ENGLISH TO GERMAN PDF TRANSLATION: POC REPORT

Introduction

The primary goal of this POC was to develop an English to German document translation system, The primary goal was to overcome the limitations of our previous approach, where we relied on the DeepL API. Key challenges with the former system included high operational costs, inaccurate translation of mathematical and technical notation and failure to preserve document formatting particularly tables and structures. This new methodology addresses these issues by integrating a layout-aware parsing technique with a locally hosted Large Language Model (LLM), delivering a cost-effective (as we use open-source models), accurate, and structurally sound translation solution.

Methodology

Step 1: Structural Parsing

The main idea of this methodology is the use of the **unstructured** library to parse the input PDF. Instead of performing a raw text extraction which I tried earlier using PyPDF2, the partition_pdf function of the **unstructured** library analyses the document's layout and identifies distinct structural elements like Header, Title, NarrativeText, Footer, Table etc. While partitioning the pdf we use the strategy ”hi_res” where we use popplers and **tesseract**, which helps us deal with mathematical expressions and language based partitioning. This layout-aware approach ensures on preserving the document's original format.

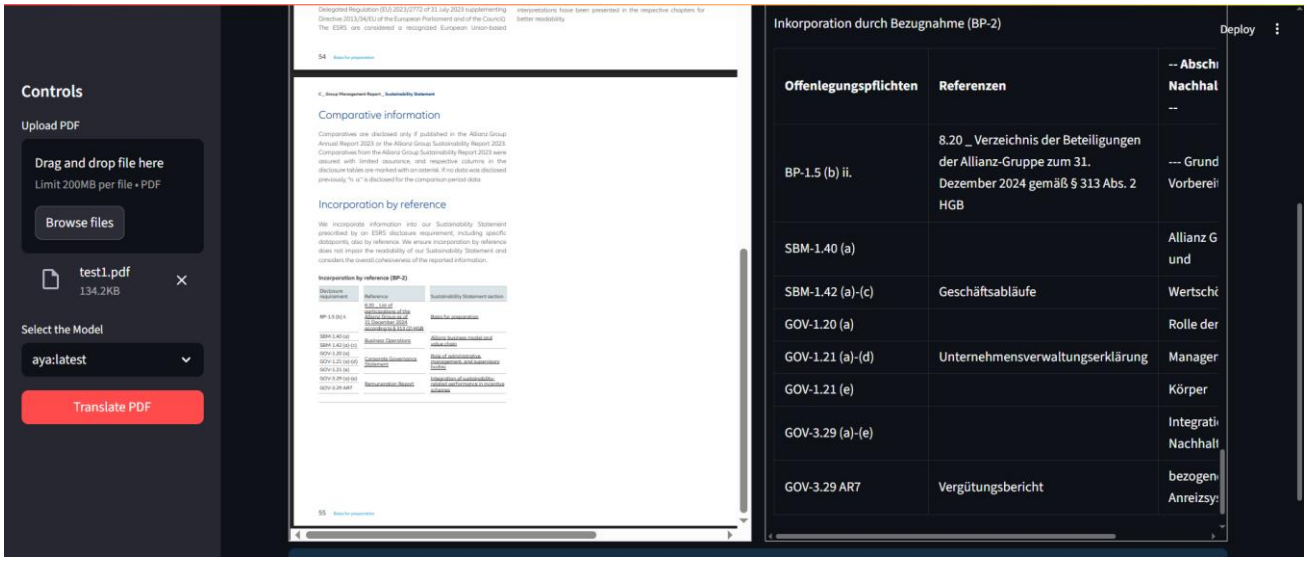
Step 2: Contextual Chunk Translation

Translation is performed not on a word-by-word basis, but on complete structural chunks (e.g., an entire paragraph or a single table cell). A locally hosted LLM (such as aya23 or a specialized model) is accessed via the Ollama framework. Each text chunk is sent to the model with an engineered prompt instructing it to act as an expert translator for technical and financial documents.

```
Step 2/3: Translating elements...
  -> Translating element 1/55 (Type: Header)...
  -> Translating element 2/55 (Type: Title)...
  -> Translating element 3/55 (Type: UncategorizedText)...
  -> Translating element 4/55 (Type: Title)...
  -> Translating element 5/55 (Type: NarrativeText)...
  -> Translating element 6/55 (Type: NarrativeText)...
  -> Translating element 7/55 (Type: Title)...
  -> Translating element 8/55 (Type: NarrativeText)...
  -> Translating element 9/55 (Type: NarrativeText)...
  -> Translating element 10/55 (Type: NarrativeText)...
  -> Translating element 11/55 (Type: Title)...
  -> Translating element 12/55 (Type: Title)...
  -> Translating element 13/55 (Type: NarrativeText)...
  -> Translating element 14/55 (Type: NarrativeText)...
  -> Translating element 15/55 (Type: NarrativeText)...
  -> Translating element 16/55 (Type: NarrativeText)...
  -> Translating element 17/55 (Type: Footer)...
  -> Translating element 18/55 (Type: Title)...
  -> Translating element 19/55 (Type: NarrativeText)...
  -> Translating element 20/55 (Type: Title)...
  -> Translating element 21/55 (Type: NarrativeText)...
  -> Translating element 22/55 (Type: NarrativeText)...
  -> Translating element 23/55 (Type: Title)...
  -> Translating element 24/55 (Type: NarrativeText)...
  -> Translating element 25/55 (Type: Footer)...
  -> Translating element 26/55 (Type: Header)...
  -> Translating element 27/55 (Type: Title)...
  -> Translating element 28/55 (Type: NarrativeText)...
  -> Translating element 29/55 (Type: Title)...
  -> Translating element 30/55 (Type: Table)...
```

