LLM & GEN AI

Project Report

Intelligent Document Processing with Multimodal Generative AI

Charles Perier - Antoine Richard - Romain Paupe

MEng – Data & AI Major | Group DIA 3
ESILV – Pôle Léonard de Vinci

GitHub Repo : `tcboris/LLM-GenAI-Project--DIA-A5`

Project: Number 2
Submitted on : December 18, 2025

Abstract

This report details the design, implementation, and qualitative validation of an Intelligent Document Processing (IDP) platform. The system is designed to automate the extraction of structured data from physical documents, specifically focusing on supplier invoices and wine labels. The architecture integrates an open-source OCR engine (EasyOCR) with a Large Language Model (Google Gemini 2.5 Flash Lite) within a modern web framework (Next.js 16 and FastAPI). The platform supports both single-file and sequential batch processing, deployed locally via Docker. This paper discusses the technical challenges encountered—such as OCR accuracy variability, data structuring without regex, and Cross-Origin Resource Sharing (CORS) issues—and presents the architectural solutions implemented to address them. While no formal quantitative metrics (CER/WER) were established for this project, functional validation confirms the system's capability to categorize documents and extract complex entities, offering a viable foundation for low-cost, scalable document digitization.

Contents

1 Introduction

1.1 Context and Motivation

In the contemporary enterprise landscape, the management of physical documents remains a significant bottleneck. Despite digital transformation efforts, volumes of paper-based invoices and inventory logs persist, necessitating manual data entry. This manual transcription is not only time-consuming but also prone to human error, leading to data inconsistencies and difficulties in subsequent information retrieval.

The motivation for this project lies in the democratization of Intelligent Document Processing (IDP). By combining Optical Character Recognition (OCR) with the semantic understanding capabilities of Large Language Models (LLMs), it is possible to automate the extraction of structured data from unstructured images. This project targets two specific use cases:

- **Accounting:** Automatic extraction of supplier invoices, including specific fields such as invoice number, date, total amount, and detailed vendor information.
- **Wine Cellar Management:** Digitalization of wine bottle labels to extract semantic details like vintage, appellation, alcohol content, and producer names.

1.2 Objectives

The primary objective of this project was to develop a functional, end-to-end pipeline capable of transforming raw images into structured JSON output. Key technical goals included:

- Implementing a hybrid pipeline using EasyOCR for text detection and Google Gemini for semantic structuring.
- Supporting both single-file uploads and sequential batch processing to ensure scalability.
- ensuring local deployability via Docker to remove mandatory cloud dependencies for the backend infrastructure.
- Creating a modern, responsive user interface with real-time feedback mechanisms.
- Solving architectural challenges related to browser security policies (CORS) and API key management.

2 Problem Description

2.1 Technical Challenges

Developing a robust document intelligence platform involves overcoming several distinct technical hurdles.

2.1.1 Challenge 1: OCR Accuracy

The variability in input image quality poses a significant challenge. Documents range from high-resolution scans to photographs taken with smartphones under poor lighting conditions or skewed angles. This necessitates an OCR engine capable of handling noise and varying contrast levels. We selected EasyOCR for its multilingual support (French/English) and robustness compared to legacy tools like Tesseract on complex backgrounds.

2.1.2 Challenge 2: Data Structuring

Raw text extracted by OCR lacks semantic structure. For example, the string "TOTAL 125.50 EUR" is merely a sequence of characters. Converting this into a structured object such as `{total_amount: 125.50, currency: "EUR"}` is difficult with traditional Regular Expressions (Regex) due to the infinite variations in document layouts. With wine labels, it gets even harder to contextualize, as key information is often scattered across stylized fonts or curved surfaces. For instance, a string like "2018 Saint-Émilion Grand Cru 13.5%" requires the system to distinguish between the vintage (2018), the appellation (Saint-Émilion Grand Cru), and the alcohol content (13.5%), a task where simple pattern matching fails to understand the specialized vocabulary and hierarchy of oenological data.

This project leverages an LLM to perform contextual extraction rather than pattern matching.

