measurable results. Comprehensive testing and evaluation revealed a significant achievement in data extraction accuracy, by up to **96.93% accuracy**, alongside a notable up to **89.46% improvement in processing time** and **37.70% reduction in token usage** compared to baseline method. These results demonstrate that the project's objectives were successfully met, particularly in improving efficiency, accuracy, and cost-effectiveness. Besides, these improvements also collectively demonstrate the system's potential for real-world deployment, particularly in high-stakes environments such as logistics, billing, and automated record management.

Besides, a key part derived from this project is the dual-functionality system, which allows the integration of LLMs as both a standalone tool for direct user interaction and as an intermediate module within automated workflows. This flexibility makes the system adaptable to a wide range of use cases, offering industries a powerful tool to enhance operational efficiency and reduce manual labour.

Despite its success, the system has some limitations, such as difficulties in handling highly illegible handwriting and limited multi-language support. These challenges present opportunities for future development, including the integration of advanced handwriting recognition models, support for diverse document formats, and deployment in broader operational contexts.

In essence, the project successfully connects AI capabilities with the practical needs of industries dealing with unstructured data. The developed system not only meets the project's initial objectives but also sets the stage for broader adoption of AI-driven solutions, contributing to a more automated and efficient future across various sectors.

7.2 Future Considerations and Recommendations

While the current system has met its primary objectives, there are several areas that warrant further exploration to enhance its performance, adaptability, and scalability in future iterations. One of the key considerations is the improvement of handwriting

recognition accuracy. Although the existing solution performs well on moderately legible text, it faces challenges when processing poorly written or stylized handwriting. Integrating advanced OCR-LLM trained specifically on diverse handwriting samples could significantly enhance recognition reliability across various document types.

Another important recommendation is to introduce **active learning and feedback mechanisms**. By allowing users to verify and correct extracted outputs within the interface, the system could collect valuable real-time data to retrain or fine-tune the LLM and validation components. This iterative learning approach would not only increase the system's accuracy over time but also enable better adaptability to document variations and evolving use cases in dynamic environments like logistics.

Looking ahead, one of the most promising avenues for enhancing the current system is the integration of **Retrieval-Augmented Generation (RAG)**. RAG combines the generative capabilities of LLMs with an external retrieval mechanism, allowing the model to ground its responses in relevant documents or datasets. By incorporating RAG, the system could dynamically retrieve supporting content, such as templates, glossary terms, domain-specific rules, or historical data, from a structured knowledge base or database during inference. In the context of document processing and information extraction, this hybrid approach could significantly improve both the **accuracy** and **contextual relevance** of the outputs.

Expanding the framework's support for **multilingual documents** is also a valuable direction for future work. Many logistics and enterprise systems operate in multilingual contexts, and enabling the extraction of data from documents written in languages other than English would significantly broaden the system's applicability. Incorporating translation modules or training language-specific LLM pipelines could address this need effectively.

Lastly, from a deployment perspective, migrating the system to a **cloud-based infrastructure** would greatly enhance its scalability and accessibility. Cloud deployment would support real-time document processing at scale, offer integration via RESTful APIs, and allow seamless updates and monitoring. It would also enable organizations to leverage the system without requiring local setup, making adoption faster and more flexible.

