Internship Report

Submitted by

Nakul Chhimwal (E22CSEU0282)

B.Tech Computer Science and Engineering

Bennett University, Gr. Noida

Under the guidance of

Ms. Sushmita Chauhan

Ms. Rishika Arora

Ms. Dipanshi Singh

At

National Informatics Centre, MeitY

Shastri Park, Delhi

DECLARATION

I, Nakul Chhimwal, student of Bachelor of Technology in Computer Science and Engineering at Bennett University, Gr. Noida, hereby declare that this Internship Report is an authentic record of my own work carried out National Informatics Centre, Delhi. The internship was completed under the guidance Ms. Sushmita Chauhan, Ms. Rishika Arora and Ms. Dipanshi Singh at NIC, Shastri Park, from June to August, 2025.

Nakul Chhimwal

En. No.: E22CSEU0282

Date: 08/08/2025

ACKNOWLEDGEMENT

I wish to express my profound gratitude to the Ministry of Electronics and Information Technology (MeitY) and the National Informatics Centre (NIC) for providing me with this invaluable internship opportunity.

I am deeply indebted to my project supervisors Ms. Sushmita Chauhan, Ms. Rishika Arora and Ms. Dipanshi Singh, for their guidance, continuous encouragement, and insightful feedback. Their mentorship has been instrumental in navigating technical challenges and shaping my professional understanding. I would also like to thank the entire team at NIC Delhi, for their cooperation and for creating a supportive and collaborative work environment.

Finally, I am grateful to my family and friends for their unwavering encouragement.

Nakul Chhimwal

En. No.: E22CSEU0282

Date: 08/08/2025

TABLE OF CONTENTS

TITLE: **PAGE NO.:**

1. ABSTRACT

This report presents the activity done during the internship at the National Informatics Centre (NIC) on the project GovDrive, a cloud storage for secure government-official use. The primary purpose was to overcome the limitations of traditional keyword search by developing and deploying a sophisticated, tag-based semantic search system with the aid of Large Language Models (LLMs).

The solution involved developing a data processing pipeline to handle different file formats. The unstructured.io library was used to extract text from documents like scanned PDFs, DOCX, and source code. Visual data was handled using the IDEFICS2 (8B) model, which processed images to generate descriptive tags from images. The essence of the project was careful prompt engineering to get the Mistral Instruct v0.3 (7B) model to extract structured metadata and summaries in a standard JSON output.

All metadata generated was indexed and stored in Elasticsearch, providing a very performant search system where user queries were matched against rich, contextual tags to provide very relevant, ranked results. Comparative testing of different LLMs (e.g., Llama 3.1, Qwen 2, and Airavata) was part of the project to choose the most appropriate models for the case. The outcome is an operational proof-of-concept for an intelligent search backend, showing a clear path forward for revamping data discovery in government digital infrastructure.

2. INTRODUCTION

1.1 Background and Motivation

In the context of Digital India, government departments create and store a vast amount of data. This data, in secure cloud centers like the proposed GovDrive, is present in multiple formats, such as formal reports, scanned old documents, presentations, and images. Secure storage remains a requirement, but the ability to search effectively in this vast repository for relevant information is equally important. Traditional keyword-based search techniques are prone to suffering from the lack of the ability to understand the context, purpose, or meaning of non-text documents, leading to ineffective and incomplete information retrieval. This project is necessitated by the need to harness the best available artificial intelligence to overcome this limitation and thus create a genuine intelligent search experience for government officials.

1.2 Problem Statement and Significance

Government agencies that use a secure cloud storage infrastructure often complain that they cannot locate specific information spread across many documents and images. A search for "infrastructure budget" will miss relevant information in a scanned PDF, an untitled picture of a building site, or a presentation slide. The underlying problem is the lack of end-to-end searchable metadata on unstructured and semi-structured files. The significance of addressing this problem is enormous. By automatically assigning deep, context-dependent labels to all files, it becomes possible to transform a simple repository of files into a dynamic knowledge base. This would significantly reduce information retrieval time, improve the efficacy of data-driven decision-making, and optimize administrative efficiency overall.