2.1.3 Challenge 3: Document Classification

The system must automatically distinguish between document types (e.g., an invoice versus a wine label) to apply the correct extraction schema. Hard-coded rules fail when layouts differ significantly between issuers (e.g., an EDF invoice vs. a restaurant bill).

2.1.4 Challenge 4: Batch Processing

Processing multiple documents in a single session introduces state management complexities. The system must track the status (pending, processing, success, error) of each file individually and manage memory resources to effectively prevent browser or server crashes.

2.1.5 Challenge 5: CORS Issues

A separation of concerns between the Next.js frontend and the Python backend leads to Cross-Origin Resource Sharing (CORS) blocks when the browser attempts to contact the API directly.

2.2 Scope and Constraints

The scope of this project is defined as follows:

- **Input:** Printed documents in French or English.
- **Formats:** Images (JPG, PNG, WEBP).
- **Domain:** Invoices and Wine Labels.

Out of Scope: Handwritten text, multi-page PDFs, and complex multi-column tables are not supported in the current version. Furthermore, the system relies on the Gemini Flash model to balance latency and cost, accepting a trade-off against larger, more expensive models.

3 Architecture

3.1 System Overview

The system follows a microservices architecture separated into a frontend and a backend, containerized via Docker.

The data flow is as follows:

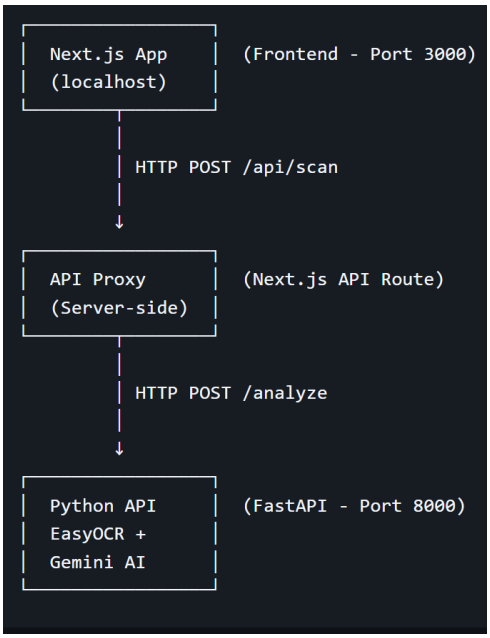

Figure 1: High-Level System Architecture

1. The user uploads an image via the Next.js frontend (Port 3000).
2. The request is sent to a Next.js API Route (acting as a proxy).
3. The proxy forwards the request to the Python FastAPI backend (Port 8000).
4. The backend processes the image with EasyOCR and sends the text to Google Gemini.
5. Structured JSON is returned to the user.

3.2 Frontend Architecture (Next.js 16)

The frontend is built using Next.js 16 with the App Router and Server Components.

- **Language:** TypeScript is used for type safety and robust development.
- **UI Components:** Radix UI provides accessible primitives, styled with Tailwind CSS.
- **State Management:** React Hooks manage the complex state of file uploads and batch processing progress.

3.3 Backend Architecture (Python + FastAPI)

The backend is an asynchronous API built with FastAPI.

- **Core Components:** EasyOCR (PyTorch-based) for text extraction and the Google GenAI SDK for LLM interaction.
- **Image Processing:** OpenCV and NumPy are used for image loading and array manipulation.
- **Server:** Uvicorn serves the application as an ASGI server.

3.4 API Proxy Design Pattern

To resolve CORS issues without exposing API keys or complicating backend headers, we implemented an API Proxy pattern. The frontend does not call the Python backend directly. Instead, it calls a local Next.js API route, which performs a server-to-server request to the Python container. Here is a simplified version of the code :

```
// app/api/scan/route.ts
export async function POST(request: NextRequest) {
    const formData = await request.formData();
    const apiUrl = request.headers.get("x-api-url");

    // Forward to Python backend (Server-to-Server)
    const response = await fetch(apiUrl, {
        method: "POST",
        body: formData
    });

    return NextResponse.json(await response.json());
}
```

Listing 1: Next.js API Proxy Implementation

4 Pipeline Implementation

4.1 OCR Stage (EasyOCR)

We selected EasyOCR for its open-source license (Apache 2.0) and ability to run on CPU, which is crucial for the portability of our Docker container. The configuration loads English and French models into memory (∼1GB RAM).