REFERENCES

- [1] A. Singh, 'Exploring Language Models: A Comprehensive Survey and Analysis', in *2023 IEEE International Conference on Research Methodologies in Knowledge Management, Artificial Intelligence and Telecommunication Engineering, RMKMATE 2023*, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/RMKMATE59243.2023.10369423.
- [2] M. A. K. Raiaan *et al.*, 'A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges', *IEEE Access*, vol. 12, pp. 26839–26874, 2024, doi: 10.1109/ACCESS.2024.3365742.
- [3] Poonam A. Wankhede and Sudhir W. Mohod, 'A Different Image Content-based Retrievals using OCR Techniques', in *International Conference on Electronics, Communication and Aerospace Technology(ICECA)2017*, IEEE, 2017, pp. 155–161.
- [4] U. K. V. Karanth, A. T. Sujana, T. Y. R. Kumar, S. S. Joshi, A. K. P. Rani, and S. Gowrishankar, 'Breaking Barriers in Text Analysis: Leveraging Lightweight OCR and Innovative Technologies for Efficient Text Analysis', in *2nd International Conference on Automation, Computing and Renewable Systems, ICACRS 2023 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 359–366. doi: 10.1109/ICACRS58579.2023.10404305.
- [5] T. N. Thi, T. H. Do, and M. Yoo, 'Implementation of OCR system on extracting information from Vietnamese book cover images.', in *International Conference on Advanced Technologies for Communications*, IEEE Computer Society, 2023, pp. 427–432. doi: 10.1109/ATC58710.2023.10318889.
- [6] C. W. Park, V. Palakonda, S. Yun, I. M. Kim, and J. M. Kang, 'OCR-Diff: A Two-Stage Deep Learning Framework for Optical Character Recognition Using Diffusion Model in Industrial Internet-of-Things', *IEEE Internet Things J*, 2024, doi: 10.1109/JIOT.2024.3390700.
- [7] A. F. Mohammad, B. Clark, and R. Hegde, 'Large Language Model (LLM) & GPT, A Monolithic Study in Generative AI', in *Proceedings - 2023 Congress in Computer Science, Computer Engineering, and Applied Computing, CSCE 2023*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 383–388. doi: 10.1109/CSCE60160.2023.00068.

REFERENCES

- [8] J. E. Ho, B. Y. Ooi, and M. Westner, 'Application Integration Framework for Large Language Models', in *2024 5th International Conference on Artificial Intelligence and Data Sciences, AiDAS 2024 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2024, pp. 398–403. doi: 10.1109/AiDAS63860.2024.10730541.
- [9] 'Reproducible outputs ', OpenAI. Accessed: May 05, 2025. [Online]. Available: <https://platform.openai.com/docs/guides/text-generation/reproducible-outputs>
- [10] S. K. Routray, A. Javali, K. P. Sharmila, M. K. Jha, M. Pappa, and M. Singh, 'Large Language Models (LLMs): Hypes and Realities', in *2023 International Conference on Computer Science and Emerging Technologies, CSET 2023*, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/CSET58993.2023.10346621.
- [11] A. Anbarasi, S. Jagadeesan, and K. C. Nithyasree, 'Converting Structured Forms to Storable Databases using Form2DB', in *2023 International Conference on System, Computation, Automation and Networking, ICSCAN 2023*, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/ICSCAN58655.2023.10394829.
- [12] A. O. Taoufik and A. Azmani, 'AI-Enhanced Techniques for Extracting Structured Data from Unstructured Public Procurement Documents', in *8th International Symposium on Innovative Approaches in Smart Technologies, ISAS 2024 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2024. doi: 10.1109/ISAS64331.2024.10845583.
- [13] Tim Mucci, 'What is document AI?', IBM. Accessed: May 05, 2025. [Online]. Available: <https://www.ibm.com/think/topics/document-ai>
- [14] L. Cui, Y. Xu, T. Lv, and F. Wei, 'Document AI: Benchmarks, Models and Applications', Nov. 2021, [Online]. Available: <http://arxiv.org/abs/2111.08609>
- [15] S. Kastanas, S. Tan, Y. He, and E. Bv, 'Document AI: A Comparative Study of Transformer-Based, Graph-Based Models, and Convolutional Neural Networks For Document Layout Analysis'. Accessed: May 06, 2025. [Online]. Available: <https://arxiv.org/abs/2308.15517>
- [16] Google, 'Limits', Google Cloud. Accessed: May 05, 2025. [Online]. Available: <https://cloud.google.com/document-ai/limits>

REFERENCES

- [17] Anjanava Biswas, Chin Rane, and Sonali Sahu, 'Intelligent Document Processing with Amazon Textract, Amazon Bedrock, and LangChain', Aug. 2023, doi: 10.13140/RG.2.2.33137.90721.
- [18] S. Muddalkar, K. Kolte, A. Batra, A. Naphade, and N. Lokhande, 'Use of OCR Technology for Data Extraction Using Amazon Textract', *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, vol. 2, no. 3, 2022, doi: 10.48175/568.
- [19] 'Unstructured Open Source', Unstructured. Accessed: May 06, 2025. [Online]. Available: <https://docs.unstructured.io/open-source/introduction/overview>
- [20] Andrei Gheorghiu, *Building Data-Driven Applications with LlamaIndex: A practical guide to retrieval-augmented generation (RAG) to enhance LLM applications*. Packt Publishing, 2024.
- [21] Denis Rothman, *RAG-Driven Generative AI: Build custom retrieval augmented generation pipelines with LlamaIndex, Deep Lake, and Pinecone*. Packt Publishing, 2024.
- [22] B. Zirnstein, 'Extended context for InstructGPT with LlamaIndex', 2023, doi: 10.13140/RG.2.2.17701.31204.
- [23] 'EXTRACTA.AI - Introduction', Extracta.ai. Accessed: May 06, 2025. [Online]. Available: <https://docs.extracta.ai/>