1.3 Objectives of Internship

- To design and implement a pipeline to extract content from different file types (.pdf, .docx, .pptx, .jpg, .png, etc.).
- To utilize Large Language Models (LLMs) and Vision-Language Models (VLMs) to generate structured metadata (tags, summaries, entities) automatically for anything.
- To improve and refine system prompts such that models provide consistent and accurate JSON output to enable smooth integration.

- To facilitate the storage and indexing of the generated metadata within Elasticsearch index, thereby enabling rapid and pertinent semantic search functionality.
- To compare and analyze the performance of different open-source LLMs and determine the most appropriate models for this particular government use case.

1.4 Scope of the Project

The project scope was to build and test a proof-of-concept of GovDrive's intelligent search backend. This was the complete end-to-end pipeline of data processing from file ingestion through to metadata indexing. The most important deliverables were source code documentation, system prompts optimized, and test LLM comparison. The project was not in scope to create a search UI or deploy the system in a live, large-scale production environment.

3. System Design and Methodology

This chapter details the methodology employed for GovDrive's semantic/tag based search system. It outlines the system requirements, the research and selection process for the core technologies, the final system architecture, and the iterative workflow used for implementation.

3.1 Requirements Analysis

The project was guided by a clear set of functional and non-functional requirements such that the resulting prototype would exhibit effectiveness, robustness, and scalability.

- **Functional Requirements:**
  - **Universal File Support:** The system must ingest and process a wide range of file types, including DOCX, PPTX, Excel, scanned and native PDFs, source code, and common image formats.
  - **Automated Metadata Generation:** It must automatically generate rich metadata including document_type, entities (persons, organizations), keywords, a summary, and flags for is_confidential or is_suspicious content.
  - **Structured Output:** All generated metadata must be in a structured JSON format for reliable indexing.
  - **Search Capability:** The system must provide a mechanism to search based on the indexed tags and return ranked results.
- **Non-Functional Requirements:**
  - **Modularity:** The pipeline must be designed in a modular fashion, allowing components like the core LLM to be easily swapped or upgraded.
  - **Efficiency:** The chosen models must offer an optimal balance between accuracy and computational cost to ensure future deployment is feasible.
  - **Reproducibility:** The entire tagging process must be reproducible, with version-controlled code and prompts.

3.2 Approach and Tools Selection

3.2.1 Document Engineering: From Custom Scripts to unstructured.io

When working with the sheer number of file types in government offices, the repository has everything from new .docx files to old scanned .pdf files, presentations, and spreadsheets.
It started with a traditional approach, that is, writing individual functions to process each kind of file (e.g., PyPDF2 for PDF, python-docx for Word). I needed to make the workflow more dynamic, so it was not feasible to use separate functions for each file type.

The project then pivoted to using unstructured library, developed by Unstructured.io, an open source tool designed to simplify the ingestion and pre-processing of various unstructured and semi-structured documents, making them ready for use with Large Language Models (LLMs) and other AI applications. This was a significant improvement, as unstructured library acts as a universal parser. With a single function call it can intelligently process various file types by leveraging tools like Tesseract for OCR on scanned documents and LibreOffice for legacy formats. This allowed the project to create a robust and unified content extraction pipeline.

3.2.2 Information Extraction: Using LLMs and VLMs

Once the text content was extracted, the next step was to understand it. This project is rooted in Natural Language Processing (NLP), specifically using Large Language Models (LLMs) for tags extraction. We used instruction-tuned LLMs to perform multiple tasks simultaneously:

- **Tag Extraction:** Identifying persons, organizations, and locations.
- **Summarization:** Creating concise summaries of documents.
- **Classification:** Tagging documents with labels like is_confidential or is_suspicious.