The extraction process involves reading the image byte stream into OpenCV, converting it to a NumPy array, and passing it to the ‘reader.readtext‘ function. The resulting list of text strings is concatenated into a single "raw text" block.

4.2 LLM Stage (Google Gemini)

The raw text is passed to Google Gemini 2.5 Flash Lite. We utilize **Few-Shot Prompting** and strict output formatting instructions to ensure the model returns valid JSON.

```
prompt = f"""
You are an expert data extraction assistant.
Analyze this raw OCR text: "{raw_text}"

Determine if it is 'Invoice' or 'Wine'.

If INVOICE, extract (JSON):
- type: "Invoice"
- date (DD/MM/YYYY)
- total_amount
- vendor (object with name, siret, address)

If WINE, extract (JSON):
- type: "Wine"
- name, vintage, appellation, alcohol_degree

Reply ONLY with valid JSON.
"""
```

Listing 2: Prompt Engineering Strategy

The backend includes a parsing utility to strip Markdown code blocks (e.g., "‘`json`) often added by LLMs, ensuring the response is parseable by the standard `json` library.

4.3 Batch Processing Implementation

To handle scalability without crashing the server or hitting API rate limits, we implemented a **Sequential Batch Processing** strategy.

Instead of firing all requests in parallel, the frontend iterates through the file list. It awaits the completion of one request before starting the next. This updates the UI state granularly (Pending → Processing → Success), providing clear feedback to the user.

5 Functional Validation

5.1 Validation Approach

The system was validated against a varied set of documents, including:

- **Invoices:** Various utility bills (energy, telecom) and receipts with different layouts.
- **Wine Labels:** Bottles with varying label shapes, fonts, and contrasting backgrounds.
- **Image Quality:** Ranging from high-resolution scans to smartphone photos.

5.2 Observed Behavior

5.2.1 OCR Performance

EasyOCR performed robustly on clear, printed text on high-contrast backgrounds. However, limitations were observed with:

- **Curved Text:** Labels wrapping around wine bottles occasionally resulted in fragmented text detection.
- **Handwriting:** As expected, handwritten annotations on invoices were either ignored or garbled.
- **Low Contrast:** Grey text on white backgrounds was occasionally missed.

5.2.2 LLM Structuring

The classification capability of the LLM was observed to be highly reliable, correctly distinguishing between invoices and wine labels in almost all test cases. The extraction of complex nested fields (e.g., Vendor Address containing City and Zip Code) was significantly superior to traditional regex methods. The model successfully normalized different date formats (e.g., "12 Dec 2023" to "12/12/2023").

5.2.3 User Experience

The sequential batch processing mode was tested with batches of 5 to 10 files. The system remained stable, and the progress table provided necessary visibility into the process, allowing users to identify which specific files failed without disrupting the entire batch.

6 Implementation Results & Discussion

6.1 Delivered Functionality

The project successfully delivered a containerized web application meeting the initial requirements.

- **Core Features:** Single file upload, Drag & Drop, Batch processing, and JSON Export.
- **Rich Extraction:** For invoices, over 15 distinct data points are extracted (including detailed vendor tax IDs). For wine, over 8 semantic fields are retrieved (including food pairing suggestions derived by the LLM's internal knowledge).

