Research Companion 🤖

Gemma 3n Impact Challenge

Godugu Anil Himam

6 August 2025

Abstract

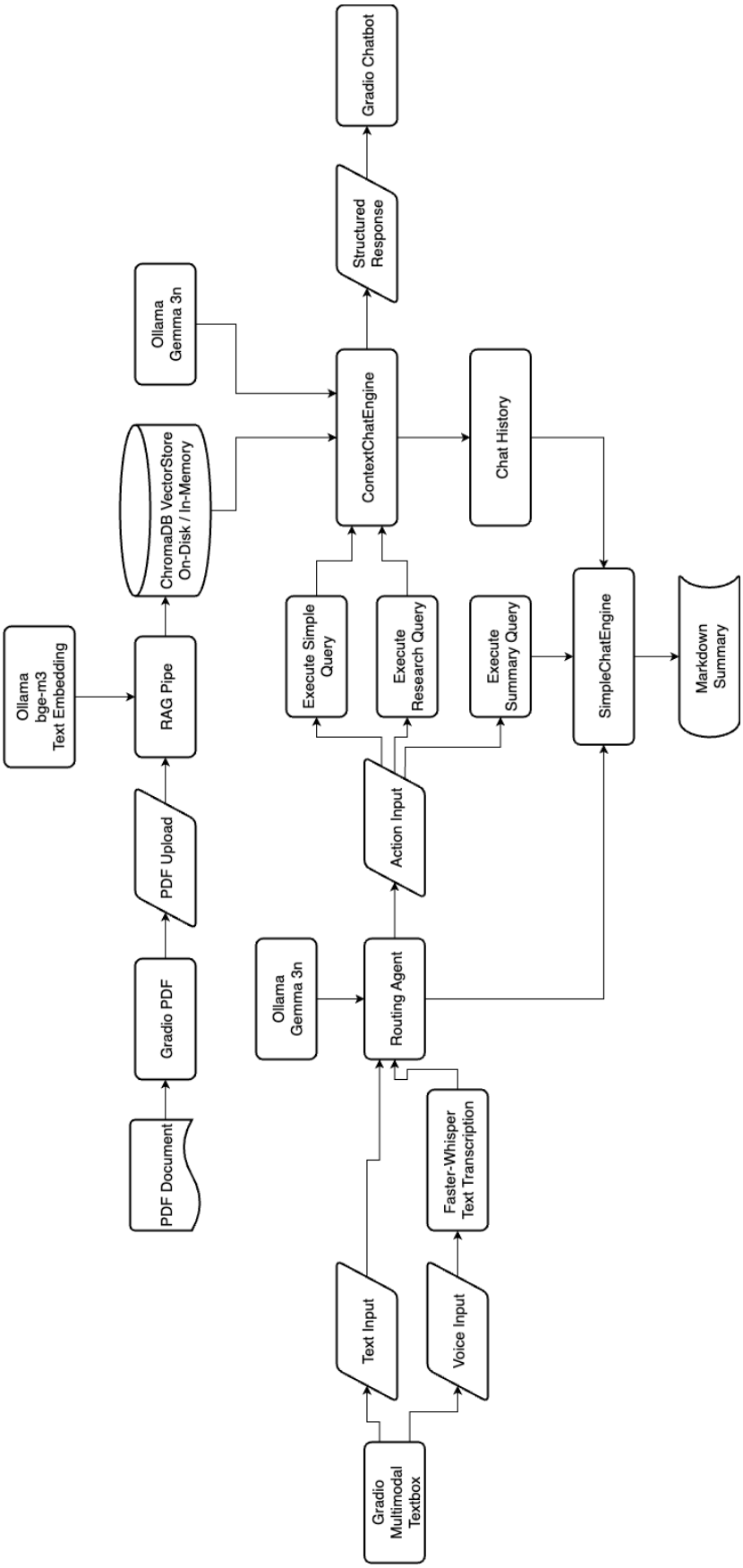
The Research Companion is a private, offline-first conversational AI agent, powered by Google's Gemma 3n. It is designed to accelerate the process of understanding complex technical documents. Leveraging a robust RAG pipeline built with LlamaIndex and a local bge-m3 embedding model, the system grounds responses in the provided source material, minimising hallucinations. At its core, a sophisticated ReAct routing agent powered by the Gemma 3n model intelligently selects the optimal response structure for a given query enabling adaptive, narrative-driven conversations. The application supports multimodal input via voice (through Faster-Whisper) and delivers a final, AI-generated summary of the entire research session, providing a tangible artefact for users.

