AI-Powered Document Insights and Data Extraction

Name: Anubhav Sharma
Email: anubhav.sharma@colorado.edu

Problem Statement | Document Intelligence

Automatically extracts structured answers from messy, multi-document PDFs from Mortgage Application so analysts can instantly query key fields without manual review and perform basic Q&A..

Challenge

Organizations across all sectors receive countless documents every day and do manual processing on them. In Mortgage Industry one example is processing of Customer Application, for which employees manually process the documents which is error prone and time consuming, and this impacts overall processing of the application.

Solution

Built a robust RAG Pipeline powered by Logical Document Segmentation, Query Routing, and top of the line Retrievers, all running on local LLM which then is integrated with a minimalistic User-friendly Gradio App allowing employees to upload documents and extract key fields and perform basic Q&A.

Pain Point	How Your Pipeline Solves It
Document Tagging	Used LLM for document tagging
Scanned Documents	Used Tesseract OCR to handle them
Blob Documents	Made a robust process for logical document segmentation
Input Errors	Used advanced Retrievers with Reranking to extract key data

System Architecture | Augmented RAG Pipeline

Document Input → OCR Processing → Logical Documents Creation → Recursive+Semantic Text Chunking → Embedding Generation → FAISS Vector Storage → Query Routing → Retrieval → LLM Response → Final Response Formatting

Component Specifications

Component	Technology Choice	Configuration Details
OCR Engine	Tesseract	DPI: 300, oce_psm: 6, ImageOps AutoContrast
Text Chunking	Recursive and Semantic. Used Recursive on Logical Documents, then used Semantic chunking on new chunks.	Recursive: Chunk size- 1600 , Overlap- 100 Semantic: Buffer Size- 15, Breakpoint Threshold- 90
Embeddings	BAAI/bge-base-en-v1.5	Dimensions: 768
Vector Database	FAISS based Vector Store	Similarity metric: L2 Indexing method: FAISS IndexFlatL2
Retriever	Metadata Filtering + Hybrid + Query Expansion	Top-K: 8, Num_Queries: 3, Reranking: MiniLM-L-2-v2
LLM	Local Llama 3.1 8B Instruct Q8 Model running on 32 GB RAM and 8GB VRAM NVIDIA Laptop GPU	Size: 7.95 GB, 8 Billion Parameters, Temperature: 0.1/ 0.2 for logical documents and query routing, Context Window: 4096, Max Tokens: 64, Generation Filters
Prompt Strategy	Context-injected, Zero-Shot Prompting	Strict context retrieval, using “Not Found” if no answer found

Pipeline Performance Metrics | Results from 1-week Testing

We made our RAG pipeline on training files which included blob PDF files, scanned PDF files, and normal PDF files. Then we tested the RAG Pipeline and Gradio Chat on test data containing files we did not refer while making the RAG pipeline.

We made 15 Test Queries and evaluated the performance.

🧪 Retrieval Performance

- **Recall@5:** 73% (target: 80%)
- **Recall@8:** 86.67% (target: 85%)
- **Mean Reciprocal Rank (MRR):** 0.37
- **Hit Rate@8:** 86.67%

💊 End-to-End Accuracy

- **Answer Accuracy:** 66.67% (evaluated on 15 test queries, 90%+ accuracy on Gemini 2.5 Flash)
- **Numeric Accuracy:** 83.33%

🧑‍🎓 System Performance

- **Average Response Time:** 42.492 seconds (In gemini it was less than 10 seconds)
- **Retrieval Latency:** 3.8 seconds