POSTER

HO JOE EE 21ACB03160

FINAL YEAR PROJECT MAY 2025



Large Language Model (LLM) Application Integration for Extracting Unstructured Data



MOTIVATION

- The need for automated systems that can efficiently process unstructured data.
- The need to unlock LLMs' context understanding ability to extract implicit data from images, in which OCR systems lack.



PROBLEMS

- Lack of solidly established Image-to-Structured pipelines framework with LLMs Integration
- Inconsistency and Inaccuracy in Data Extraction by LLMs
- Inefficient resource utilisation for LLM-based system.

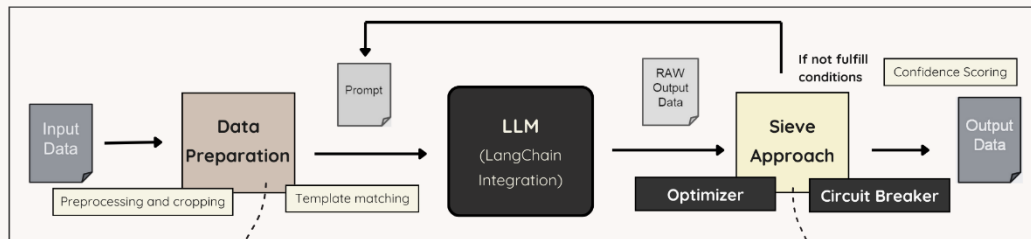


OBJECTIVES

- Development of a framework to ensure consistent and accurate data extraction, by reducing variability in LLM outputs to facilitate integration of LLMs in existing applications.
- Reduce resource consumption by and improving efficiency by 30%.



PROPOSED SOLUTION - AUTOMATED DATA PROCESSING PIPELINE



FOR OPTIMISATION

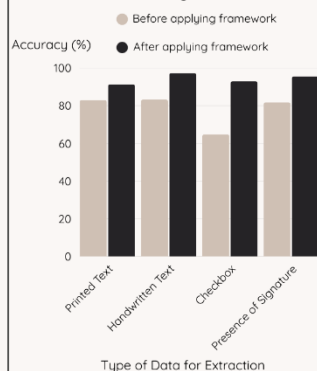
Data Preparation

- Image pre-processing
- Image quality check
- Image enhancement for better readability
- Template matching using OCR + fuzzy match or barcode
- Smart cropping to minimize token usage

Data Extraction and Analysis

- Integrated with LangChain for LLM orchestration
- Barcode data extraction
- Dual LLM Challenger for result comparison
- Referee logic to select the best output
- Confidence Scoring to assess data reliability
- Threaded processing with rate limiter for batch efficiency

Impact of Framework on Data Extraction Accuracy Across Data Types



NOVEL FRAMEWORK

- **Sieve** involves multiple iterations of validation, comparison, and refinement of the results.
- Performs regex check and separates the outputs of the LLM that do not conform to the required format for preprocessing.
- Generates dynamic prompt and ROIs of failed fields for better LLM focus.
- Exits loop via **Circuit Breaker** once values are validated, or max attempt is reached.
- Optimizes the max attempt and batch processing using **Optimizer** algorithm.

Outcome: Refined the accuracy, encouraging more consistent result.

Successfully addressed the critical challenge of LLM integration in automating the extraction of meaningful and structured information from unstructured data sources, with achievement in:



Overall Accuracy
96.93%



Processing Time
89.46% ↓



Token Usage
37.70% ↓



Cost
38.43% ↓

Final Deliverables:



As Terminal Module: Desktop app as a standalone tool for direct user interaction.



As Intermediate Module: Integrated in a real-time workflow with various sources via API.