Introduction

The process of academic and technical research is often gated by a formidable wall the Literature Review. Last year, during my undergraduate dissertation I experienced this firsthand. It was a frustrating bottleneck to learning and innovation where critical insights were buried in dense language and complex mathematics with very minimal footnotes. While modern AI tools offer a glimpse of a solution, they are often constrained by strict token limits, file sizes, high costs and significant privacy concerns.

The rise of powerful, open-source models presented a new frontier. This project was born from a simple question: Can we leverage a state-of-the-art model like Google's Gemma 3n to build a research partner that is not only powerful but also completely private, free, limited only by the hardware that the model ran on ? Research Companion is my answer. By building on the on-device capabilities of Gemma and the accessible Ollama ecosystem, this project aims to create a truly democratised tool for knowledge discovery for the next wave of researchers worldwide.

System Architecture



The system is designed to a modular, multi-stage and multi-step pipeline, as illustrated in the diagram above. The core components are:

- **The Multimodal Input Layer:** This is the entry point for the user. It is designed to be flexible, currently accepting both direct **Text Input** and **Voice Input**. Voice commands are transcribed into text using a local **Faster-Whisper** model. This ensures all inputs are normalised into a text format before parsing through Routing Agent.
- **The Agentic Core (ReAct Router):** This is the brain of the application, instead of a single, monolithic prompt, a **ReAct** agent acts as an intelligent router. It is powered by a local **Gemma 3n** model. It analyses the user's intent to select the most appropriate tool for the job. This allows the system to adapt its response strategy based on the user input. The ReAct agent chooses between a simple conversational response, a deep research response and a session summary response.
- **The RAG Knowledge Base:** This is the application's "long-term memory" for the document. When a PDF is uploaded, a robust RAG pipeline is triggered. The document is parsed, chunked, and then embedded using an offline **bge-m3** model served by **Ollama**. These embeddings are stored in a **ChromaDB** vector store, creating a searchable knowledge base that the **Research Query** tool can use to retrieve highly relevant context and ground the LLM's responses in fact.
- **The Chat Engines:** The system utilises two distinct chat engines for specialised tasks. The primary user interface is powered by a **ContextChatEngine**, which maintains conversational memory and synthesises responses by combining the user's prompt, chat history, and the retrieved RAG context to provide a structured output. For the final summary, a **SimpleChatEngine** is used. It is given the complete conversational history from the user's session and is prompted to produce a single, concise, and conceptually-driven markdown summary.

- **The Gradio User Interface**: The entire interactive experience is built and served by Gradio. It provides the essential bridge between the user and the complex, multi-stage backend, creating an intuitive and responsive web interface that runs entirely on the user's local machine. Gradio's event-driven model and state management capabilities play a critical role in orchestrating the application's asynchronous workflow. It manages the QueryEngine's state across multiple user interactions and handles the chained events (.click().then(...)) required for features like the one-click summary generation. Gradio serves to abstract away the backend complexity, presenting the user with a clean, powerful, and easy-to-use partner for their research.

Core Technical Components

This section details the key engineering decisions and implementations that power The Research Companion.

Structured and Adaptable Responses with Pydantic

At the core of a reliable conversational agent is the ability to generate predictable, structured output. I learnt this the hard way by working with naive LLM only responses at early stages of development. To move beyond simple text strings, this project leverages Pydantic data models to define a strict API contract with the LLM.

```
class SimpleResponse(BaseModel):
    """Class implements the expected structured response for a Simple Query to the LLM."""
    answer: str = Field(
        description="""A simple descriptive answer to the user's query."""
    )
```

```
class ResponseTypes(str, Enum):
    """Enumeration of all the response types."""
    RESEARCH = "research"
    SUMMARY = "summary"
    SIMPLE = "simple"
```

```python
class Citation(BaseModel):
    """Class implements the Expected Structured response for the LLM when citating other works."""
    page_number: int = Field(
        description="The exact page number from the document where the supporting text was found."
    )
    source_text: str = Field(
        description="The verbatim text chunk from the source document that supports the answer."
    )
    simplification: str = Field(
        description="""A brief simplification of the verbatim text chunk from the source
        document that supports the answer for better understanding."""
    )
```

```python
class ResearchResponse(BaseModel):
    """Class implements the Expected Structured response from the LLM's."""
    answer: str = Field(
        description="""The primary, comprehensive answer to the user's query.
        This text should be fully formatted in Markdown, including any equations using LaTeX syntax (eg. $ h=6.64 * 10 ^ {-34} $)."""
    )
    follow_up_questions: list[str] = Field(
        description="""A maximum of three follow-up questions that the user's query relates to
        as chain of thought prompts."""
    )
    citations: list[Citation] = Field(
        description="A list of all source citations from the document that were used to formulate the answer."
    )
```