Pipeline in Action | Gradio App

Example Query 1: Factual Lookup

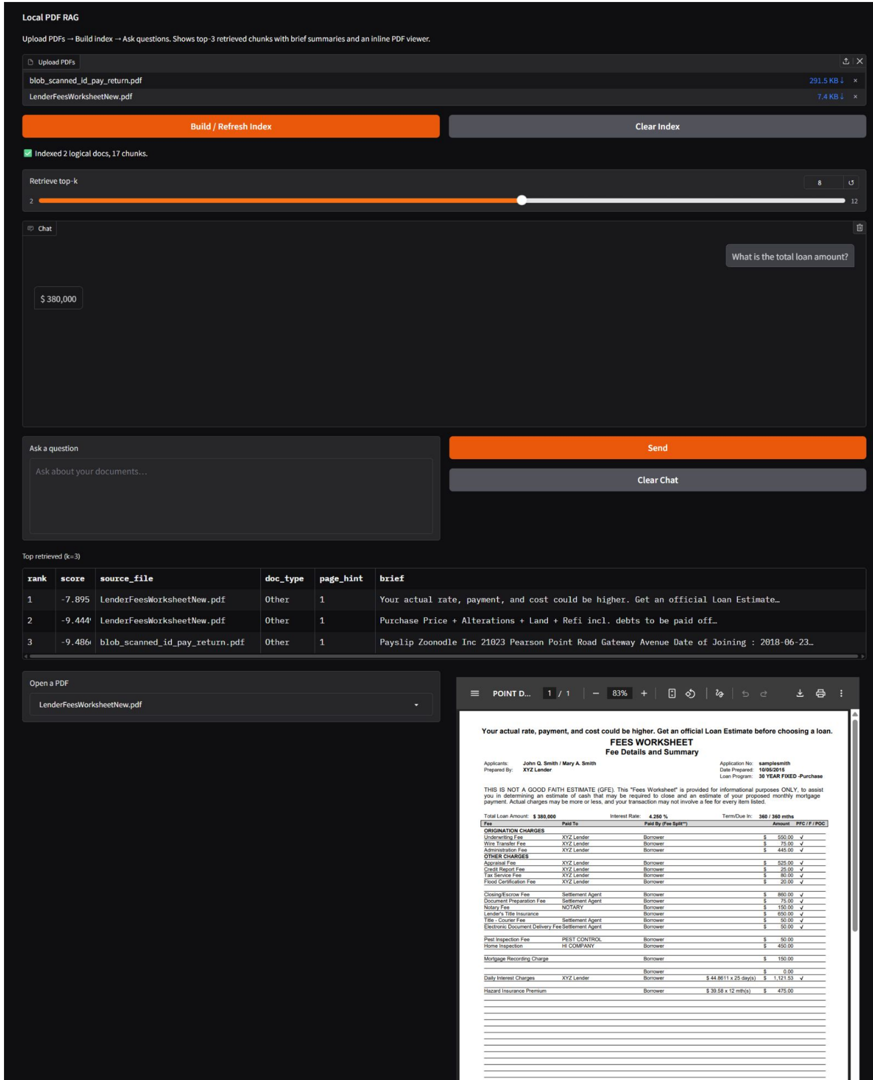
Query: "What is the total loan amount?"

Top 3 Chunks:

1. LenderFeeWorksheetNew.pdf page 1, Score -7.895
2. LenderFeeWorksheetNew.pdf page 1, Score -9.444
3. Blob_scanned_id_pay_return.pdf page 1, Score -9.486

Note: *These are reranker scores and are not logits, regardless, higher scores are better*

Final Answer: $380,000



Pipeline in Action | Notebook

Example Query 2: Summarization

Query: Summarize Neighborhood Description in the appraisal report.

Final Answer: A residential neighborhood comprised predominantly of 2-3 story, wood frame, row style, and detached SFRs

Retrieved Context: 3 Chunks result by Reranker, second chunk has the answer. All three chunks related to “neighborhood”.

Response Quality: Answer is direct and to the point. Due to limitation in LLM and its tendency of rambling and repetition we limited the response and formatted it, leading to direct short responses but this would impact ability of answering more complex queries.

```
start_time = time.time()

query = "Sumamrize Neighborhood Description in the appraisal report."
rag_engine = build_rag_pipeline(index, llm_rag)
response = rag_engine.query(query)

elapsed_time = time.time() - start_time

raw = str(response)
final = finalize_answer_minimal(raw)

print('\nFinal Response:\n ---------------------- \n')
print(final)  # print the cleaned, single-line answer
print(f"\nQuery execution time: {elapsed_time:.3f} seconds")
```

```
2025-09-26 18:30:00,779 - DEBUG - Building index from IDs objects
Batches: 100%  1/1 [00:00<00:00, 69.30it/s]

Final Response:
 ----------------------

A residential neighborhood comprised predominantly of 2-3 story, wood frame, row style, and detached SFRs

Query execution time: 49.625 seconds
```

Design Decision Analysis | High Recall Centric RAG and Minimalistic App UI

Model Selection Reasoning

Decision	Rationale	Trade-offs Considered
BAAI/bge-base-en-v1.5	Better performance than small version and can support multi-language	Higher compute vs. lighter models
Recursive+Semantic	Recursive Splitting Logical Documents and then Semantic Chunking	Higher compute for slight improvement in chunks
Llama 3.1 8B Instruct	Best performing Local Model, tested 7 different Local LLMs	Higher compute vs. lighter models like Mistral 7B Instruct Q5. Local running Model compared to APIs.
FAISS Vector Store	Better than default RAM based Vector Store and highly scalable	Increased Complexity and GPU Load.

