Concept Paper

Building a Robust Retrieval-Augmented Generation Chatbot for Immigration Knowledge Base Using Google Cloud Vertex AI and Open-Source LLMs

Ajay kumar *

Posted Date: 28 May 2025

doi: 10.20944/preprints202505.2219.v1

Keywords: LLM; RAG; AI

 doi:10.20944/preprints202505.2219.v1

Article

Building a Robust Retrieval-Augmented Generation Chatbot for Immigration Knowledge Base Using Google Cloud Vertex AI and Open-Source LLMs

Ajay Kumar

Independent Researcher; ch.ajay1707@gmail.com

Abstract: This paper details the end-to-end development and implementation of a Retrieval-Augmented Generation (RAG) chatbot designed to provide accurate and grounded answers from a custom knowledge base on immigration policies and procedures. Leveraging Google Cloud's Vertex AI for robust text embedding and vector search capabilities via Matching Engine, the system demonstrates an effective architecture for combating the inherent limitations of large language models (LLMs), such as factual inaccuracies and knowledge cut-off dates. The development journey highlights practical challenges encountered during LLM integration, including free-tier service access limitations with Google's proprietary models and task-specific API constraints with hosted open-source solutions like Hugging Face's Inference API. Ultimately, the project pivots to a strategy of local LLM deployment, providing valuable insights into the full lifecycle of a RAG system and the critical considerations for productionizing such applications within cloud environments, particularly concerning resource management and cost optimization.

Keywords: LLM; RAG; AI

1. Introduction

Navigating the complexities of immigration policies and procedures can be an overwhelming task for individuals seeking clarity on visa requirements, application processes, and eligibility criteria. Traditional methods of information retrieval, such as searching official government websites or legal databases, often present users with vast amounts of dense, technical jargon, making it difficult to extract precise and relevant answers to specific queries. While the advent of large language models (LLMs) offers a seemingly intuitive approach to answering questions, their inherent limitations, including a knowledge cut-off date and a propensity for generating plausible but factually incorrect information (often termed *'hallucinations'*), render them unreliable for domains requiring high factual accuracy and up-to-date information, such as immigration law. [1,2]

2. Background

2.1. Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) represents a transformative architectural paradigm in natural language processing, specifically designed to enhance the reliability and factual accuracy of Large Language Models (LLMs) [3]. Fundamentally, RAG addresses the inherent limitations of standalone LLMs, which, despite their impressive generative capabilities, are susceptible to producing *'hallucinations'* (factually incorrect but confidently presented information) and possess knowledge restricted to their training data's cut-off date. By integrating an external, authoritative knowledge base into the LLM's inference process, RAG systems ensure that generated responses are not only coherent and fluent but also explicitly grounded in verifiable, real-time information.

The RAG framework operates in two distinct, yet interconnected, phases: [3] Retrieval Phase: When a user poses a query, the RAG system first performs a search against a vast, external knowledge

 doi:10.20944/preprints202505.2219.v1

base. This knowledge base is typically pre-processed, with its documents or text segments (*'chunks'*) converted into numerical vector embeddings. The user's query is also converted into an embedding using the same embedding model. A vector database then efficiently identifies the most semantically similar chunks from the knowledge base to the query. This process ensures that only the most relevant pieces of information are considered for generating a response.

Generation Phase: The retrieved relevant text chunks are then appended to the original user query, forming an augmented prompt. This enriched prompt is then fed to an LLM. The LLM is instructed to synthesize a comprehensive and accurate answer, drawing exclusively from the provided retrieved context. This grounding mechanism constrains the LLM's output, significantly reducing the likelihood of hallucinations and enabling the generation of responses that are directly supported by the current and domain-specific information from the knowledge base.

This hybrid approach not only enhances the factual integrity and trustworthiness of LLM-generated content but also offers the crucial benefit of *'explainability'* or *'attributability,'* as the system can often point to the specific source documents that informed its answer. This is especially vital in critical applications like legal or governmental information, where verifiability and up-to-date content are paramount.