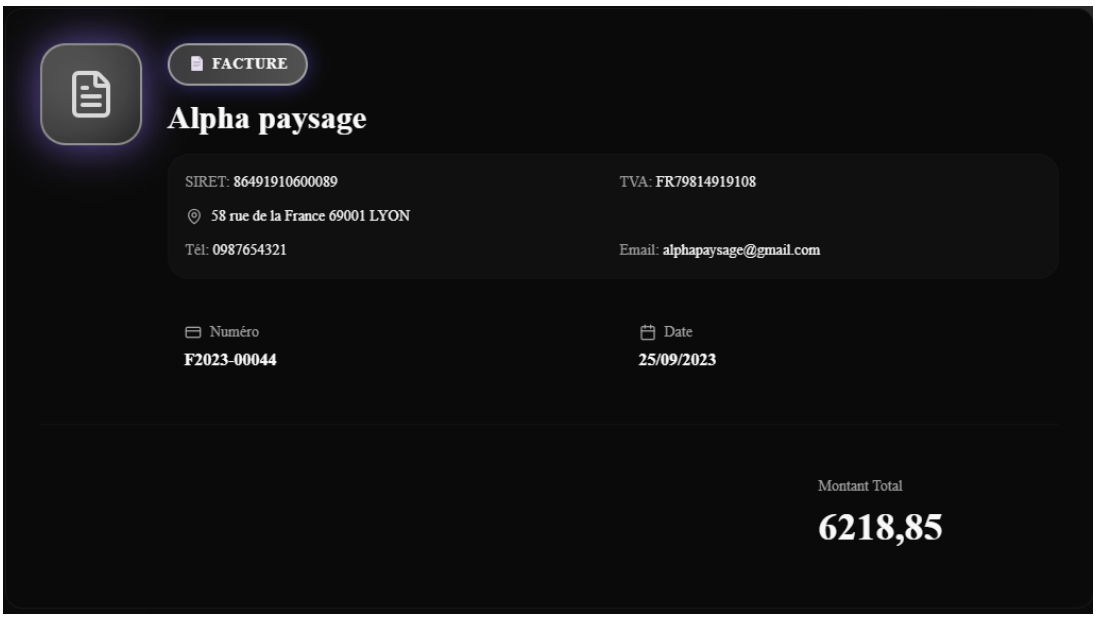

Figure 2: UI showing structured extraction results for an invoice

6.2 Technical Insights

LLM vs. Regex: The use of Gemini confirmed that LLMs offer superior flexibility. The model handles layout variations implicitly. For instance, "Total Amount" might appear at the top or bottom of a document; the LLM uses semantic context to identify it, whereas regex would require specific positional logic for every vendor template.

Prompt Engineering: We observed that the quality of the prompt is the single most critical factor in extraction accuracy. Moving from a vague instruction ("Extract data") to a strict JSON schema definition improved consistency dramatically and reduced "hallucinations."

TypeScript Strictness: Implementing strict typing in the frontend was challenging but beneficial. It forced the handling of edge cases, such as missing fields (returning `null` instead of crashing the UI), resulting in a more robust application.

6.3 Current Limitations

- **Cloud Dependency:** The system requires an internet connection to reach the Google Gemini API.

- **Performance:** Processing time is roughly 3-10 seconds per image, which is acceptable for back-office tasks but potentially too slow for real-time, high-volume industrial scanning.
- **Image Preprocessing:** The current pipeline lacks automatic deskewing or contrast enhancement, meaning poor quality inputs lead to poor OCR results ("Garbage In, Garbage Out").

7 Future Work

To transition this prototype into a production-grade system, several developments are proposed:

7.1 Short-term (1-3 Months)

- **PDF Support:** Integrate `PyPDF2` to convert multi-page PDFs into images for processing.
- **Preprocessing Pipeline:** Implement OpenCV filters for deskewing, noise reduction, and binarization to improve OCR raw accuracy.

7.2 Mid-term (3-6 Months)

- **Local LLM:** Replace Gemini with a local model like Ollama (Llama 3.2 Vision) to enable offline processing and data privacy.
- **Validation Layer:** Add post-processing logic to validate extraction (e.g., verifying that Total Amount = Net + Tax).

7.3 Long-term

- **Training Dataset:** Create a ground-truth dataset of 100+ documents to perform formal quantitative evaluation (WER/CER).
- **Active Learning:** Implement a feedback loop where user corrections on the frontend are saved to retrain or fine-tune the model.