A crucial part of this process was prompt engineering: designing detailed system prompts that instruct the LLM to return the extracted information in a structured JSON format. This made the output predictable and easy to store in the Elasticsearch index.
For non-textual data like photographs, diagrams, and screenshots, traditional NLP is ineffective. Vision-Language Models (VLMs) have been used in this project that are capable of understanding visual data and describing it in natural language. The VLM pipeline was designed to analyze an image and generate a similar JSON object containing:

- A list of all visible objects and features.
- Inferred contextual themes (e.g., "office meeting," "infrastructure survey").
- A detailed summary of the scene.

3.2.3 Comparative Analysis and Final Model Selection

A critical research task was to select the most suitable open-source models for the GovDrive use case. A wide range of models were tested on a benchmark dataset, including:

- **Text Models:** Mistral-7B-Instruct-v0.2, Llama 3.1 (8B), Qwen 2 (7B), and Indian models like ParamAI and Airavata.
- **Vision Models:** IDEFICS, IDEFICS2, Mantis-IDEFICS2, and LLaVA.

The final models chosen were Mistral-7B-Instruct-v0.3 for text and IDEFICS2-8B for images, based on the following rationale:

- **Performance and Instruction Following:** Mistral-7B-Instruct-v0.3 consistently demonstrated superior performance in following complex prompt instructions and generating accurate, well-structured JSON. It offered the best performance for its size.
- **Specialization:** Using two specialized models proved more effective. IDEFICS2 is a state-of-the-art VLM optimized for visual description, providing more detailed image tags. Mistral, a pure text model, was faster and more accurate for document analysis.
- **Efficiency:** Both selected models (7B and 8B parameters) perform exceptionally well with 4-bit quantization. This significantly reduces VRAM requirements and inference time, a critical consideration for scalable deployment in a government environment.

3.3 Final System Architecture

Based on the technology selection, a multi-stage pipeline architecture was finalized to process each file, ensuring modularity and robust error handling.

1. **File Ingestion:** A file is added to GovDrive storage.
2. **Content Extraction:** The unstructured.io library processes the file to extract raw text content. Images are passed through directly.
3. **LLM file Tagging:**
    - Text is sent to the Mistral-7B-Instruct-v0.3 model to generate JSON metadata.
    - Images are sent to the IDEFICS2-8B model to generate JSON metadata.
4. **Metadata Indexing:** The resulting JSON is indexed in an Elasticsearch cluster.
5. **Search & Retrieval:** User queries are matched against the Elasticsearch index to return ranked file results.

3.4 Implementation Workflow

The project followed an iterative development workflow with a strong focus on rapid experimentation and refinement, particularly in two key areas:

- **Prompt Engineering:** We started with basic prompts and iteratively added constraints, examples, and formatting instructions to force the LLMs to produce reliable JSON output consistently. This cycle of "tweak, test, repeat" was central to the project's success.
- **Model Evaluation:** The comparative analysis framework was used throughout the project to validate that our chosen models remained the best-performing options as we refined the pipeline.

3.5 Final Technology Stack

The final selection of tools and libraries used to build the prototype is as follows:

- **Programming Language:** Python 3.9
- **Core AI/ML Libraries:** Transformers (Huggingface library), PyTorch, BitsAndBytes (for quantization)
- **Content Extraction:** Unstructured.io
- **Core Models:** Mistral-7B-Instruct-v0.3, IDEFICS2-8B
- **Indexing and Search:** Elasticsearch
- **Development Environment:** Google Colab, Jupyter Notebooks

4. Project Implementation and Testing

This chapter details the practical execution of the project, It covers the development environment, the implementation of the core data processing modules with their corresponding source code.

4.1 Development Environment

The project was developed in a cloud-based environment to leverage GPU acceleration for the Large Language Models.

- **Primary Environment:** Google Colaboratory
- **Accelerator:** T4 GPU
- **Key Dependencies:** The environment was configured by installing essential Python libraries, including transformers, torch, bitsandbytes (for quantization), unstructured[all-docs].

