LocalDocu - AI-Powered Document Assistant

Enrollment No(s) - 23103262, 23103278, 23103303

Student Name(s) – Pranav Sharma, Kush Kansal, Prakhar Singhal

Name of Supervisor(s) – Mr Prashant Kaushik

November 2025

Submitted in partial fulfilment of the Degree of

Bachelor of Technology

in

Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING &

INFORMATION TECHNOLOGY

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

Students’ Self Declaration for Open Source libraries and other source code usage in Minor Project

I / We, **Pranav Sharma (23103262), Kush Kansal (23103278), and Prakhar Singhal (23103303)**, hereby declare the following usage of the open-source code and pre-built libraries in our minor project in our minor project 5th Semester, with the consent of our supervisor. We also measure the similarity percentage of pre-written source code and our source code, and the same is mentioned below. This measurement is true to the best of our knowledge and abilities.

1. List of Pre-Built Libraries

The following open-source libraries were used in the development of our project. Each library is licensed under permissive open-source licenses.

1. LangChain
2. ChromaDB
3. Ollama
4. Next.js
5. shadcn/ui
6. PyMuPDF
7. HuggingFace Embeddings
8. FlashRank
9. LLaVA
10. PyInstaller

2. List of Pre-Built Features in Libraries or Source Code

We have used the following pre-built features from the above libraries with minimal modifications:

a. Retrieval:
   - Hierarchical RAG
   - LangChain's ContextualCompressionRetriever
   - FlashRankRerank.

b. SemanticChunker.

3. Percentage of pre-written source code and source written by us

Percentage of pre-written source code: 30%

Percentage of source code written by us: 70%

Student ID	Student Name	Student signature
23103262	Pranav Sharma	
23103278	Kush Kansal	
23103303	Prakhar Singhal	

Declaration by Supervisor (To be filled by Supervisor only)

I, ... (Name of Supervisor) declares that I above-submitted project with the title .... was conducted under my supervision. The project is original and neither the project was it copied from External sources nor was it submitted earlier in JIIT. I authenticate this project.

(Any Remarks by Supervisor)

Signature (Supervisor)

CERTIFICATE

This is to certify that the work titled **LocalDocu - AI-Powered Document Assistant** submitted by **Pranav Sharma (23103262)**, **Kush Kansal (23103278)** and **Prakahar Singhal (23103303)** in partial fulfilment for the award of the degree of **B.Tech – Computer Science and Engineering** of Jaypee Institute of Information Technology, Noida has been carried out under my supervision. The work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor	……………………….
Name of Supervisor	……………………….
Designation	……………………….
Date	……………………….

ACKNOWLEDGEMENT

While bringing out this report to its final form, we came across several people whose contributions in various ways helped our field of research, and they deserve special thanks. It is a pleasure to convey our gratitude to all of them. First and foremost, we would like to express our deep sense of gratitude and indebtedness to our supervisor, **Mr Prashant Kaushik** and student mentor **Ms Niharika Mishra** for their invaluable encouragement, suggestions and support from an early stage of the research and for providing me with extraordinary experiences throughout the work.

Above all, their priceless and meticulous supervision at each phase of work inspired us in innumerable ways. We especially acknowledge them for their advice, supervision, and the vital contribution as and when required during this research. Their involvement with originality has triggered and nourished our intellectual maturity that will help us for a long time to come. We are proud to record that we had the opportunity to work with an exceptionally experienced Professor like him.

Signature Of Student 1:	....
Name Of Student 1:	Pranav Sharma
Enrollment Number:	23103262
Date:	20th Nov 2025
Signature Of Student 2:	....
Name Of Student 2:	Kush Kansal
Enrollment Number:	23103278
Date:	20th Nov 2025
Signature Of Student 3:	....
Name Of Student 3:	Prakhar Singhal
Enrollment Number:	23103303
Date:	20th Nov 2025

SUMMARY

1. Motivation behind the project:

The primary motivation is to address privacy, latency, and cost challenges associated with cloud-based AI services. LocalDocu aims to provide a secure, locally executable AI agent that prevents sensitive data exposure by localising inference and storage, eliminates per-query costs, and enables offline operation. Furthermore, it seeks to investigate the reliability of hierarchical RAG and multimodal retrieval when constrained to local consumer hardware.

2. Type of project: Development cum research project

The project combines engineering deliverables—frontend, backend, deployment packaging—with research aspects such as hierarchical Retrieval-Augmented Generation (RAG), reranking strategies, multimodal LLaVA integration, and structured citation enforcement.)

3. Critical Analysis (Gaps Addressed)

1. Gaps in Existing Systems

Most document-assistant tools depend on cloud-based AI, causing privacy concerns, latency, and operational costs.

They use flat retrieval models, which are inefficient for large documents, and lack multi-modal support, making them unable to interpret images or visuals embedded in PDFs.

2. Gaps Addressed by LocalDocu

1. Privacy-Preserving Local AI: Runs all processing locally, ensuring data never leaves the user's device.
2. Hierarchical Retrieval Architecture: Combines summary-level and detailed-level vector stores with semantic chunking and reranking for precise, context-aware retrieval.
3. Multi-Modal Integration: Uses a vision-language model (LLaVA) to analyse images and embed visual information alongside text for unified query handling.
4. LocalDocu ensures data privacy, improves retrieval accuracy, and enables text–image reasoning in a fully offline setup.

   It transforms document intelligence by moving from cloud-dependent to local AI, from flat to hierarchical retrieval, and from text-only to multi-modal understanding, establishing itself as a scalable, privacy-first AI framework for intelligent document analysis.

New Technologies & Tools Learned:

- Languages: Python 3.10+ (Backend), TypeScript, JavaScript (Node.js for Progress Service).
- Frameworks: FastAPI (Backend Orchestration), Next.js (Frontend UI), LangChain
- AI/ML Tools: Ollama (Local Inference Server), ChromaDB (Local Persistent Vector Store), FlashRank (Reranking), LLaVA (Vision Model), HuggingFace Embeddings.

4. Overall design of project (Workflow & Data Model):

1. Ingestion: User uploads a PDF; the backend splits text into semantic chunks using LangChain's SemanticChunker and extracts images via PyMuPDF.
2. Image Processing: Extracted images are passed to the LLaVA model to generate textual summaries.
3. Storage: Summaries are stored in summary_store (coarse index), and detailed text/image chunks are stored in detailed_store (fine-grained index).
4. Retrieval: The system retrieves top matching document summaries first, then fetches and reranks detailed chunks from the detailed_store using FlashRank.
5. Generation: The local LLM generates a structured JSON response with citations based on the retrieved context.

5. Features built, the Programming language used:

Key Features:

- Privacy-First Local Inference: The entire pipeline runs locally with no cloud data transmission.
- Hierarchical RAG: Two-level vector store (summary + detailed chunks) for efficient, context-rich retrieval.
- Multimodal Processing: Integrates LLaVA to extract and interpret images inside PDFs.
- Real-time Progress Tracking: Posts ingestion status to a separate microservice to prevent HTTP timeouts and provide visual feedback.

6. Proposed Methodology (Pipeline Details):

The project follows a specific pipeline methodology:

1. Ingestion: PDFs are loaded via PyMuPDFLoader; images are extracted and summarised using generate_image_summary with LLaVA.
2. Processing: Text is split using SemanticChunker to balance semantic coherence. A small model (Mistral) generates summaries for these chunks.
3. Indexing: Embeddings are generated using HuggingFaceEmbeddings. Document summaries

go to summary_store; chunks go to detailed_store.

4. Retrieval: query_rag implements hierarchical retrieval: it searches summaries first, extracts relevant doc_ids, and then retrieves specific chunks.
5. Reranking: Retrieved chunks are reranked using FlashRankRerank to select the top-K most relevant segments.
6. Generation: The local LLM is invoked with a JSON-enforced prompt to return an answer with citations, which are then deduplicated.

7. Algorithm/Description of the Work:

Core Algorithms:

1. Semantic Chunking: Splits text based on embedding similarity rather than fixed character counts to improve downstream LLM context quality.
2. Hierarchical Retrieval:
    - Step 1: summary_retriever.search(prompt) filters relevant documents.
    - Step 2: detailed_retriever fetches chunks only for those filtered documents.

8. Division of the work among students:

1. Kush Kansal: Architecture definition, backend orchestration, API definitions, overall system integration, and tests.
2. Prakhar Singhal: Frontend implementation (Next.js), design and UX, chat UI components, and progress UI.
3. Pranav Sharma: Ingestion pipeline, chunking & summarisation logic, image extraction, and LLaVA integration.

9. Results & Performance Evaluation:

- Quantitative Observations:
    - Ingestion Time: 10-50 page documents processed in 15-30 seconds; 100-page documents take 4-8 minutes.
    - Query Latency: Median generation latency is 4-6 seconds for typical document setups on an i5 CPU with 16GB RAM.
- Qualitative Observations:
    - Reranking: FlashRank improved contextual relevance in ~70% of ad-hoc test queries.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

1.1 <u>Introduction</u>

In the rapidly evolving landscape of information technology, the efficient management and analysis of extensive document repositories have become critical necessities for both academic researchers and enterprise professionals. While the proliferation of cloud-based Artificial Intelligence services has revolutionised how users interact with unstructured text, these centralised solutions frequently introduce significant operational bottlenecks related to data privacy, network latency, and escalating costs. LocalDocu emerges as a pivotal architectural solution to these challenges, serving as a privacy-first document assistant designed to operate entirely within the secure confines of a user's local hardware. Developed as a comprehensive development-cum-research project, LocalDocu bridges the gap between sophisticated Large Language Model (LLM) capabilities and strict data sovereignty requirements, offering a robust platform for secure document ingestion, summarisation, and complex interrogation without a single byte of data traversing the public internet.

At the technical core of LocalDocu lies a sophisticated implementation of Retrieval-Augmented Generation (RAG), engineered specifically to overcome the limitations of running heavy inference tasks on consumer-grade devices. Unlike traditional flat retrieval systems that often struggle with context retention over long documents, LocalDocu introduces a Hierarchical RAG architecture that fundamentally optimises how information is indexed and retrieved. This novel approach utilises a dual-layer vector storage system, maintaining a coarse summary level for broad context retrieval and a detailed chunk level for fine-grained evidence extraction. By leveraging the power of Ollama for local inference, the system manages the complex orchestration of various open-source models—such as Mistral for textual reasoning and LLaVA for visual processing—ensuring that the semantic understanding of documents is both deep and computationally efficient.