We also tested whole pipeline on Gemini 2.5 Flash and it worked much better than Llama 3.1 8B Instruct and faster too. Since we are limited to Local LLM only for this project we can't drastically improve pipeline as both latency and accuracy are limited to Llama model, which performed better compared to 7 other local models but still too limited and slower compared to commercial API based LLMs.

Key Trade-offs Made

🧪 **Speed vs. Accuracy:**

- Chose larger Q8 Llama 3.1 8B over smaller local LLMs we could have run locally.
- Speed and Accuracy could be improved with better LLMs or API based ones

📍 **Complexity vs. Maintainability:**

- Minimalistic Gradio App UI, limited control over RAG for users.
- Used FAISS Vector to make more scalable pipeline but increased complexity.
- Streamlined RAG for Gradio App, improving latency by 40% for response, and reduced index preparing time from 5+ minutes to 15 seconds.

Current Limitations & Next Steps

Current Limitations

1. **Input Processing Issues**
   - Require improvements in tabular data extraction
   - Current OCR is not able to process blurry images with tables well, we could use more advanced methods
2. **LLM Performance and latency Issues**
   - We use LLMs to create logical documents and RAG responses, current local Llama LLM performs much worse in both compared to Gemini API.
3. **Scalability Concerns**
   - We used FAISS and other techniques to make it more scalable, but in 1000 page documents it may take an hour to make index alone if we use our limited model on limited laptop based hardware

Proposed Enhancements

Short-term- 2 Weeks Period:

- Improve OCR workflow and overall text ingestion workflow, especially Tabular data extraction.
- Switch to Gemini, Claude, or OpenAI API based LLMs for instant improvement across all metrics.

Medium-term- 4 Weeks Period:

- Research on best UI features for end users.
- Optimize Retrieval Stack: Change Indexing to an ANN Index. Improve Routing and response pipeline.
- Prepare an advanced version of APP for power users who want more control over RAG.

Long-term Vision:

- Integrate both versions of App to create one robust minimalistic APP with advanced features as toggles
- Migrate whole pipeline to Cloud, allowing for features like autoscaling, sharding, audit trails, and more.
- Expand training set for Multilingual support and add support for other areas besides Mortgage.
- Integrate with Cloud Storages and Slack/Team Bots

Project Impact & Learning Outcomes

🧪 Key Technical Learnings

- **LLM Fused RAG:** Using good quality LLMs for logical documents generation, chunking, and routing improve the performance drastically.
- **Good Ingestion is the Core:** By improving text extraction for tables we improved our Recall@8 from 70% to 86.67% for test documents where each document was unique and for different persons.
- **Local LLMs Issues:** Project limited us to local machine running LLMs and all instruct models were bad with prompt engineering and all of them tends to ramble and repeat. Had to switch to Zero Shot Prompting and cheap heuristics to stop their rambling.

💊 Business Impact Potential

- **Efficiency Gains:** Replaces manual cross-document searching with instant Q&A accelerating Mortgage application.
- **Accuracy Improvement:** Grounded answers to retrieved sources alongside rule based final formatting, mitigating off-topic generations and rambling. With Recall of 0.87 on the test set, most questions had the right evidence in context, creating a knowledgeable RAG.
- **Scalability:** Pipeline is modular and highly scalable and can be integrated with secure Cloud services and downstream/upstream apps. It can be scaled locally too if by streamlining Text Parsing and LLM calls.

🧑‍🎓 Skills Developed

- LLMs and RAG Engineering with LlamaIndex and using LlamaCPP library for local models
- OCR + PDF Parsing with with PyMuPDF, tesseract, ImageOPs, OpenCV, and other Python libraries
- Iterative evaluation and trade-off management. Working under hardware and data constraints.

Thank you