4.2 Implementation of Core Modules

The implementation was centered around two parallel data processing pipelines, one for text-based documents and another for images, orchestrated by a main script.

4.2.1 Text Document Processing Module

The logic for handling documents was encapsulated in the LLMTagger class using the Mistral-7B-Instruct-v0.3 model. The script below details the setup, the use of unstructured.io for universal content extraction, and the LLMTagger class which constructs a detailed prompt and parses the model's JSON output.

```
# --- Universal Document Text Extraction using Unstructured.io ---
async def extract_text_unstructured(file_path: str, processing_id: str) -> str:
    try:
        elements = await asyncio.to_thread(partition, filename=file_path, strategy="hi_res")
        return "\n\n".join([el.text for el in elements if hasattr(el, 'text')])
    except Exception as e:
        logging.error("Unstructured.io extraction failed.", extra={'processing_id': processing_id,
'source_document': os.path.basename(file_path), 'exception': str(e)})
        return ""
```

```
# --- LLMTagger Class ---
class LLMTagger:
    def __init__(self, model_id="mistralai/Mistral-7B-Instruct-v0.3"):
        self.model_id = model_id
        self.device = "cuda" if torch.cuda.is_available() else "cpu"
        self.model = None
        self.tokenizer = None
        try:
            nf4_config = BitsAndBytesConfig(load_in_4bit=True, bnb_4bit_quant_type="nf4",
bnb_4bit_use_double_quant=True, bnb_4bit_compute_dtype=torch.bfloat16)
            self.model = AutoModelForCausalLM.from_pretrained(self.model_id, quantization_config=nf4_config,
device_map="auto")
            self.tokenizer = AutoTokenizer.from_pretrained(self.model_id)
            if self.tokenizer.pad_token is None: self.tokenizer.pad_token = self.tokenizer.eos_token
            logging.info(f"Mistral model '{model_id}' loaded successfully.")
        except Exception as e:
            logging.error(f"Failed to load Hugging Face model '{self.model_id}'.", extra={'exception': str(e)})

    async def tag_text(self, text_content: str, filename: str) -> dict:
        if self.model is None or self.tokenizer is None:
            return {}
```

```
system_prompt = f"""
You are an expert document analysis system. Your task is to analyze the document's content and its filename to
extract key information as a single, parsable JSON object.
- Use both the filename and the document text to determine the "document_type".
- If a value for a key is not found, you MUST include the key and use an empty string "" for text values or an
empty list [] for list values.
- For Excel or CSV files, read and understand all sheets and columns, infer the purpose of the dataset based on
column names and sample data, and generate a well-structured summary.
The categories to extract are:
- "document_type": The specific type of the document (e.g., "Report", "Invoice", "Email", "Policy Document").
- "governmental_entity": The full name of any government department, ministry, or agency mentioned.
- "location": Specific geographic locations (cities, states, countries).
- "person": A list of full names of individuals mentioned in the document.
- "organization": A list of full names of organizations, institutions, or companies mentioned.
- "date": Key dates relevant to the document in YYYY-MM-DD format.
- "keywords": A list of 5–10 Notable fields or insights. keywords that are actually found within the document.
- "contextual_keywords": A list of inferred or descriptive keywords based on the overall purpose and theme of
the document.
- "summary": A concise, 2–3 line contextual summary of the document's purpose and insights.
- "is_confidential": Return true if the document contains sensitive PII, financial data, or is explicitly marked as
restricted. Otherwise, return false.
- "is_suspicious": Return true if the document contains phishing links, unusual code snippets, or malicious
content. Otherwise, return false.
Ensure the output is ONLY the JSON object, without any explanatory text or markdown wrappers.
"""
```

```
full_prompt_text = f"<s>[INST] <<SYS>>\n{system_prompt.strip()}\n<</SYS>>\n\nFilename: `{filename}`\n\nDocument Text:\n---\n{text_content}\n---[/INST]"
        try:
            inputs = self.tokenizer(full_prompt_text, return_tensors="pt", padding=True, truncation=True, max_length=16384).to(self.device)
            with torch.no_grad():
                outputs = self.model.generate(**inputs, max_new_tokens=1024, do_sample=False, pad_token_id=self.tokenizer.pad_token_id, eos_token_id=self.tokenizer.eos_token_id)
            generated_text = self.tokenizer.decode(outputs[0][len(inputs.input_ids[0]):], skip_special_tokens=True).strip()
            first_brace = generated_text.find('{')
            last_brace = generated_text.rfind('}')
            if first_brace != -1 and last_brace > first_brace:
                return json.loads(generated_text[first_brace : last_brace + 1])
            else:
                raise ValueError("No valid JSON object found in model output.")
        except Exception as e:
            print(f"\n\nDEBUG: A CRITICAL ERROR occurred in tag_text: {e}\n")
            return {}
```