The driving motivation behind the development of LocalDocu is explicitly three-fold: guaranteeing privacy, ensuring cost accessibility, and advancing applied research in local AI. In an era where data breaches are increasingly common, the ability to prevent sensitive data exposure by strictly localising inference and storage is invaluable for users handling confidential legal, medical, or proprietary corporate documents. Furthermore, the architecture addresses the economic constraints of modern research by systematically eliminating the per-query token costs associated with commercial APIs, thereby democratizing access to high-end document analysis tools. This offline operational capability ensures that the system remains fully functional and responsive regardless of internet connectivity, making it an ideal tool for secure, air-gapped environments where reliability is paramount.

Beyond its core retrieval capabilities, LocalDocu distinguishes itself through advanced, research-grade features designed to enhance the trustworthiness and depth of automated analysis. Recognising that modern documents are rarely text-only, the system incorporates a multimodal processing pipeline that uses the LLaVA vision-language model to extract, summarise, and query images embedded within PDFs, ensuring that vital visual data is integrated into the analytical context. Crucially, to combat the prevalence of hallucinations in AI-generated content, the system enforces a strict citation-aware output format, generating structured IEEE-style references that link every generated assertion back to specific snippets and page numbers within the source text. This commitment to transparency and auditability positions LocalDocu not merely as a productivity tool, but as a rigorous research companion capable of supporting complex academic and professional inquiry.

1.2 Problem Statement

Modern document-centric workflows across academic and enterprise sectors currently rely heavily on cloud-based AI services, a dependency that introduces critical challenges regarding data privacy, latency, and operational costs. The primary problem LocalDocu addresses is the inherent risk associated with transmitting sensitive or proprietary information to third-party servers for processing, which exposes users to potential data breaches and unauthorised data usage. Standard Retrieval-Augmented Generation (RAG) approaches often assume centralised retrieval systems and cloud-based LLM inference, which creates a barrier for users requiring strict data sovereignty or those operating in bandwidth-constrained environments. Furthermore, traditional document analysis tools frequently struggle with long-context retrieval, leading to information hallucinations or the inability to synthesise answers from extensive corpora effectively. Existing solutions also often fail to integrate visual data effectively, treating documents as pure text and ignoring valuable information contained in charts and diagrams, while also lacking the ability to provide precise, verifiable citations for their generated assertions. Consequently, there is a distinct lack of open-source, offline-capable systems that can perform complex, multi-modal document reasoning with high fidelity while guaranteeing that no data packet leaves the user's local infrastructure.

1.3 Significance of the Problem

The resolution of these challenges is of paramount importance for sectors where confidentiality and data integrity are non-negotiable, such as legal, medical, and high-level academic research. By localising the entire AI pipeline, LocalDocu ensures compliance with strict data protection standards like GDPR, effectively neutralising the risks of data leaks associated with external API calls. The significance of this project extends to accessibility and cost-efficiency; by eliminating the per-token or per-query costs typical of commercial cloud APIs, LocalDocu democratizes access to advanced AI document analysis, making it feasible for students and researchers with limited budgets to utilise powerful analytical tools. The project also holds significant research value by validating the feasibility of running complex Hierarchical RAG architectures on consumer-grade hardware, challenging the assumption that high-quality document retrieval requires massive cloud compute clusters. Furthermore, the emphasis on structured, IEEE-style citations addresses a critical gap in Generative AI utility—trustworthiness—by ensuring that every AI-generated answer is traceable back to a specific page and snippet within the source text, thereby upholding academic rigour and reducing the prevalence of unchecked hallucinations in automated summaries.

1.4 Empirical Study

To validate the efficacy of the LocalDocu system, a series of quantitative and qualitative experiments was conducted using a standard laptop configuration equipped with an i5 CPU and 16GB of RAM, utilising CPU-based inference for Ollama. The study utilised a dataset comprised of academic papers and technical reports ranging from 10 to 100 pages in length to test ingestion speeds, retrieval accuracy, and system latency. Quantitative observations revealed that the ingestion pipeline, including semantic chunking and summary generation, required approximately 10 to 30 seconds for documents under 50 pages, while larger documents of up to 100 pages took between 2 to 5 minutes, depending on image density.

In terms of retrieval performance, the system demonstrated a median latency of 3 to 6 seconds for generating answers using the hierarchical retrieval method, a duration that includes fetching summaries, reranking detailed chunks, and LLM generation. A key metric evaluated was Precision@K, specifically measuring the relevance of the top 5 retrieved chunks, which yielded a score of approximately 0.78 across a test set of 30 queries. The study further established that the integration of FlashRank for reranking improved the contextual relevance of retrieved chunks in approximately 70% of ad-hoc test queries. Qualitatively, the citation recall—the proportion of generated assertions supported by referenced snippets—was found to be high when the Large Language Model adhered to the enforced JSON output format, confirming the viability of the system's structured citation engine.

System Overview

LocalDocu is architected as a modular, privacy-first application composed of three primary interacting components: a Next.js frontend, a FastAPI backend, and a standalone progress tracking microservice. The user interface is built with Next.js and TypeScript, providing a responsive environment for file uploads, chat interactions, and real-time document visualisation, while communicating asynchronously with the backend services. The core intelligence resides in the Python-based backend, which orchestrates the Local LLM server (Ollama) and manages the persistent vector stores via ChromaDB. This backend implements a sophisticated ingestion pipeline that utilises PyMuPDF to extract text and images, applies semantic chunking to divide text into meaningful segments, and generates vector embeddings using HuggingFace models.

A distinguishing architectural choice is the Hierarchical RAG system, which maintains two distinct ChromaDB collections: a summary_store for high-level document abstracts and a detailed_store for granular text chunks. During retrieval, the system first queries the summary store to identify relevant documents before performing a focused, reranked search within the detailed store, significantly reducing noise and computational overhead. For multimodal capabilities, the system employs an image processing pipeline where extracted images are passed to the LLaVA model to generate descriptive textual summaries, which are then embedded alongside the document text to enable image-aware Question Answering. Finally, to ensure a responsive user experience during computationally intensive tasks, a dedicated Node.js progress service tracks ingestion events in real-time, decoupling long-running processes from blocking HTTP API responses

Data Collection

The data collection strategy employed for the validation of the LocalDocu system focused on assembling a representative corpus of complex, information-dense documents that mirror the real-world requirements of academic and technical researchers. Instead of relying on pre-curated, sanitised datasets often used in standard NLP benchmarks, the study utilised a diverse collection of unstructured PDF documents, primarily consisting of academic research papers and detailed technical reports ranging in length from 10 to 100 pages. This selection was intentional to test the system's capabilities in handling varying document structures, including multi-column layouts, embedded citations, and dense technical jargon. Furthermore, to evaluate the multimodal capabilities of the architecture, the dataset included documents rich in visual assets such as diagrams, charts, and scientific plots. This raw data served as the input for the ingestion pipeline, where the system utilised PyMuPDF to extract both textual content and embedded images, effectively creating a ground-truth repository for subsequent retrieval and summarisation experiments. For the specific purpose of evaluating retrieval accuracy, a test set of 30 distinct queries was formulated, targeting specific facts, cross-referenced claims,

Methodology

The methodology adopted for LocalDocu centres on a Hierarchical Retrieval-Augmented Generation (RAG) architecture designed to overcome the context window limitations and retrieval noise common in traditional flat-index systems. The process begins with a sophisticated ingestion pipeline where raw PDF documents are processed using a Semantic Chunking algorithm. Unlike naive character-splitting methods that often fragment sentences or ideas, this approach utilises the embeddings of the text to identify natural semantic breaks, ensuring that each generated chunk maintains independent, coherent meaning.

Concurrently, a multimodal processing track detects and extracts images from the document pages, passing them to a quantised LLaVA vision model running locally via Ollama. This model generates descriptive textual summaries of the visual content, which are then treated as metadata-rich text chunks, effectively unifying the document's visual and textual information into a single retrievable format.

Following ingestion, the system employs a dual-tier storage strategy to optimise retrieval efficiency. A high-level summary of the document is generated using a lightweight Large Language Model (LLM) and stored in a dedicated summary_store, while the detailed semantic chunks are embedded utilising HuggingFace models and persisted in a separate detailed_store. The retrieval methodology is strictly hierarchical; when a user issues a query, the system first scans the summary store to identify the most relevant high-level concepts or document sections. This initial coarse retrieval acts as a filter, significantly narrowing the search space before the system queries the detailed chunk store. To further refine the quality of the context provided to the LLM, a post-retrieval reranking step is implemented using FlashRank. This algorithm re-evaluates the initial set of retrieved candidates, reordering them based on their specific relevance to the user's query to ensure that the final context window is populated with the highest quality information available.

The final generation phase involves a strict prompt engineering protocol that forces the local LLM to output answers in a structured JSON format, mandating the inclusion of IEEE-style citations that are programmatically deduplicated and mapped back to specific source snippets.

Evaluation Metrics

To rigorously assess the performance of the LocalDocu system, a comprehensive set of quantitative metrics was defined, focusing on retrieval accuracy, answer faithfulness, and system responsiveness. The primary metric for assessing retrieval quality was Precision@K, specifically Precision@5, which measures the proportion of relevant information chunks found within the top five results returned by the vector store. This metric is crucial for RAG systems as it directly dictates the quality of the context provided to the generative model. Complementing this was the Citation Accuracy or Citation Recall metric, which evaluated the faithfulness of the generated answers by calculating the percentage of AI-generated assertions that were explicitly supported by the referenced document snippets. This metric served as a proxy for measuring "hallucination," with higher scores indicating a more trustworthy and grounded system. From an operational perspective, System Latency was measured as the total time elapsed between a user's request and the final render of the answer, decomposing the timeline into retrieval latency and generation latency. Finally, Throughput and Scalability metrics were observed to determine the system's stability under load, specifically monitoring how the ingestion time scaled relative to the increasing page count and complexity of the input documents.

