N26 Task Solution

Introduction

I decided to use vision LLM from HuggingFace to read provided PDF documents. I decided to write a lot of custom code using HuggingFace library to demonstrate my expert level understanding of how LLMs or VLLMs work.

Overall, the task is quit simple and if I had enough time and confirmation that all the documents have fixed standard format, I believe I could archived perfect results on information extraction. Below I will share my ideas about what I would do in the future.

Method

I decided to convert all PDFs into images and process them that way. I decided so since some PDFs had only images as internal information, which meant text could not be extracted via PDF parsing libraries in Python, and other files had only text with tables. Thus, I decided to convert all of them into an image format and work with them that way.

I found some files has 1 page, others 2 pages, but none of the files had more than 2 pages. Thus, to provide all the document pages to AI for information extraction at once, I decided to concatenate images of pages if document had 2 page long information.

To get right information back, I used `field_descriptions.csv` and `extraction-fields.json` files to provide as much information to VLLM via a short prompt as possible. For this reason, I even slightly have changed field descriptions in `field_descriptions.csv` to make information more clear for VLLM.

To provide AI with proper field names to find information for, I first ask general information to be extracted, and then I selected specific fields depending on document type. So in total I do two forward passes for a single document. I realised that I could have used K,V caching to speed up the process, since I need only one pre-fill operation for image tokens. In other words, in my second forward pass, I change only text tokens and keep the same image tokens, which opens opportunity for such caching

optimization. But I did not have time to implement it. On the meeting, I can explain how easily I could do that in HuggingFace library.

The `document_extractor.py` file contains document processing code and `model_inference.py` contains code to make a forward pass of a model.

Model

I selected Gemma3 model, since it is quit small model (just 4 billion parameters), it is the most recent, was trained on 140+ languages and showed good results on OCR related benchmarks. But if I had more time, I would test other models as well.

On average, model takes 175 seconds to process a single document on my NVIDIA GeForce RTX™ 4060 Laptop GPU, and 6.5 Gigabytes of GPU RAM in 4bit quantized version.

Metrics

Unfortunately, I did not count any of the matrics since it requires having labeled data, which I did not have. If I had such, I would count per each field exact matches on entire string level, and on character level as well. This would give me information of which fields are hardest to extract.

During the code development, I was manually checking model extracted information and comparing it to the text in the file. During this process, I tested the model on different OCR capabilities, and I was surprised how well it performed.

Results

You can find extracted JSON results in `experiment.ipynb` file for all the documents.

Future Improvements

For the future, I would check if the document have fixed standardized format. If so, I could estimate bounding boxes of a specific fields for different document types. I think this can improve results since I found that classifying document types and estimating document languages, was easier task for the model. It never made a mistake with languages.

Thus, we could reliably predict document type and split the document into different parts via bounding boxes of the fields. Then, instead of asking model to fill all the information at once, we would perform

this operation step by step, which can lead to significant improvements. Also, since this sounds very much as agentic workflow, we could incorporate LangGraph library to write such code.

For Production

I would never put this code into production. First of all, I don't like the code structure. For production, I would change:

- Configuration of the code: I would add configuration yaml file to change prompt and models easily;
- Usage of K,V caching to not to pass the same token to the model twice.
- GPU specific speed-ups. This could be done via tools like TensorRT, which also supports K,V caching.
- I would use Agentic libraries to create more complex workflows, like calling a model multiple times to double-check extraction results from previous steps for correctness & grammar.

📦 Requirements

Python 3.12+

Dependencies defined in `requirements.txt`.

📂 Project Structure

- experiment.ipynb - Contains extracted JSONs per each document
- document_extractor.py - The main file. Contains code for document reading and information extraction.
- model_inference.py - Contains code to simplify model initialization, tokenization and forward pass
- extraction-fields.json - Name of the field to be extracted per different type of document
- field_descriptions.csv - Descriptions of each field
- requirements.txt - Code dependencies

▶️ Commands to Run

Install the application locally:

```
pip install -r requirements.txt
```

In your Python code:

```
from document_extractor import DocumentExtractor

document_reader = DocumentExtractor(model_id='google/gemma-3-4b-it')

result_dict = document_reader.extract(document_path="documents/doc-01.pdf")

print(result_dict)
```

Output:


```
{

    "account_number": "DE89 3704 0044 0532 0130 00",
    "account_type": "Checking Account",
    "statement_period": "31. März 2025",
    "opening_balance": "2.457,63 EUR",
    "closing_balance": "0 EUR",
    "available_balance": "0 EUR",
    "transaction_number": "N/A",
    "document_id": "KTO-DE-78901-2025",
    "document_type": "personal account",
    "document_date": "05.03.2025",
    "customer_name": "Hannah Schmidt",
    "customer_id": "K-78945612",
    "institution_name": "Deutsche Sparkasse",
    "institution_address": "Hauptstraße Berliner Allee 33, 40212 Düsseldorf,
Deutschland",
    "language": "German"
}
```