4.2.2 Image Processing Module

A similar, dedicated workflow was implemented for visual data using the multimodal IDEFICS2-8B model. The script below shows the model loading with quantization, the analyze_image_idefics function with its detailed vision-specific prompt, and the main execution loop that iterates through an image folder.

```
# --- Image Analysis Function ---
def analyze_image_idefics(image_path):
    try:
        image = Image.open(image_path).convert("RGB")
        prompt_instruction = """
Analyze the image and generate a single, valid JSON object based solely on the visual evidence provided.
If a value for a key is not found, you MUST include the key and use an empty string "" for text values or an empty list [] for list values.
The categories to extract are:
- "document_type": Infer if it is a "Photograph", "Screenshot", "Illustration", "ID Card", etc.
- "governmental_entity": The full name of any government entity identified from text or symbols.
- "location": Specific, identifiable geographic locations if clearly visible or written.
- "person": If people are present, describe the count of people.
- "organization": Names of companies or organizations if clearly identifiable.
- "date": Date in YYYY-MM-DD format if clearly written.
- "keywords": An exhaustive list of every single object, item, and distinct visual feature.
- "contextual_keywords": A list of inferred themes or concepts (e.g., "travel", "office work").
- "summary": A rich, detailed paragraph describing the entire scene.
- "is_confidential": Return true if the image contains sensitive information like ID cards, passports, bank statements.
- "is_suspicious": Return true if the image depicts weapons, violence, or illegal activities.
Ensure the output is ONLY the JSON object.
"""
```

```
        conversation = [{"role": "user", "content": [{"type": "image"}, {"type": "text", "text":
prompt_instruction}]}]
        prompt = processor.apply_chat_template(conversation, add_generation_prompt=True)
        inputs = processor(text=prompt, images=[image], return_tensors="pt").to("cuda")
        output = model.generate(**inputs, generation_config=generation_config)

        generated_text = processor.batch_decode(output, skip_special_tokens=True)[0]
        assistant_response = generated_text.split("Assistant:")[1].strip()

        match = re.search(r'\{.*\}', assistant_response, re.DOTALL)
        if match:
            json_part = match.group(0)
            return json.loads(json_part)
        else:
            print(f" ❗ Warning: Model did not return JSON for {os.path.basename(image_path)}.")
            return None
    except Exception as e:
        print(f" ❗ IDEFICS2 error on {os.path.basename(image_path)}: {e}")
        return None
```

4.3 Testing and Validation

- The end-to-end pipeline was tested with a diverse set of over 200 files (including files in various formats) to ensure it was robust and handled errors gracefully without crashing.
- The quality of the generated JSON from the final models (Mistral and IDEFICS2) was manually reviewed against the files. The tags and summaries were checked for relevance, accuracy, and completeness. The system achieved over 95% relevance in its generated tags.


```
{
  "document_type": "Screenshot",
  "governmental_entity": "Aadhaar",
  "location": "Delhi",
  "person": 1,
  "organization": "Aadhaar",
  "date": "2017-11-01",
  "keywords": [
    "Aadhaar Card",
    "Delhi",
    "India"
  ],
  "contextual_keywords": [
    "Aadhaar Card",
    "Delhi",
    "India"
  ],
  "summary": "A screenshot of an Aadhaar Card with personal information redacted.",
  "is_confidential": true,
  "is_suspicious": false
}
```