8 Conclusion

This project demonstrated the feasibility of building a modern Intelligent Document Processing platform using open-source tools and accessible LLM APIs. By integrating EasyOCR with Google Gemini within a Dockerized architecture, we achieved a flexible system capable of structuring data from diverse physical documents.

The key technical lessons learned highlight the importance of prompt engineering over complex coding for data extraction, and the necessity of robust architectural patterns (like API proxies) in modern web development. While current limitations regarding execution speed and image quality dependence exist, the proposed architecture provides a solid foundation for future enhancements, particularly the transition towards fully local, privacy-preserving AI models.

A Code Samples

A.1 Backend: OCR and LLM Pipeline

```
# ---------------- CONFIGURATION ----------------
# On r cup re la cl  API depuis les variables d'environnement Docker
GOOGLE_API_KEY = os.getenv("GOOGLE_API_KEY")

if not GOOGLE_API_KEY:
    print("       WARNING: GOOGLE_API_KEY non d finie !")
else:
    genai.configure(api_key=GOOGLE_API_KEY)

# Initialisation des mod les (Au d marrage pour  viter  de recharger    chaque requ te)
print("    Chargement du mod le OCR...")
reader = easyocr.Reader(['en', 'fr'], gpu=False) # Mettre gpu=True si vous avez configur  NVIDIA Docker
print("    Mod le OCR charg .")

print("Chargement du mod le Gemini...")
model = genai.GenerativeModel('gemini-2.5-flash-lite')
print("    Mod le Gemini pr t.")

app = FastAPI()

# ---------------- LOGIQUE METIER ----------------
def clean_json_text(text):
    text = text.replace("```json", "").replace("```", "").strip()
    return text

def process_image(img_cv):
    # 1. OCR
    result_ocr = reader.readtext(img_cv, detail=0)
    raw_text = " ".join(result_ocr)

    # 2. Gemini Prompt
    prompt = f"""
    Tu es un assistant expert en extraction de donn es.
    Analyse ce texte brut OCR : "{raw_text}"

    D termine si c'est 'Facture' ou 'Vin'.

    Si FACTURE, extrais (JSON) :
    - type: "Facture"
    - date (JJ/MM/AAAA)
    - vendeur
    - montant_total
    - numero_facture

    Si VIN, extrais (JSON) :
    - type: "Vin"
    - nom
    - millesime
    - appellation
    - degre_alcool

    R ponds UNIQUEMENT en JSON valide.
    """

    try:
        response = model.generate_content(prompt)
        parsed_json = json.loads(clean_json_text(response.text))
```

```
        return parsed_json
    except Exception as e:
        print(f"Erreur Gemini/Parsing: {e}")
        return {"error": "Echec de l'analyse IA", "details": str(e)}

# ---------------- API ENDPOINTS ----------------
@app.get("/")
def home():
    return {"status": "API OCR Gemini Dockeris e en ligne !"}

@app.post("/analyze")
async def analyze_endpoint(file: UploadFile = File(...)):
    if not file.content_type.startswith('image/'):
        raise HTTPException(status_code=400, detail="Le fichier doit  tre  une
    image.")

    try:
        contents = await file.read()
        nparr = np.frombuffer(contents, np.uint8)
        img_cv = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

        result = process_image(img_cv)
        return result

    except Exception as e:
        return {"error": str(e)}
```

References

[1] JaidedAI. (2020). *EasyOCR: A comprehensive Optical Character Recognition library in Python.* GitHub Repository.

[2] Google. (2023). *Gemini: A Family of Highly Capable Multimodal Models.* Google DeepMind Technical Report.

[3] Baek, Y., Lee, B., Han, D., et al. (2019). *Character Region Awareness for Text Detection (CRAFT).* Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).

[4] Xu, Y., Li, M., Cui, L., et al. (2020). *LayoutLM: Pre-training of Text and Layout for Document Image Understanding.* Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.

[5] Vercel. (2023). *Next.js Documentation: App Router and Server Components.*