The above images illustrate the code snippet that were used to construct each of the response structures used by system when generating a response. This approach ensured that the application's UI layer always received data in a consistent, validated format, eliminating the need for fragile string parsing. It allowed for the robust rendering of complex components like the **Follow - Up Chain of Thought** and the detailed, **Multi-part Citations** forming the foundation of the agent's ability to have narrative driven conversations.

The Agentic Core: A ReAct Router for Intelligent Tool Selection

A single prompt is insufficient for a truly interactive assistant. The user's intent can shift throughout the conversation from a simple query to a deep analytical request. To handle this, the application is orchestrated by a **ReAct** (Reason and Act) agent from LlamaIndex. This agent acts as an intelligent router, on receiving a query it is not immediately answered. The agent is prompted to think and select the most appropriate tool for the task.

The tools available to the agent are:

- **Simple Query Tool:** A lightweight tool designed for answering simple question, filling the conversation by outputting a **SimpleResponse** model.
- **Research Query Tool:** A workhorse tool designed for deep analysis and outputting the detailed **ReseachResponse** model. It contains several fields such as the answer field for the base answer, the follow up questions field for proactive follow up chain of thought prompts and a citation field that provides detailed citations supporting the base answer.
- **Summary Query Tool:** A specialised tool that takes the full conversation history and generates a final **SummaryResponse** model. It provides a creative title for the summary along with a raw markdown string that encompasses the concept-driven summary for the conversation.

The ReAct agent can dynamically adapt the application's response strategy by parsing the user input against the description of each tool guiding the LLM response generation process. It ensures that simple questions get fast answers while complex questions receive the full power of the RAG pipeline. This agentic layer is the key to creating a truly responsive and intelligent user experience.

The RAG Knowledge Base: ChromaDB and Custom Prompting

The agent's ability to answer questions about a document is powered by a robust Retrieval-Augmented Generation (RAG) pipeline. All the logic of the RAG pipeline is encapsulated in the **QueryEngine** object.

- **Ingestion:** Upon PDF upload, **SimpleDirectoryReader** parses the document.
- **Embedding:** The text chunks are then vectorised using a local bge-m3 model served via Ollama. This powerful open-source model was chosen for its excellent performance in semantic retrieval tasks.
- **Storage:** The resulting embeddings are persisted in a local ChromaDB vector store. ChromaDB was selected for its simplicity, performance, and ability to run

entirely on-disk, fulfilling the project's offline-first requirement. It was also chosen for its feature to index text chunks using both sparse and dense techniques.

To elicit the most insightful answers, the RAG pipeline does not use a default prompt. It uses a custom prompt - engineered template to guide with generation.

```
RAG_PROMPT_TEMPLATE = PromptTemplate(
    """You are a world-class research assistant with expertise in this domain.
    You are 'pro-active' and 'inquisitive' like Jarvis from Iron Man always ready to brainstorm research and churn ideas.
    Your primary goal is to provide a direct and concise answer to the user's question based on the provided context.
    After providing the answer, you must generate a maximum of three follow-up questions that the user might ask.
    Do not add any information that is outside the provided context.

    ------------------------------

    Context from the document:
    {context_str}

    ------------------------------

    Chat History:
    {chat_history}

    ------------------------------

    User's Question:
    {question}

    ------------------------------
    Answer: """
)
```

This template explicitly instructs the Gemma 3n model to act as a domain expert, synthesising information from the retrieved context with its own internal knowledge. It encourages the model to explain the "why" behind the facts and to provide attribution, resulting in responses that are not just summaries, but true explanations.