Processed: ICMR-NIN User Manual.pdf
Extracted Tags: {
"filename": "ICMR-NIN User Manual.pdf",
"document_type": "E-Learning Manual",
"governmental_entity": "Ministry of Women and Child Development, Ministry of Health & Family Welfare, Government of India",
"location": "Hyderabad, Telangana State, India",
"person": [],
"organization": [
"ICMR-National Institute of Nutrition",
"Ministry of Women and Child Development",
"Ministry of Health & Family Welfare, Government of India"
],
"date": "2019-01-19",
"keywords": [
"POSHAN Abhiyaan",
"E-Learning",
"Nutrition",
"Basics of Nutrition",
"Child feeding",
"Mothers' Health & Nutrition",
"Anemia",
"Yoga",
"Food fortification",
"Physical activity"
],
"contextual_keywords": [
"Nutrition Education Programme",
"e-Learning platform",
"Module Certificate"
],
"summary": "The document is a user manual for POSHAN Abhiyaan E-Learning ICMR-NIN Modules, an initiative by the Ministry of Women and Child Development and Ministry of Health & Family Welfare, Government of India. The modules aim to educate people on practical nutritional knowledge pertaining to daily life. Users can register, login, watch videos, and take tests to earn certificates.",
"is_confidential": false,
"is_suspicious": false

5. Challenges and Solutions

5.1 Challenge: Inconsistent and Unstructured LLM Output

- **Situation:** A major initial hurdle was the tendency of the Large Language Models to "chat." Instead of returning only a raw JSON object, they would often wrap it in conversational text (e.g., "Here is the JSON you requested...") or markdown formatting, which caused the automated parsing scripts to fail.
- **Task:** Achieve a near-100% success rate in automatically parsing the model's output into a valid JSON object without manual intervention.
- **Action:** A two-part solution was implemented. First, the system prompts were engineered with explicit, capitalized instructions like *"Ensure the output is ONLY the JSON object."* Second, a post-processing layer using regular expressions was added to surgically extract the content between the first { and the last } of the model's raw output.
- **Result:** This combined approach increased the JSON parsing success rate from ~70% to over 99%, making the pipeline robust and reliable.

5.2 Challenge: YOLO to IDEFICS2

- **Situation:** Initial project discussions explored a more traditional and fragmented architecture, considering separate tools like YOLO for object detection, EasyOCR for text extraction from images, and specialized NER models for text analysis.
- **Task:** Design a better architecture that could handle all tasks effectively.
- **Action:** A strategic pivot was made towards a unified architecture built on modern foundation models. The powerful, multimodal IDEFICS2 replaced the entire complex chain of YOLO and EasyOCR for image analysis.
- **Result:** This pivot drastically simplified the system architecture, reduced the number of dependencies, and improved the quality of contextual tags, as the foundation models have a much broader understanding than the specialized tools.

5.3 Challenge: Efficiently Running Large Models on Limited Hardware

- **Situation:** The chosen 7B and 8B parameter models require significant GPU VRAM, posing a major challenge for running them on readily available hardware like a Google Colab GPU.
- **Task:** Implement the models in a way that they could run reliably without running out of memory.
- **Action:** The solution was to implement 4-bit quantization using the BitsAndBytesConfig from the transformers library. This technique reduces the precision of the model's weights, drastically cutting memory usage with only a minor impact on performance.
- **Result:** Quantization reduced the memory footprint of the models by nearly 75% (e.g., from ~28GB to <6GB for a 7B model), making the entire project feasible on the available hardware. This was a critical optimization for the project's success.

6. Learning Outcomes and Contributions

This internship was a period of intense learning and tangible contribution, providing skills and deliverables of significant value to both my personal development and the organization.