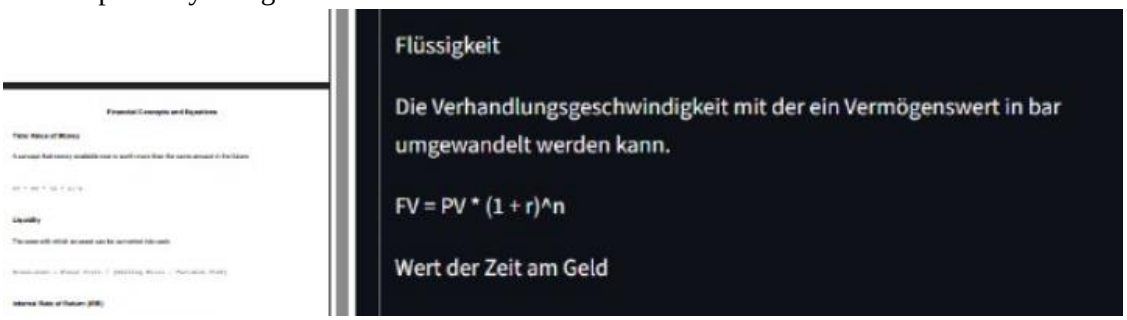
Step 3: Document Reconstruction

After each element is translated, it is reassembled into a final HTML document. Text elements are wrapped in paragraph tags, and translated table data is used to reconstruct the original table structure. This ensures the final output mirrors the layout of the source document as closely as possible.



Addressing Issues

1. *Operational costs* – This system eliminates API costs. By using a locally hosted open-source LLM via Ollama or by using a colab cloud environment, the expenditure model shifts from a variable pay-per-character service to a fixed-cost infrastructure. This provides a minimal budget and ensures significant long-term savings, making the translation process economically sustainable.
2. *Inaccuracy with Mathematical Expressions* – The idea of using models better with translating accurately on mathematical expression and working on an engineered prompt exceptionally increased the accuracy on mathematical expressions. When I tried out a different sample 3 paged document which contained a lot of mathematical expressions, it worked perfectly along with German translated text.



3. *Formatting and Table Preservation* – The real idea for keeping the document's layout lies in how we now handle tables, which is all thanks to the unstructured library. Instead of just seeing a wall of text, this system recognizes a table and treat it specially. It meticulously works through the table one cell at a time, translating the contents of each cell in isolation. Once translated, every piece of text is carefully placed back into its original position within a newly reconstructed table. This makes sure that the final German document has perfectly aligned columns, rows, and headers, maintaining the table's structure in a way our previous, more basic text-extraction method simply couldn't manage.

```
def translate_table(html_table, model_name):
    soup = BeautifulSoup(html_table, 'html.parser')
    cells = soup.find_all(['td', 'th'])
    for cell in cells:
        if cell.string:
            translated_text = translate_chunk(cell.string, model_name)
            cell.string.replace_with(translated_text)
    return str(soup)
```

The same way, we could also address issues if with other structures like graphs etc.

Performance

This methodology was evaluated using a 3-page PDF of a sustainability statement, which included paragraphs and data tables.

Accuracy

Tests using the aya-expanse:8b and aya23 models yielded an estimated accuracy of 96-99%. This was determined by feeding translated samples to an LLM. This is an informal benchmark and we know that model can hallucinate, achieving such a high score is an indication of the system's quality and its ability.

Latency

Local Environment: When running on a standard development machine using Ollama, the 3-page document took approximately 30 minutes to process. To mitigate this, I was trying optimizations using asynchronous processing to improve latency on local hardware without losing the high accuracy achieved.

Cloud Environment (Google Colab): The same task was completed in just 3 minutes, a 10x speed improvement due to superior hardware. However, this came at the cost of a noticeable drop in accuracy, which fell to the 85-90% range.

Model Selection and Evaluation

The models tested included **llama3.2:latest, aya:latest, aya-expanse:8b, qwen2.5:latest, mistral-nemo:12b, mixtral:8x7b, tinyllama:latest, deepseek-r1:1.5b, and Helsinki-NLP/opus-mt-en-de,** while also trying with gpt-4o for state-of-the-art performance. From this testing, both gpt-4o and the Aya models consistently delivered translation quality. However, the Aya models proved to be the ideal choice, as their specialization in multilingual tasks delivers high accuracy within our cost-effective, locally-hosted system.