Contextual Memory and The Chat Engines

To enable a fluid, multi-turn conversation, the system utilises two specialised chat engines built on top of the RAG pipeline.

- The primary interface is powered by a **ContextChatEngine**. This engine is initialised with a **ChatMemoryBuffer**, allowing it to maintain the history of the conversation.

- The summary feature utilises a **SimpleChatEngine**. This engine is not designed for back-and-forth conversation. Instead, it is given the *entire* finalised chat history from the user's session as a single, large context. It is then prompted with a one-shot instruction to produce a comprehensive markdown summary, effectively transforming the conversational data into a structured report. The below image illustrates the prompt template utilised by the SimpleChatEngine for Summary Generation.

```
CONCEPT_DRIVEN_SUMMARY_PROMPT_TEMPLATE = """You are an Expert Research Tutor and Synthesizer. Your mission is to transform a conversation transcript into a rich,
    educational document. This document should not just summarize the chat, but *re-teach* the core concepts discussed, elaborating on
    them with your own deep knowledge.

    First, read the entire chat history to identify the user's main goal and the key technical or scientific concepts they
    explored. Then, generate a response following this precise structure:

    **Title:**
    [Create a new, original, and academic title that accurately reflects the core subject of the conversation.]

    **Conversation Abstract:**
    [Write a short, narrative paragraph that sets the stage. Describe the overall topic of the conversation
    and the user's primary objective. This should read like a brief introduction to a study guide, explaining the "why" behind
    the conversation.]

    **Key Concepts Revisited:**
    [This is the core of your task. For each major concept, topic, or question the user raised, create a dedicated section. Use the
    original question or a descriptive topic as a header. Then, provide a detailed, standalone explanation.]

    ---

    **Concept:** [Name of the first major concept]

    **In-Depth Explanation:** [Here, provide a detailed, clear, and comprehensive explanation of the concept. **Do not just repeat
    the chat.** Use your own expert knowledge to provide rich context, analogies, definitions, and even mathematical formulations if
    relevant (using LaTeX syntax like $E=mc^2$). Your goal is to provide a definitive, standalone explanation that someone could use to
    study from, filling in any gaps from the original conversation.]

    ---

    [Repeat this "Concept" and "In-Depth Explanation" structure for every major topic discovered in the chat history.]

    ---
    """
```

Why Gemma 3n and Ollama

The selection of the core technology stack was a deliberate choice to best suit the project's ambitious goals of creating a powerful, private, and accessible AI partner. Gemma 3n and Ollama were not just tools they were the evident building blocks for the foundation of the project's vision.

Why Gemma 3n: The On-Device Reasoning Engine

Gemma 3n was the unequivocal choice for the core LLM for several key reasons that align perfectly with the needs of a sophisticated agentic application:

- **State-of-the-Art Reasoning and Instruction Following:** The success of the ReAct routing agent and the reliability of the structured Pydantic Data Model output are directly attributable to Gemma's powerful instruction-following capabilities. The model's ability to understand and adhere to the complex prompts generating response by combining its own knowledge with retrieved is a testament to its SOTA reasoning and instruction following capabilities.
- **On-Device Performance and Architecture:** To achieve a truly private, offline-first experience, the model had to be performant on consumer hardware. Gemma's architecture which includes innovations like the MatFormer, is optimised for this exact use case. It provides the high-level reasoning of a much larger model in a compact, efficient package that can be run locally without requiring specialised accelerators.
- **Large Context Window:** A research assistant must be able to handle dense, multi-turn conversations and large amounts of retrieved context. Gemma's large context window (up to 32k tokens) provides the necessary "working memory" for the ContextChatEngine to maintain coherence and for the SimpleChatEngine to process an entire conversation in a single pass.
- **Native Multimodal Capabilities:** While the final implementation used a pragmatic workaround for image RAG due to environmental challenges, Gemma was chosen from the outset for its native multimodal potential. The architecture was designed to leverage these capabilities, and the **Future Work** section outlines a clear path to integrating Gemma's vision understanding directly.

Why Ollama: The Democratisation Layer

The project's mission is to democratise access to knowledge. The Ollama ecosystem is the perfect embodiment of that same mission for AI models.