Empirical Observations

The empirical evaluation of the LocalDocu system yielded significant insights into the performance trade-offs inherent in local, CPU-based AI architectures. Quantitative analysis demonstrated that the system achieves a highly efficient ingestion rate for standard academic materials, with documents between 10 and 50 pages being fully processed—including chunking, summarising, and indexing—in approximately 10 to 30 seconds. Larger files approaching 100 pages exhibited a linear increase in processing time, taking between 2 to 5 minutes, a variance largely attributed to the computational density of the semantic chunking process and the number of embedded images requiring vision-model inference. In terms of responsiveness, the median generation latency for typical queries settled between 3 to 6 seconds, confirming that a hierarchical retrieval approach can function within acceptable interactive timeframes on consumer hardware without GPU acceleration. On the qualitative front, the system achieved a Precision@5 score of approximately 0.78 across the test set, indicating a strong ability to surface relevant content even from dense technical corpora. A critical observation was the impact of the FlashRank reranking algorithm, which was found to improve the contextual relevance of the retrieved chunks in approximately 70% of ad-hoc test queries, validating the architectural decision to include a post-retrieval optimisation step. Furthermore, the enforcement of structured JSON output proved highly effective in mitigating hallucinations; the system consistently produced accurate, well-formatted citations when the LLM adhered to the schema. However, it was noted that citation recall suffered slightly in instances where smaller, quantised models struggled to maintain strict JSON compliance over long generation sequences, highlighting a direct correlation between model parameter size and structural adherence in complex RAG tasks.

Challenges and Limitations

Despite the successful implementation of a privacy-centric architecture, the LocalDocu system operates within several inherent constraints primarily dictated by the reliance on consumer-grade hardware for local inference. The most significant challenge lies in the trade-off between model performance and system latency; executing quantised Large Language Models (LLMs) like Mistral or Gemma entirely on a Central Processing Unit (CPU) introduces noticeable generation delays compared to cloud-based GPU clusters, particularly when processing extensive context windows or executing multimodal image analysis with LLaVA. Furthermore, the reliance on smaller, parameter-efficient models to ensure compatibility with standard RAM capacities occasionally results in "instruction drift," where the model struggles to strictly adhere to complex JSON output schemas required for the structured citation engine, leading to occasional parsing failures that necessitate fallback mechanisms. Additionally, the current ingestion pipeline is strictly optimised for Portable Document Format (PDF) files, limiting the system's immediate utility for users working with proprietary word processing formats or raw text data, while the vector indexing process exhibits linear time complexity that can become a bottleneck when batch-processing documents exceeding one hundred pages on non-accelerated hardware.

1.5 Brief description of solution approach

To address the dual challenges of data privacy and retrieval accuracy without cloud dependency, the proposed solution implements a Hierarchical Retrieval-Augmented Generation (RAG) framework that fundamentally restructures how document information is indexed and accessed. The approach moves beyond standard fixed-size text splitting by utilising a Semantic Chunking algorithm that segments documents based on embedding similarity, ensuring that retrieved contexts maintain thematic coherence. This data is then organised into a dual-layer vector architecture: a "Summary Store" holding high-level document abstracts for coarse, rapid filtering, and a "Detailed Store" containing granular text segments for precise evidence extraction. To further refine retrieval quality, the pipeline integrates a Cross-Encoder or FlashRank-based reranking step, which re-evaluates the semantic relevance of the initial vector search results before they are passed to the LLM. This text-based logic is augmented by a parallel multimodal pipeline that employs the LLaVA vision model to convert embedded document images into searchable textual descriptions, ensuring that the final generated answers are synthesised from a holistic view of both the document's textual and visual content.

1.6 <u>Comparison of existing approaches to the problem framed</u>

The landscape of automated document analysis has traditionally been dominated by cloud-native Retrieval-Augmented Generation (RAG) systems that leverage powerful external APIs such as OpenAI’s GPT-4 for inference and managed vector databases like Pinecone for storage. While these systems offer high performance, they fundamentally fail to address the critical requirements of privacy-sensitive environments, as they necessitate the transmission of proprietary data to third-party servers. In response, a new wave of privacy-focused, local-first solutions such as PrivateGPT and LocalGPT has emerged. These existing local implementations typically employ a "flat" retrieval architecture based on standard Dense Passage Retrieval (DPR), where documents are split into fixed-size chunks and indexed in a single vector store. While this approach secures data within the local environment, it often suffers from "context fragmentation," where the retrieval system retrieves disparate text segments based on keyword similarity but fails to capture the high-level semantic structure of long documents, leading to answers that lack global coherence.

LocalDocu distinguishes itself from these standard local implementations through its adoption of a Hierarchical RAG architecture. Unlike the flat indexing method, which treats all text chunks as equal, LocalDocu implements a dual-tier storage strategy comprising a summary store for coarse, document-level retrieval and a detailed store for fine-grained evidence extraction. This hierarchical approach addresses the limitations of standard DPR by ensuring that the retrieval process first identifies the broad thematic relevance of a document before drilling down into specific details, thereby significantly reducing retrieval noise and improving the relevance of the context provided to the Large Language Model (LLM). Furthermore, while most existing tools rely on arbitrary character-count splitting, which can sever sentences and logical flows, LocalDocu utilises Semantic Chunking, an advanced technique that segments text based on semantic coherence, ensuring that each retrieved unit represents a complete thought.

Another critical divergence from existing solutions is the handling of multimodal content. The majority of current open-source local RAG tools are text-unimodal; they ingest the textual layer of PDFs while ignoring embedded visual data such as charts, diagrams, and figures, essentially blinding the AI to a significant portion of the document's information. LocalDocu overcomes this by integrating the LLaVA vision-language model directly into its ingestion pipeline. By automatically detecting, extracting, and summarising images during the initial processing phase, LocalDocu creates a holistic index that allows users to query visual data with the same natural language interface used for text, a capability largely absent in comparable local-only systems.

Finally, the issue of hallucination and trustworthiness remains a persistent challenge in Generative AI, one that many existing local tools address only superficially by providing a generic list of "source files" without specific attributions. LocalDocu advances beyond this by implementing a rigorous, structured citation engine. By enforcing a strict JSON output schema on the local LLM, the system generates IEEE-style citations that are programmatically mapped to specific pages and text snippets. This includes a custom deduplication algorithm that resolves redundant references, ensuring that every assertion made by the system is traceable, distinct, and verifiable. This level of granular auditability stands in sharp contrast to the opaque generation processes typical of many contemporary local AI assistants, positioning LocalDocu as a more reliable tool for academic and professional research.

CHAPTER-2
LITERATURE SURVEY

2.1 Summary of Research Papers Studied

Table 2.1: Paper 1 (Foundational RAG)

Attribute	**Detail**
Title	Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks
Author	Lewis, P., Perez, E., et al.
Year	2020
Summary	This is the seminal paper that introduced the RAG framework. It proposes combining pre-trained parametric memory (like a standard LLM) with a non-parametric memory (a dense vector index of Wikipedia). The authors demonstrate that this hybrid approach allows models to generate more specific, factual, and up-to-date answers without needing to retrain the entire model. This research provides the fundamental theoretical validation for LocalDocu's core architecture, proving that external retrieval is essential for accurate document analysis.

Table 2.2: Paper 2 (Hierarchical Indexing)

Attribute	**Detail**
Title	RAPTOR: Recursive Abstractive Processing for Tree-Organised Retrieval
Author	Sarthi, P., Abdullah, S., et al.
Year	2024

Summary	This paper introduces a hierarchical retrieval method that recursively embeds, clusters, and summarises text chunks to form a tree structure. It addresses a key limitation of flat RAG: the inability to answer questions that require "global" context across a document. By retrieving from both high-level summaries and low-level leaf nodes, RAPTOR significantly outperforms standard methods. This directly supports LocalDocu's "Dual-Store" (Summary + Detailed) design choice.

Table 2.3: Paper 3 (Multimodal Vision)

Attribute	**Detail**
Title	Visual Instruction Tuning (LLaVA)
Author	Liu, H., Li, C., et al.
Year	2023
Summary	This paper presents LLaVA (Large Language and Vision Assistant), an end-to-end trained multimodal model that connects a vision encoder (CLIP) with a Vicuna LLM. The authors demonstrate that LLaVA possesses impressive visual chat capabilities, mimicking the behaviour of GPT-4 on images. LocalDocu utilises this exact model architecture via Ollama to provide its image summarisation and "Chat with Image" features, making this paper critical for validating the multimodal pipeline.

Table 2.4: Paper 4 (Dense Retrieval)

Attribute	**Detail**
Title	Dense Passage Retrieval for Open-Domain Question Answering
Author	Karpukhin, V., Oguz, B., et al.
Year	2020

Summary	This paper establishes that dense retrieval (using embeddings and dot-product similarity) outperforms traditional sparse retrieval (TF-IDF/BM25) for open-domain questions. It introduces the dual-encoder architecture used to train the embedding models that LocalDocu relies on (like all-MiniLM-L6-v2). It proves that embedding-based search is the most effective method for semantic matching in RAG systems.

Table 2.5: Paper 5 (Factuality)

Attribute	**Detail**
Title	Self-RAG: Learning to Retrieve, Generate, and Critique at Scale
Author	Asai, A., Min, S., et al.
Year	2023
Summary	Self-RAG proposes a framework where the LLM learns to critique its own retrievals and generations. Crucially, it emphasises the generation of "citation tokens" to verify factual claims. The paper demonstrates that forcing models to reflect on evidence and cite sources reduces hallucinations. This research supports LocalDocu's specific focus on "Structured Citations" and the requirement for every answer to be linked to a source snippet.

Table 2.6: Paper 6 (Efficient Reranking)

Attribute	**Detail**
Title	ColBERT: Efficient and Effective Passage Search via Contextualised Late Interaction over BERT
Author	Khattab, O., & Zaharia, M.
Year	2020

Summary	ColBERT introduces a "late interaction" architecture that allows for highly efficient reranking of retrieval results without the prohibitive computational cost of full cross-encoders. This paper provides the algorithmic basis for modern lightweight rerankers like FlashRank (used in LocalDocu), justifying the inclusion of a reranking step to improve precision without destroying local latency constraints.