2.2. Large Language Models (LLMs)

Large Language Models represent a significant leap in artificial intelligence, characterized by their immense scale, often containing billions or even trillions of parameters. These deep learning models are trained on colossal datasets encompassing a vast spectrum of internet text and code, enabling them to comprehend, generate, and interact with human language in remarkably nuanced ways. The Transformer architecture, a cornerstone of many modern LLMs, was introduced by Vaswani et al. [4]. Their capabilities extend beyond simple pattern matching to include complex reasoning, contextual understanding, and even creative writing, making them highly versatile for applications like sophisticated chatbots, content generation, and code assistance. Prominent examples include Google's internally developed Gemini series [5,6], OpenAI's GPT family [1,7,8], and a rapidly growing ecosystem of powerful open-source models such as Mistral [9], Llama [10,11], and Gemma [12].Despite their impressive fluency and generalized knowledge, LLMs, when used in isolation, present notable limitations for applications demanding high factual integrity and real-time information. A primary concern is their *'knowledge cut-off date,'* meaning their understanding of the world is limited to the data they were trained on, making them inherently incapable of providing current information on recent events or evolving policy changes. More critically, LLMs are susceptible to *'hallucinations'* – a phenomenon where they confidently generate plausible-sounding but entirely fabricated or incorrect information. This tendency is particularly problematic in sensitive domains, such as legal, medical, or governmental information, where precision, verifiability, and adherence to established facts are paramount. Relying solely on a standalone LLM for critical information can lead to significant misinformation and detrimental consequences, underscoring the need for supplementary mechanisms to ground their outputs in verifiable knowledge.

2.3. Vector Databases and Embeddings

The paradigm shift in information retrieval towards semantic understanding is fundamentally driven by the synergy of text embeddings and specialized vector databases. Text embeddings are dense, numerical representations (vectors) of textual data, ranging from individual words to entire documents, generated by sophisticated neural networks known as embedding models. These models are trained to map semantically similar pieces of text to proximate locations within a high-dimensional vector space. Consequently, the *'distance'* between two vectors in this space directly correlates with the semantic similarity of their corresponding texts. For the immigration knowledge base, Google's Vertex AI Text Embedding API, specifically the `text-embedding-004` model, was chosen for its state-of-the-art performance, broad language support, and native integration within the Google Cloud

 doi:10.20944/preprints202505.2219.v1

ecosystem, ensuring the embeddings accurately capture the nuanced meanings within legal and policy documents.

Efficiently querying these vast collections of high-dimensional vectors to find the most similar ones requires specialized infrastructure beyond traditional relational databases. This is where vector databases, or vector search services, come into play. These systems are optimized for performing Approximate Nearest Neighbor (ANN) searches, which efficiently identify vectors closest to a given query vector, even across datasets containing millions or billions of items. Vertex AI Matching Engine stands out as a highly scalable, fully managed vector database service provided by Google Cloud. It excels at handling real-time, ultra-low latency similarity queries, making it an ideal choice for the retrieval component of our RAG system. By indexing the pre-computed embeddings of our immigration knowledge base, Matching Engine enables the rapid identification of the most semantically relevant text chunks to a user's query, forming the critical first stage of the Retrieval-Augmented Generation process. [3]

2.4. Google Cloud Vertex AI

Google Cloud's Vertex AI platform is a unified machine learning development platform designed to streamline the entire ML lifecycle, from data ingestion and model training to deployment and monitoring. Its comprehensive suite of tools and services makes it a robust choice for developing and scaling AI-powered applications. Within the context of this RAG chatbot project, several Vertex AI components were strategically employed to build a scalable and efficient system.

Central to the embedding generation process was the `vertexai.language_models.TextEmbeddingModel`. This managed API service provides access to pre-trained, state-of-the-art embedding models like `text-embedding-004`, eliminating the need for users to host or manage their own embedding infrastructure. Its consistent embedding space ensures that both the knowledge base chunks and incoming user queries are represented in a semantically comparable manner, which is crucial for accurate similarity search.

For the core vector search functionality, Vertex AI Matching Engine was indispensable. As a fully managed approximate nearest neighbor (ANN) service, it simplifies the deployment and management of high-scale vector indexes. The platform handled the complexities of indexing the millions of immigration text embeddings, automatically managing the underlying infrastructure and providing a low-latency endpoint for real-time similarity lookups. This allowed the RAG system to quickly identify the most relevant information chunks for any given user query without significant operational overhead.

Furthermore, Google Cloud Storage (GCS) served as a versatile and highly durable object storage solution. It acted as the primary repository for the raw immigration knowledge base data, the intermediate JSON files containing the embedded chunks, and ultimately, the source for data ingestion into Vertex AI Matching Engine. Its tight integration with other Google Cloud services facilitated seamless data flows and accessibility for both the embedding pipeline and the live RAG application, where chunk content was retrieved in-memory for prompt augmentation.