- **Offline-First and True Privacy:** Ollama is the critical component that makes the private, no-limits promise a reality. By serving both the Gemma 3n LLM and the bge-m3 embedding model from a local server, it ensures that no user data, no queries, and no documents ever leave the user's machine.
- **Accessibility and Simplicity:** Ollama abstracts away the immense complexity of managing model weights, dependencies, and hardware acceleration. It makes running a state-of-the-art model as simple as an Ollama Pull request. This philosophy of accessibility and simplicity was a direct inspiration for the Research Companion's user-friendly design.
- **Cross-Platform Hardware Acceleration:** A key challenge in local AI is performance. Ollama's ability to automatically detect and utilise the appropriate hardware acceleration (Metal on macOS, CUDA/ROCm on Linux/Windows) is a game-changing feature that makes this project feasible for a wide range of users without requiring them to become performance engineering experts.

Challenges Overcome

Building a sophisticated, offline-first AI application under a tight deadline presented several significant engineering challenges. Over the last three weeks I was successful in navigating several of these issues. It required pragmatic decision-making and a focus on delivering a robust, functional product a safe core submission for the hackathon. Nothing came easy and in my case I am proud to share that I have experienced many firsts throughout this project.

The Multi-Modal RAG Instability Problem

The Challenge: The initial vision for the project included a full multi-modal RAG pipeline using vision embeddings from a CLIP model to analyse images within documents. However, the implementation was immediately blocked by a persistent and critical environmental issue. The default ClipEmbedding downloader provided by the Open-AI Clip library repeatedly failed due to network corruption, resulting in a

series of SHA256 Checksum mismatch errors. This was not a code issue, but an infrastructure and dependency problem that could not be solved by retries.

The Solution: Instead of losing valuable time on a single, fragile dependency, I made a strategic decision to de-risk the feature while still delivering the core user value. I pivoted to a more robust, text-based approach:

- The PDF is parsed, and extracted using the PyMuPDF library.
- The extracted text was then embedded using the standard bge-m3 model and indexed in the ChromaDB vector store with metadata linking them back to their source page.

This solution successfully allows users to ask questions about figures and diagrams, fulfilling the core feature requirement in a reliable and stable manner.

Taming Latency in a Local Environment

The Challenge: Initial prototypes of the RAG pipeline, while accurate, suffered from high latency (60-300 seconds per query or timeout). This was unacceptable for a conversational interface. The bottleneck was identified as the inference time of the local LLM when generating complex, structured JSON responses. It also lead to unnecessary increase in token consumption that blew through the LLM's context length.

The Solution (Agent Routing): I implemented several optimisations to attack both actual and perceived latency:

- **Quantisation:** The first and most impactful step was using a 4-bit quantised version of the Gemma 3n model, which dramatically reduced the computational overhead with minimal impact on quality.
- **The Agentic Router:** The most significant architectural solution was the implementation of a ReAct routing agent. This agent classifies user intent and routes queries to specialised tools. Simple conversational queries are sent to a

lightweight engine that returns a simple text string, providing low latency responses. Only complex, analytical queries trigger the full RAG pipeline with the detailed, structured ReseachResponse model. This matches the computational cost to the user's need, making the application feel fast and intelligent.

By combining these techniques, the application's responsiveness was improved by an order of magnitude, making a fluid, conversational experience possible on local hardware.

While during final testing the agent still had an average latency of 60 - 300 seconds depending on the length of the conversation the agent was stable and provided reliable responses throughout the test. I believe with increasingly powerful hardware we can reduce the latency time soon.

Future Work

The key ideas for the future are as follows:

- Fully Multimodal RAG Pipeline: Extending the RAG pipeline to provide rich insight into figures, graphs and other image information present in documents.
- Refactoring the Multimodal Input Pipeline: To capture the native multi-modal capabilities of Gemma 3n through HuggingFace Transformers.
- Extending the Multimodal input suite with Images: To provide users with auxiliary workflows where they can upload hand-written shorts of their learnings, understandings and diagrammatic representations. The model could then tutor the user based using its knowledge pipeline.
- Personalised Character Vector: Implementing a personalised character vector to tailor the interactions with AI assistant to individual users.
- Gemma 3n Fine-Tuning: Creating a fine-tuned Gemma 3n model that excels in Research Analysis, Graphs, Diagrams and Bibliography Management.

- Voice Input: Completion of the Conversational Interface by applying prompting techniques to generate a concise summary of the result from the LLM for voice output.