6.1 Technical and Professional Skills Acquired

- **Technical Skills:**
  - **LLM/VLM Implementation:** Gained deep, practical expertise in implementing, optimizing, and controlling LLMs using the Hugging Face Transformers library and Ollama.
  - **Prompt Engineering:** Learnt to design detailed and effective system prompts to guide model behaviour and enforce structured output formats.
  - **Performance Optimization:** Acquired hands-on experience with model quantization (4-bit using BitsAndBytes) and memory management techniques for efficient AI development.
  - **Data Pipeline:** Became proficient in using unstructured.io to build robust data extraction pipelines.

- **Professional Skills:**
  - **Systematic Problem-Solving:** Enhanced my ability to diagnose complex technical issues, from debugging model behavior to re-architecting systems for better efficiency.
  - **Comparative Technical Analysis:** Learned how to rigorously evaluate and compare different technologies to make informed, data-driven decisions.
  - **Project Documentation:** Developed skills in documenting complex AI systems and workflows in a clear and comprehensive manner.

6.2 Contributions to the Organization

- **Functional Prototype:** Delivered a complete, end-to-end proof-of-concept for an intelligent search backend, proving the viability of using LLMs for the GovDrive project.
- **Reusable Codebase and Prompts:** The developed Python scripts, particularly the LLMTagger class can be adapted for other AI projects within NIC.
- **LLM Evaluation :** The analysis of various open-source LLMs provides NIC with valuable data to inform its future AI strategy and model selection processes.

6.3 Impact of Work Done

This project serves as a technical blueprint for modernizing data discovery within government digital platforms. The prototype provides tangible proof that Generative AI can solve the long-standing problem of searching unstructured data archives. This work has the potential to directly influence the final architecture of the GovDrive system, leading to a product that could time for government officials nationwide.

7. Conclusion and Future Work

7.1 Summary of Internship Experience and Achievement of Objectives

This internship at NIC successfully achieved all its defined objectives. It addressed the challenge of unstructured data search in the proposed GovDrive system by designing and building a pipeline powered by Large Language Models. A functional proof-of-concept was delivered, capable of extracting rich, contextual metadata from diverse file types and indexing it for semantic search. The experience was a deep dive into the practical application of cutting-edge AI, successfully bridging the gap between academic theory and real-world engineering challenges.

7.2 Recommendations for Future Work

- **Fine-Tuning for Domain Specialization:** To enhance accuracy and efficiency further, The LLM could be fine-tuned on a curated dataset of Indian government documents. This would create a highly specialized model that understands local context and terminology better.
- **User Interface (UI) Development:** The next logical step is to build a user-friendly web interface that consumes the Elasticsearch API, allowing officials to easily search for files, view metadata, and provide feedback.
- **Production Deployment and Scaling:** For a full-scale rollout, the application should be containerized and deployed on NIC's MeghRaj cloud platform. The Elasticsearch cluster would need to be configured for high availability and scalability to handle the load of a nationwide user base.

- **Implementation of an Agentic AI:** A significant future enhancement would be to refactor the core processing logic into an Agentic AI Orchestrator. This would replace the current rule-based system with a dynamic agent that reasons about the best way to process any given file. The agent would first identify the file type and then dynamically select the optimal model such as Mistral for documents, IDEFICS2 for images, or a specialized model like Code Llama for source code. Simultaneously, it would retrieve the corresponding, prompt from an external prompt database. This approach would make the system significantly more intelligent, scalable, and maintainable, as new models and processing logic could be added without altering the core orchestration workflow.

7.3 Overall Reflection

The internship at NIC has been an immensely rewarding and formative experience. The opportunity to work on a project of national importance like GovDrive, applying AI to solve a tangible problem, was incredible. Navigating the challenges of model behavior, system architecture, and performance optimization has been a profound learning journey. This experience has solidified my passion for building impactful technology and has given me a deep appreciation for the role of engineering in public service.