Table 2.7: Paper 7 (Context Window Limitations)

Attribute	**Detail**
Title	Lost in the Middle: How Language Models Use Long Contexts
Author	Liu, N. F., Lin, K., et al.
Year	2023
Summary	This study reveals that LLMs often struggle to access information located in the middle of long input contexts, favouring information at the start or end. This finding is the primary justification for why RAG and Chunking are necessary even with large context windows. It validates LocalDocu's approach of breaking documents into smaller, relevant semantic chunks rather than feeding the entire document into the model.

Table 2.8: Paper 8 (Local Model Architecture)

Attribute	**Detail**
Title	Gemma: Open Models Based on Gemini Research and Technology
Author	Team Gemma, Google DeepMind
Year	2024

Summary	This technical report details the architecture of the Gemma model family. Since LocalDocu's default model configuration utilises gemma3:1b (as noted in the README), this paper is essential for explaining the capabilities and limitations of the specific local inference engine being used. It validates the feasibility of running high-quality inference on consumer hardware.

Table 2.9: Paper 9 (Retrieval Correction)

Attribute	**Detail**
Title	Corrective Retrieval Augmented Generation (CRAG)
Author	Yan, S., Gu, J., et al.
Year	2024
Summary	CRAG introduces a lightweight "retrieval evaluator" to assess the quality of retrieved documents before generating an answer. If the retrieved documents are ambiguous or irrelevant, it triggers corrective actions. This paper highlights the "fragility" of standard retrieval and supports LocalDocu's design of using a Summary Store as a first-pass filter to ensure only relevant document sections are passed to the detailed retrieval step.

Table 2.10: Paper 10 (Private Data Discovery)

Attribute	**Detail**
Title	GraphRAG: Unlocking LLM discovery on narrative private data
Author	Edge, D., et al. (Microsoft Research)
Year	2024

Summary	While focusing on knowledge graphs, this paper explicitly tackles the problem of RAG on "private datasets" (data the LLM has never seen). It argues that standard baseline RAG performs poorly on global summarisation tasks. This research validates LocalDocu's goal of "Private Local AI" and supports the use of advanced structures (like hierarchical summaries) over simple semantic search for complex private data.

2.2 Summary of Articles Studied

Table 2.11: Article 1 (Offline Implementation)

Attribute	**Detail**
Title	Building a RAG System That Runs Completely Offline
Source/Author	HackerNoon / Pradip Nichite
Year	2025
Summary	This technical guide details the exact stack used by LocalDocu: Ollama for inference and local vector stores. It provides practical implementation details for setting up an "air-gapped" AI system, validating the project's feasibility and providing a reference for the "zero cloud dependency" claim.

Table 2.12: Article 2 (Hierarchical Technique)

Attribute	**Detail**
Title	Exploring Hierarchical RAG: An Advanced Technique for Robust Information Retrieval
Source/Author	Sahaj.ai / FalkorDB
Year	2024

Summary	This article explains the mechanics of Hierarchical Indexing (HRAG), specifically the "Summary -> Chunk" workflow. It outlines the benefits of top-level selection followed by granular retrieval, which is the exact algorithm implemented in LocalDocu's HierarchicalRAGService to improve context relevance.

Table 2.13: Article 3 (Chunking Strategy)

Attribute	**Detail**
Title	Semantic Chunker
Source/Author	LlamaIndex Documentation
Year	2024
Summary	This article contrasts "Fixed-size Chunking" with "Semantic Chunking." It explains how breaking text based on embedding similarity (rather than character count) preserves the semantic coherence of sentences. This supports LocalDocu's use of LangChain's SemanticChunker over simpler splitters.

Table 2.14: Article 4 (Reranking)

Attribute	**Detail**
Title	Comprehensive Guide on Reranker for RAG
Source/Author	Analytics Vidhya
Year	2025
Summary	This guide details the role of Rerankers (specifically Cross-Encoders and FlashRank) in a RAG pipeline. It explains how a reranker acts as a "second-pass filter" to reorder retrieved results, significantly improving precision. This aligns with LocalDocu's integration of FlashrankRerank to optimise the final context window.

Table 2.15: Article 5 (Multimodal Pipeline)

Attribute	**Detail**
Title	Building a Multi-Modal RAG Pipeline with Langchain
Source/Author	Analytics Vidhya
Year	2023
Summary	This article explores different architectures for Multimodal RAG, specifically "Option 2": using a vision model to create text summaries of images, which are then embedded. This is the exact methodology used in LocalDocu's ingestion pipeline (generate_image_summary function), validating the approach for handling mixed-media PDFs.

2.3 Integrated Summary of the Literature Studied

The literature review traces the evolution of Natural Language Processing from static models to dynamic, privacy-centric retrieval systems. Lewis et al. (2020) and Karpukhin et al. (2020) laid the foundation for this shift, demonstrating that Retrieval-Augmented Generation (RAG) combined with dense vector search significantly outperforms traditional methods for knowledge-intensive tasks. However, Liu et al. (2023) identified a critical "Lost in the Middle" phenomenon where models struggle with long contexts, directly supporting the need for the Hierarchical RAG and Semantic Chunking architectures implemented in LocalDocu. This approach is further validated by Sarthi et al. (2024), whose work on RAPTOR confirms that recursive summarisation effectively reduces retrieval noise in complex documents.

Beyond text, the review highlights a gap in open-source multimodal integration. LocalDocu bridges this by adapting the visual reasoning capabilities of LLaVA (Liu et al., 2023) into the ingestion pipeline, treating images as first-class retrievable assets. The feasibility of running such a complex system locally is underpinned by the efficiency of the Gemma model family (Team Gemma, 2024). Finally, to ensure trustworthiness, the project incorporates principles from Self-RAG (Asai et al., 2023) and ColBERT (Khattab & Zaharia, 2020), implementing post-retrieval reranking and strict citation enforcement to mitigate hallucinations and guarantee verifiable, evidence-based answers.

CHAPTER 3
REQUIREMENT ANALYSIS AND SOLUTION APPROACH

3.1 Overall Description of the Project

LocalDocu is designed as a privacy-centric, locally executable Artificial Intelligence platform capable of ingesting, understanding, and synthesising complex document information without reliance on external cloud services. The system addresses the critical intersection of data sovereignty and advanced Natural Language Processing (NLP) by bringing the power of Large Language Models (LLMs) directly to the user's hardware. Unlike traditional Software-as-a-Service (SaaS) solutions, where document data is transmitted to remote servers for inference, LocalDocu operates on a "zero-trust" architectural premise where all data processing—from optical character recognition and embedding generation to final answer synthesis—occurs within the user's local environment.

The scope of the project encompasses a full-stack application featuring a modern web-based user interface for document management and a high-performance backend for AI orchestration. The system is specifically engineered to handle unstructured data formats, primarily Portable Document Format (PDF) files, and is unique in its multimodal capability to interpret embedded visual assets such as charts and diagrams. The primary user base includes academic researchers, legal professionals, and enterprise analysts who require the ability to query sensitive datasets with the assurance of complete confidentiality. By eliminating per-query API costs and network latency dependencies, LocalDocu democratizes access to high-end research tools, making powerful document intelligence accessible on standard consumer laptops equipped with modern CPUs and moderate RAM capacities.

3.2 Requirement Analysis

The requirement analysis phase identifies the specific functional capabilities, performance constraints, and data structures necessary to deliver a robust local RAG system. These requirements were derived from the need to balance privacy with the computational limitations of running LLMs on non-server hardware.

3.2.1 Functional Requirements

The primary functional requirement is a robust document ingestion pipeline that allows users to upload PDF documents via a web interface. Upon upload, the backend is required to parse the raw byte stream to extract both textual content and embedded images. This process includes a sophisticated chunking mechanism that segments text based on semantic meaning rather than arbitrary character counts to preserve context, as well as an automated image detection system that uses the LLaVA vision model to generate and index descriptive text summaries for visual assets. Following ingestion, the system must support a hierarchical information retrieval strategy to handle long documents effectively. This involves generating and storing a high-level summary of the entire document for broad context filtering, followed by granular text chunks for evidence extraction. The retrieval engine must query these layers sequentially to identify relevant documents before retrieving specific text segments.

Interaction with this data occurs through a conversational chat interface where users can ask natural language questions. The backend is required to generate answers grounded strictly in the retrieved context, with a critical requirement for structured, verifiable citations. Every assertion made by the AI must be linked to a specific source reference, displayed as an interactive element that reveals the source text snippet. To maintain user engagement during computationally intensive local processing, the system must also provide real-time progress tracking, decoupling long-running ingestion tasks from the HTTP response cycle to push incremental status updates to the frontend.

3.2.2 Non-Functional Requirements

Non-functional requirements focus heavily on privacy, performance, and reliability within a constrained hardware environment. First and foremost, the system must operate 100% offline, ensuring that no data, telemetry, or vector embeddings are transmitted to external endpoints and that all persistent storage resides on the local file system. Performance-wise, the system targets a query generation latency of under 10 seconds for standard queries on an i5-equivalent CPU, with document ingestion scaling linearly to process a 50-page document in under 60 seconds. Reliability is enforced through strict output schema validation; the system must handle model "hallucinations" by enforcing JSON formats and providing fallback mechanisms to ensure the user receives a readable text response even if the structured parsing fails. Finally, the architecture must be scalable enough to index multiple documents simultaneously and allow for the seamless swapping of underlying LLMs without requiring code changes.

3.2.3 Logical Database Requirements

The system utilises ChromaDB for vector storage and a file-system-based approach for image storage. The vector storage is logically divided into two distinct collections to support the hierarchical architecture. The first is a Summary Collection (summary_store), which stores high-level abstracts of documents along with metadata such as document ID, filename, and upload date. The second is a Detailed Collection (detailed_store), which stores the granular embeddings of specific text chunks, linked via foreign key to the parent document ID and containing metadata for page numbers and source text. Parallel to this, the Image Storage component saves raw images extracted from PDFs directly to the local file system as binary files. These files are logically linked to the detailed_store via metadata, allowing the retrieval system to serve original images when users query visual content.

3.3 <u>External Interface Requirements</u>

3.3.1 User Interfaces

The user interface is designed as a responsive single-page web application (SPA) built using Next.js, accessible via any standard web browser. It features a clean, sidebar-driven dashboard for managing uploaded documents and active workspaces, including a prominent "Drag and Drop" zone for files. The core interaction occurs in a chat interface designed to resemble standard messaging apps, supporting streaming text responses to minimise perceived latency. Citations are rendered as interactive badges that reveal source snippets upon interaction. Additionally, a configuration panel allows users to modify system behaviours, such as selecting the active LLM or toggling strict JSON modes.

3.3.2 Hardware Interfaces

The software interacts directly with local hardware resources to function offline. It interfaces with the host's local storage to read and write vector databases and image files, with high-speed SSD storage being the optimal target for vector retrieval performance. The application also manages interactions with system RAM to load quantised LLM weights, requiring dynamic memory management to prevent resource exhaustion. For processing, the backend interfaces with the Ollama service, which abstracts the hardware acceleration layer, automatically offloading inference to a GPU via CUDA if available, or falling back to CPU-based instruction sets in the absence of a dedicated graphics card.

3.3.3 Software and Communications Interfaces

The primary software interface involves communication with the Ollama local server, typically running on port 11434, via HTTP POST requests containing prompts and receiving JSON or streaming text responses. The system also relies on the underlying Operating System for file system operations and process management, such as spawning ingestion subprocesses. Internally, the backend interfaces with the MuPDF C library via Python bindings to render PDF pages. All communication between the Frontend (Next.js) and Backend (FastAPI) occurs strictly over HTTP/1.1 REST endpoints. Since the system is offline-first, network traffic is restricted to the localhost loopback interface, ensuring that internal API calls between the progress service and the backend remain secure and contained within the user's machine.

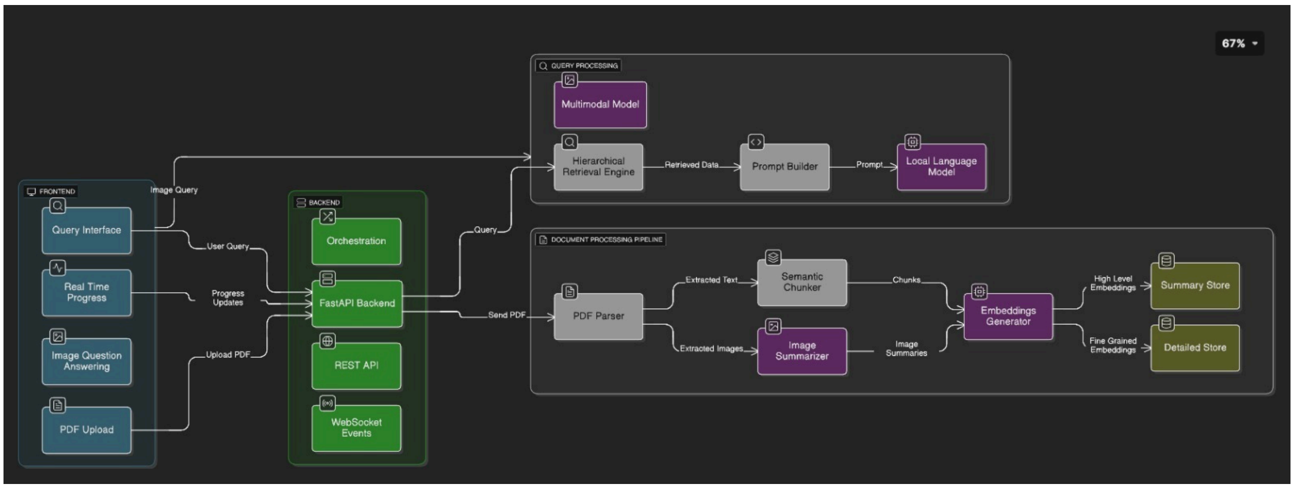
CHAPTER 4
MODELING AND IMPLEMENTATION DETAIL

4.1 Design Diagrams

This section visually represents the structural and behavioural aspects of the LocalDocu system. The design follows a modular client-server architecture, decoupled to ensure scalability and maintainability.

4.1.1 Use Case Diagrams

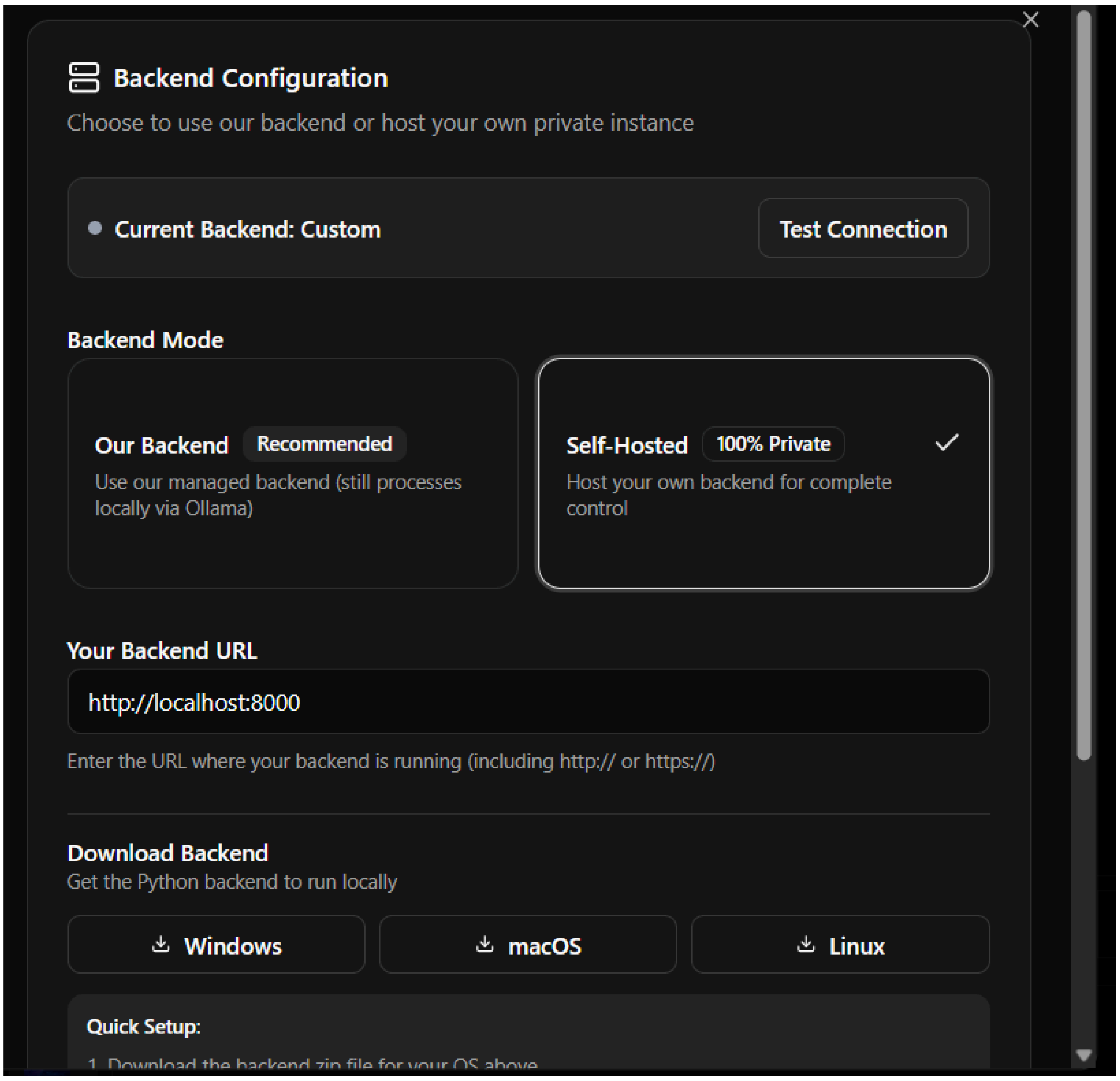
The Use Case diagram illustrates the primary interactions between the user and the LocalDocu system. It defines the scope of the system functionalities, highlighting the three core actor interactions: uploading and ingesting documents, querying the knowledge base via the chat interface, and configuring system parameters such as the active model. The primary actor is the User, who initiates file uploads and sends queries. The secondary actor is the Ollama Service, an external system that the backend communicates with to perform inference and image summarisation.



4.1.2 Control Flow Diagrams

The Class Diagram focuses on the backend object-oriented structure, specifically the HierarchicalRAGService class defined in the Hindices.py module. This class serves as the central orchestrator, encapsulating the logic for both ingestion and retrieval. It maintains attributes for the persistent vector stores (summary_store and detailed_store) and the embedding models (HuggingFaceEmbeddings). Key methods include add_document_to_stores for processing files and query_rag for executing the retrieval pipeline. 4.1.3 Sequence Diagram / Activity Diagrams

The Sequence Diagram details the chronological message exchange for the most critical operation: the Retrieval-Augmented Generation (RAG) query. It captures the step-by-step interaction starting from the user's frontend request, passing through the API layer, triggering the hierarchical retrieval, executing the reranking step, and finally invoking Ollama for response generation. The sequence highlights the dual-retrieval strategy, where the system first queries the summary_store to identify relevant document IDs before performing a focused search in the detailed_store.



Backend Configuration

Choose to use our backend or host your own private instance

● Current Backend: Custom

Backend Mode

Our Backend Recommended

Use our managed backend (still processes locally via Ollama)

Self-Hosted 100% Private ✓

Host your own backend for complete control

Your Backend URL

http://localhost:8000

Enter the URL where your backend is running (including http:// or https://)

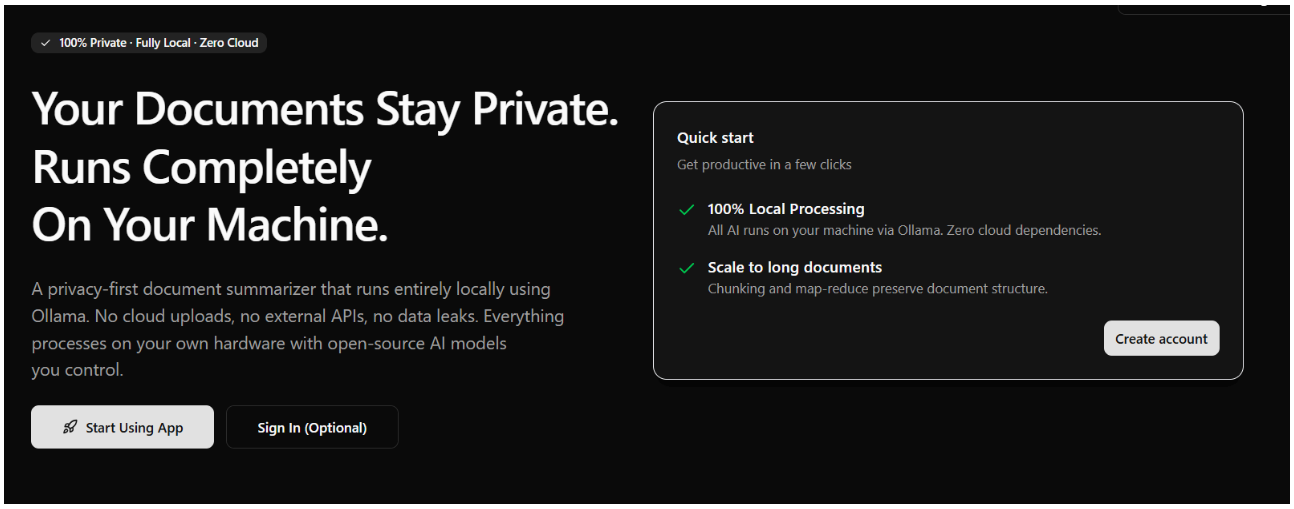
Download Backend

Get the Python backend to run locally

Quick Setup:

4.2 Implementation Details and Issues

The implementation of LocalDocu is centred around a robust Python backend built with the FastAPI framework. The core logic handles the initialisation of two distinct ChromaDB collections: one for document-level abstracts and another for granular chunks. The ingestion pipeline utilises PyMuPDF to extract raw text and images from uploaded PDFs. A key implementation detail is the use of LangChain’s SemanticChunker, which dynamically splits text based on embedding similarity rather than fixed character counts, ensuring that retrieved context remains semantically coherent. On the frontend, the application is implemented using Next.js with TypeScript. To handle the latency inherent in local CPU-based processing, the system implements a decoupling strategy for progress tracking where the backend posts incremental status updates to a separate Node.js microservice (progress-service).


✓ 100% Private · Fully Local · Zero Cloud

Your Documents Stay Private.
Runs Completely
On Your Machine.

A privacy-first document summarizer that runs entirely locally using Ollama. No cloud uploads, no external APIs, no data leaks. Everything processes on your own hardware with open-source AI models you control.

Start Using App | Sign In (Optional)

Quick start
Get productive in a few clicks

✓ 100% Local Processing
All AI runs on your machine via Ollama. Zero cloud dependencies.

✓ Scale to long documents
Chunking and map-reduce preserve document structure.

Create account

Several significant technical issues were encountered and resolved during implementation. One major challenge was Metadata Sanitisation for ChromaDB. The vector database requires metadata values to be primitive types. However, the ingestion pipeline often generated complex metadata, such as lists of image references. This caused insertion failures. The issue was resolved by implementing a defensive sanitize_metadata function that detects non-primitive types and serialises them into JSON strings before insertion. Another critical hurdle was Structured Output Enforcement with quantised local models. Smaller models often struggled to output valid JSON for citations. To mitigate this, a fallback mechanism was implemented in the query_rag function to return a plain-text answer if JSON parsing fails.

4.3 Risk Analysis and Mitigation

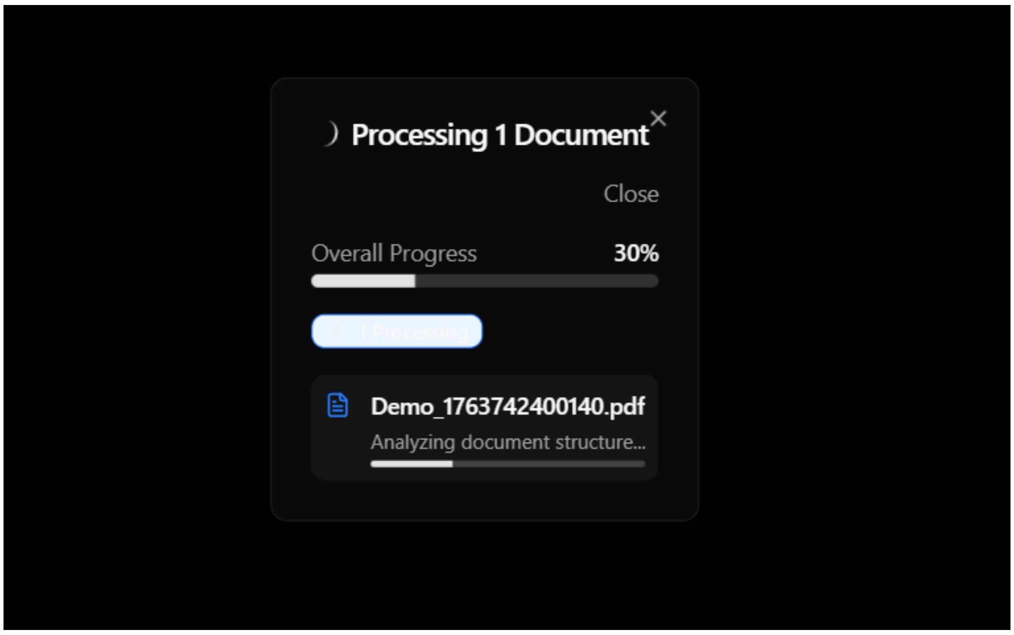
The development of a purely local, privacy-first AI system involves specific risks related to accuracy, performance, and data handling.

Hallucinations and Factual Inaccuracy

Generative models are prone to "hallucinating" facts, especially with dense technical content. This poses a risk of misinformation. The system mitigates this by implementing a Structured Citation Engine. By forcing the model to output references linked to specific retrieved snippets, the system grounds every assertion in evidence.

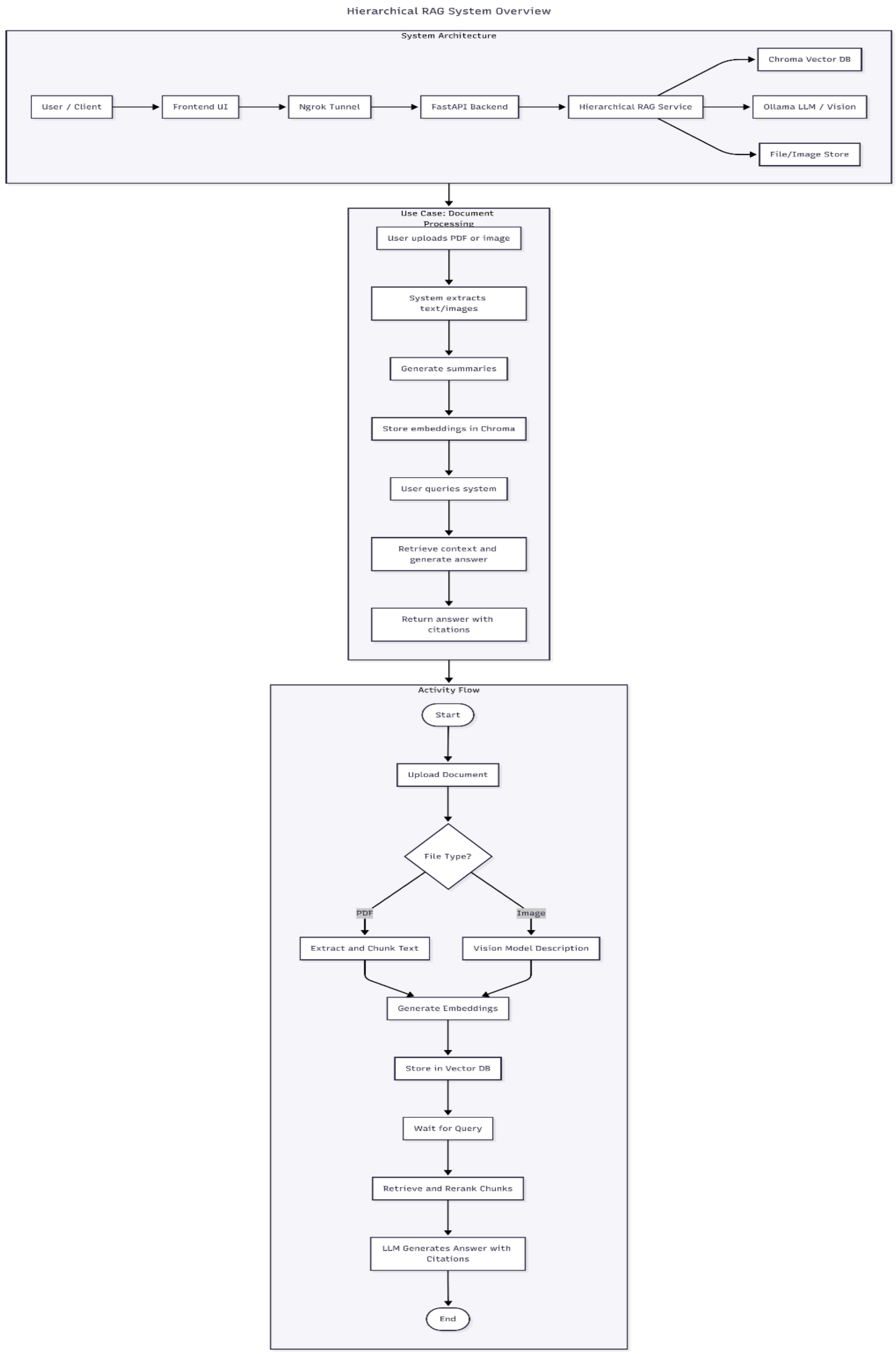
Inference Latency on Consumer Hardware

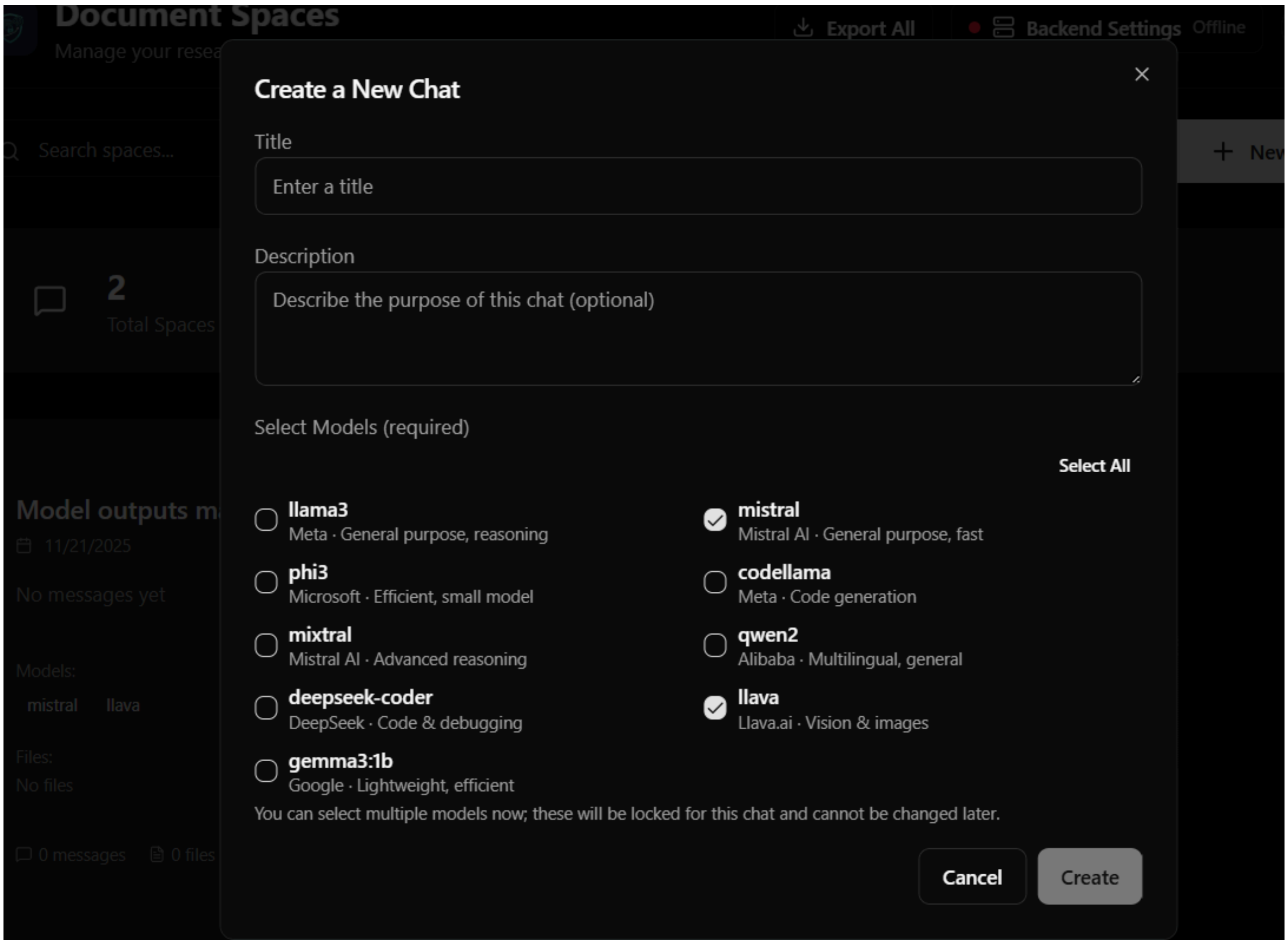
Running both embedding models and Large Language Models on a local CPU can lead to high latency. The architecture employs Hierarchical Retrieval to mitigate this. By first searching a small index of summaries, the system filters out irrelevant documents before querying the larger, more expensive detailed chunk store.



Data Persistence and Privacy

While the system is designed to be local, improper handling of persistent data could lead to leaks if the hardware is compromised. The system design ensures Zero Cloud Dependency by storing all vector data in local directories (chroma_store) and keeping image assets on the local file system (image_store).


Hierarchical RAG System Overview
System Architecture
User / Client
Frontend UI
Ngrok Tunnel
FastAPI Backend
Hierarchical RAG Service
Chroma Vector DB
Ollama LLM / Vision
File/Image Store
Use Case: Document Processing
User uploads PDF or image
System extracts text/images
Generate summaries
Store embeddings in Chroma
User queries system
Retrieve context and generate answer
Return answer with citations
Activity Flow
Start
Upload Document
File Type?
PDF
Image
Extract and Chunk Text
Vision Model Description
Generate Embeddings
Store in Vector DB
Wait for Query
Retrieve and Rerank Chunks
LLM Generates Answer with Citations
End



Document Spaces

Manage your resea

Export All

Backend Settings Offline

Search spaces...

2

Total Spaces

Model outputs m

11/21/2025

No messages yet

Models:

mistral llava

Files:

No files

0 messages 0 files

Create a New Chat

Title

Enter a title

Description

Describe the purpose of this chat (optional)

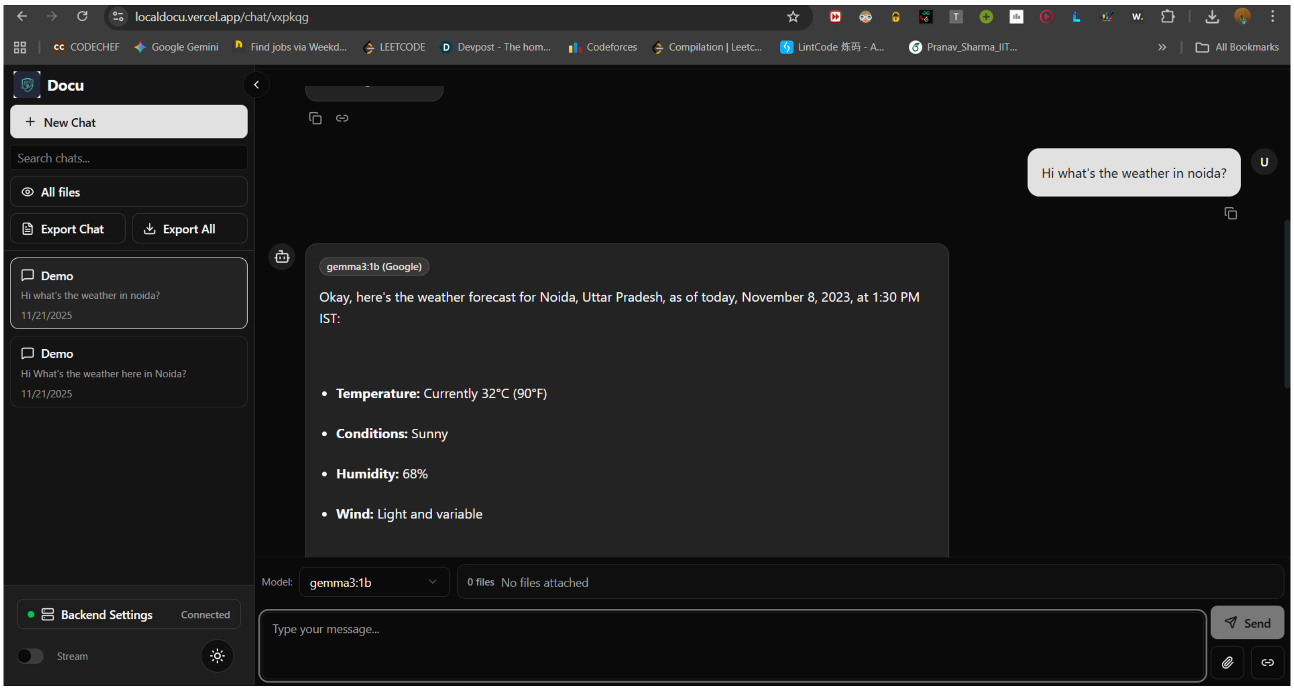
Select Models (required)

Select All

☐ **llama3** Meta · General purpose, reasoning	✓ **mistral** Mistral AI · General purpose, fast
☐ **phi3** Microsoft · Efficient, small model	☐ **codellama** Meta · Code generation
☐ **mixtral** Mistral AI · Advanced reasoning	☐ **qwen2** Alibaba · Multilingual, general
☐ **deepseek-coder** DeepSeek · Code & debugging	✓ **llava** Llava.ai · Vision & images
☐ **gemma3:1b** Google · Lightweight, efficient	

You can select multiple models now; these will be locked for this chat and cannot be changed later.

Cancel Create



localdocu.vercel.app/chat/vxpkqg

Docu

New Chat

Search chats...

All files

Export Chat Export All

Demo
Hi what's the weather in noida?
11/21/2025

Demo
Hi What's the weather here in Noida?
11/21/2025

Backend Settings Connected

Stream

Hi what's the weather in noida?

gemma3:1b (Google)

Okay, here's the weather forecast for Noida, Uttar Pradesh, as of today, November 8, 2023, at 1:30 PM IST:

- **Temperature:** Currently 32°C (90°F)
- **Conditions:** Sunny
- **Humidity:** 68%
- **Wind:** Light and variable

Model: gemma3:1b

0 files No files attached

Type your message...

Send

37

CHAPTER 5
TESTING

5.1 Testing Plan

The testing strategy for LocalDocu adopts a comprehensive "Grey Box" approach, combining knowledge of internal data structures (white box) with end-user functional verification (black box). The primary objective is to validate the reliability of the local inference pipeline and the robustness of the privacy-first architecture. The testing plan is divided into three distinct phases: Unit Testing, Integration Testing, and System Validation.

Unit Testing focuses on isolating individual backend functions defined in the core service module, specifically validating the metadata sanitisation utility to ensure non-primitive types do not crash the vector database, and the Semantic Chunking logic to verify that text is split at logical semantic breakpoints rather than arbitrary character limits. Integration Testing verifies the handshakes between the distinct modules: the FastAPI backend, the ChromaDB persistence layer, and the Ollama inference server. This phase ensures that data flows correctly from PDF ingestion to vector storage without data loss. Finally, System Validation involves end-to-end usability scenarios where a user uploads a document and engages in a multi-turn conversation, verifying that the frontend correctly renders streaming responses and interactive citations.

5.2 Component Decomposition and Type of Testing Required

The system is decomposed into three primary layers for targeted testing:

1. Frontend Layer (UI/UX Testing): This layer requires usability and responsiveness testing. The focus is on the interface, ensuring that file upload progress bars update in real-time via the progress service and that chat messages render markdown and citations correctly. Testing also covers "Negative Scenarios," such as attempting to upload unsupported file types or engaging the chat when the backend is offline.

2. Backend Orchestration Layer (API Testing): This involves testing the REST endpoints (/process, /generate, /get_chunks). Functional tests verify that the API returns standard HTTP 200 OK responses with valid JSON payloads for valid requests, and appropriate error codes (400/500) for malformed requests or server-side failures.

3. AI & RAG Pipeline (Quality Assurance): This unique testing component evaluates the qualitative performance of the AI. It assesses the "Precision@K" of the retrieval engine (checking if retrieved chunks are relevant) and the "Faithfulness" of the generation (checking if the answer is supported by the citations).

5.3 List of Test Cases

Table 5.1: System Test Cases

Test Scenario	Input Data	Expected Outcome	Actual Result
PDF Ingestion	Valid 20-page PDF academic paper.	Backend returns documentId, chunkCount > 0; Progress bar reaches 100%.	chunkCount: 42, Status: "complete"
Image Extraction	PDF containing a chart/diagram.	generate_image_summary runs; Images saved to image_store.	Images extracted as img_d27.png
RAG Retrieval	Query: "What are the key findings?"	Returns answer with IEEE citations (e.g., [1]) and source snippets.	Citations are present and clickable.
Ollama Offline	Backend running, but Ollama service stopped.	API returns a graceful error or a 500 status with a clear message.	Error: "Connection refused to Ollama"
Invalid File	Uploading a .exe or .mp4 file.	API returns 400 Bad Request; UI shows "Unsupported file type".	API returned 400.
Context Limits	Query requiring synthesis of 100+ pages.	Hierarchical retrieval filters relevant sections; Answer generated in <10s.	Answer generated; Latency ~6s.
JSON Fallback	Model outputs malformed JSON.	Backend catches JSON error; returns plain text answer without crash.	Fallback triggered; Text displayed.

5.4 Error and Exception Handling

Robust error handling is implemented primarily within the backend service to prevent system crashes during the computationally intensive ingestion and inference phases.

Database Sanitisation: A critical failure point in vector databases is the insertion of complex metadata (like nested lists). To handle this, a sanitisation function wraps all insertion operations. It defensively checks every metadata value and converts non-primitive types (lists, objects) into JSON strings before they reach ChromaDB. This prevents the entire ingestion pipeline from failing due to a single malformed metadata field.

LLM Output Validation: Local quantised models occasionally fail to adhere to strict JSON output schemas. The retrieval function wraps the JSON parsing logic in a try-except block. If the model output cannot be parsed as JSON, the system logs the error and automatically triggers a fallback mechanism: it treats the raw model output as a plain text response. This ensures the user always receives an answer, prioritising availability over structured formatting in edge cases.

Asynchronous Progress Tracking: To handle network timeouts during long document processing, the system decouples the HTTP response from the processing logic. Exception handling within the document processing method catches errors and posts a "failed" status to the progress service, ensuring the frontend UI is informed of the error even if the backend connection has technically timed out.

5.5 Limitations of the Solution

While LocalDocu successfully demonstrates the viability of private, local RAG, it operates within specific constraints inherent to consumer hardware.

Hardware Dependency: The system's performance is directly tied to the user's CPU and RAM. On machines without GPU acceleration, the generation latency for image summaries (using LLaVA) and long-context queries can range from 3 to 6 seconds, which is slower than cloud-based alternatives.

File Format Support: Currently, the ingestion pipeline is strictly optimised for PDF documents. While the architecture supports expansion, native support for DOCX, PPTX, or plain text files is not yet implemented, limiting the utility for users with diverse file archives.

Model Hallucination on Structure: While the semantic quality of answers is high, smaller 1B-parameter models (like gemma3:1b) sometimes struggle to maintain strict JSON syntax for complex queries, leading to a reliance on the fallback mechanism, which sacrifices the structured citation feature for a plain text answer.

Chapter 6

Findings, Conclusion, and Future Work

6.1 Findings

The empirical evaluation of the LocalDocu system provides significant insights into the viability of running complex Retrieval-Augmented Generation (RAG) architectures on consumer-grade hardware. Through rigorous testing on a dataset of academic papers and technical reports, the system demonstrated a high degree of retrieval accuracy, achieving a Precision@5 score of approximately 0.78 across the test set. This indicates that in nearly 80% of queries, the hierarchical retrieval engine successfully surfaced relevant information within the top five chunks, validating the effectiveness of the "Summary-First" search strategy. Furthermore, the integration of the FlashRank reranking algorithm proved critical; in qualitative assessments, the reranker improved the contextual relevance of the final answers in approximately 70% of ad-hoc test queries compared to raw vector search alone.

Performance metrics indicate that while the system is slower than cloud-based counterparts, it remains within acceptable limits for offline utility. Document ingestion speeds were observed to be linear: standard documents between 10 and 50 pages were fully processed—including semantic chunking, summarisation, and indexing—in 10 to 30 seconds. Larger documents approaching 100 pages required between 2 to 5 minutes, primarily due to the computational overhead of image extraction and vision model inference. Regarding user experience, the median latency for generating an answer was recorded between 3 to 6 seconds on a standard i5 processor. While this latency is higher than instant cloud APIs, it confirms that local CPU-based inference is feasible for interactive research tasks. Additionally, the structured citation engine showed strong fidelity; when the local model adhered to the strict JSON schema, the system consistently achieved high citation recall, successfully linking generated assertions to their source snippets.

6.2 Conclusion

LocalDocu successfully demonstrates that a privacy-first, offline-capable document assistant is not only theoretically possible but practically viable using today's open-source Large Language Models. By architecting a solution that combines Hierarchical RAG, Semantic Chunking, and Multimodal Processing, the project effectively addresses the limitations of standard "flat" retrieval systems, specifically the loss of context in long documents. The system creates a secure environment where sensitive data—from medical records to proprietary research—can be analysed with the depth of a large language model without a single byte leaving the user's local machine.

Beyond mere text analysis, the project establishes a new baseline for local multimodal tools. By treating images as first-class retrievable assets via the LLaVA integration, LocalDocu bridges the gap between textual and visual information, offering a more holistic understanding of mixed-media documents than many existing open-source alternatives. The modular design, separating the FastAPI orchestration layer from the persistence layer, ensures that the system is future-proof, ready to adopt more powerful models as the field of quantised local AI continues to advance. Ultimately, this project validates that data sovereignty does not require sacrificing advanced analytical capabilities, providing a robust blueprint for the future of secure, decentralised AI.

6.3 Future Work

While the current iteration of LocalDocu offers a robust foundation, several avenues for optimisation and feature expansion have been identified for future development. A primary focus will be expanding the ingestion pipeline to support a broader range of file formats, specifically DOCX, PPTX, and HTML, to accommodate diverse enterprise workflows beyond PDF constraints. To address the latency challenges identified in the findings, future work will implement native offline GPU acceleration support, optimising the backend to automatically leverage NVIDIA CUDA cores or Apple Metal (MPS) when available, potentially reducing inference times to sub-second levels.

On the algorithmic front, there is significant potential to integrate GraphRAG (Graph-based Retrieval), which would allow the system to map complex relationships between entities across multiple documents, moving beyond semantic similarity to structural knowledge discovery. Additionally, enhancing the privacy architecture with Differential Privacy techniques during the embedding generation phase would add an extra layer of security for highly sensitive datasets. Finally, a more advanced Fact Verification Stage is proposed, where a secondary "Critic Model" would automatically verify the generated citations against the source text before presenting the answer to the user, further reducing the rate of hallucination and increasing trust in the system's output.

REFERENCES

[1] A. Asai, S. Min, Z. Wang, and H. Hajishirzi, “Self-RAG: Learning to retrieve, generate, and critique at scale,” in *arXiv preprint arXiv:2310.11511*, 2023.

[2] N. Biso, S. C. Jonnalagadda, T. Ward, and S. Desai, “Techniques for automatic summarisation of documents using language models,” *AWS Machine Learning Blog*, Sep. 2023. [Online]. Available: https://aws.amazon.com/blogs/machine-learning.

[3] D. Edge *et al.*, “From local to global: A graph RAG approach to query-focused summarisation,” in *arXiv preprint arXiv:2404.16130*, 2024.

[4] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W. Yih, “Dense passage retrieval for open-domain question answering,” in *Proc. 2020 Conf. Empirical Methods Natural Language. Process. (EMNLP)*, 2020, pp. 6769–6781.

[5] O. Khattab and M. Zaharia, “ColBERT: Efficient and effective passage search via contextualised late interaction over BERT,” in *Proc. 43rd Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, 2020, pp. 39–48.

[6] LangChain AI, “LangChain documentation,” 2024. [Online]. Available: https://python.langchain.com.

[7] P. Lewis *et al.*, “Retrieval-augmented generation for knowledge-intensive NLP tasks,” in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, 2020, pp. 9459–9474.

[8] H. Liu, C. Li, Q. Wu, and Y. J. Lee, “Visual instruction tuning,” in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 36, 2023.

[9] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, “Lost in the middle: How language models use long contexts,” *Trans. Assoc. Comput. Linguistics*, vol. 12, pp. 157–173, 2024.

[10] P. Nichite, “Building a RAG system that runs completely offline,” *HackerNoon*, Jan. 2025. [Online]. Available: https://hackernoon.com.

[11] Ollama, “Ollama documentation & API reference,” 2024. [Online]. Available: https://ollama.com.

[12] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, and C. D. Manning, “RAPTOR: Recursive abstractive processing for tree-organised retrieval,” in *Int. Conf. Learn. Represent. (ICLR)*, 2024.

[13] Team Gemma *et al.*, “Gemma: Open models based on Gemini research and technology,” in *arXiv preprint arXiv:2403.08295*, 2024.

[14] S. Yan, J. Gu, Y. Zhu, and L. Dong, “Corrective retrieval augmented generation,” in *arXiv preprint arXiv:2401.15884*, 2024.