3. Methodology and Implementation

The development of the RAG chatbot for the immigration knowledge base involved a multi-stage process, encompassing data preparation, vector database construction, and the iterative design of the core RAG application logic. This section delineates each component, emphasizing the technical choices and the evolving LLM integration strategy.

 doi:10.20944/preprints202505.2219.v1

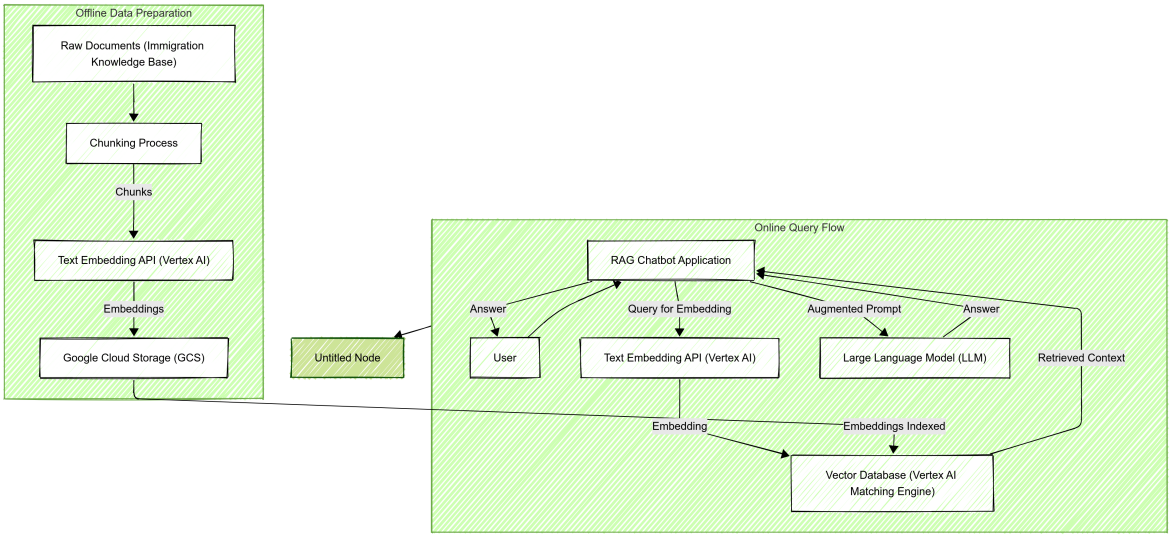

Figure 1. Overall RAG System Architecture.

3.1. Data Preparation

The foundational step involved preparing the raw immigration knowledge base. This dataset, comprising extensive documentation on visa requirements, application procedures, and policy guidelines, was initially in various textual formats. To make it suitable for embedding and retrieval, a critical pre-processing step was text chunking. Large documents were systematically segmented into smaller, coherent text units. The rationale behind chunking is twofold: firstly, it ensures that the retrieved content fits within the context window limitations of typical LLMs; secondly, smaller, semantically focused chunks lead to more precise retrieval, reducing the noise of irrelevant information that might otherwise be present in larger documents. Each chunk was assigned a unique identifier to facilitate lookup. While specific chunking parameters can vary, a strategy focusing on maintaining semantic coherence within each chunk (e.g., splitting by paragraphs, or by fixed token counts with overlap) was employed to optimize retrieval accuracy.

3.2. Embedding Generation

Following data chunking, each individual text chunk was transformed into a high-dimensional numerical vector, or embedding. This process was executed using Google Cloud's Vertex AI Text Embedding API, specifically leveraging the `text-embedding-004` model. This model, part of the `vertexai.language_models` suite, is recognized for generating high-quality embeddings that accurately capture the semantic meaning of text segments. The Python script iterated through all prepared text chunks, sending them in batches to the Vertex AI Embedding API to generate their respective 768-dimensional vector representations.

The output from this process—each chunk's unique ID, its original text content, and its corresponding embedding vector—was serialized into JSON format. These JSON files were then uploaded to a designated Google Cloud Storage (GCS) bucket, configured as gs://immbot-us-central1/immigration-knowledge-base-embedded/. GCS served as a resilient and scalable staging area, not only for persistent storage of these embeddings but also as the direct input source for the subsequent Matching Engine index creation.

 doi:10.20944/preprints202505.2219.v1

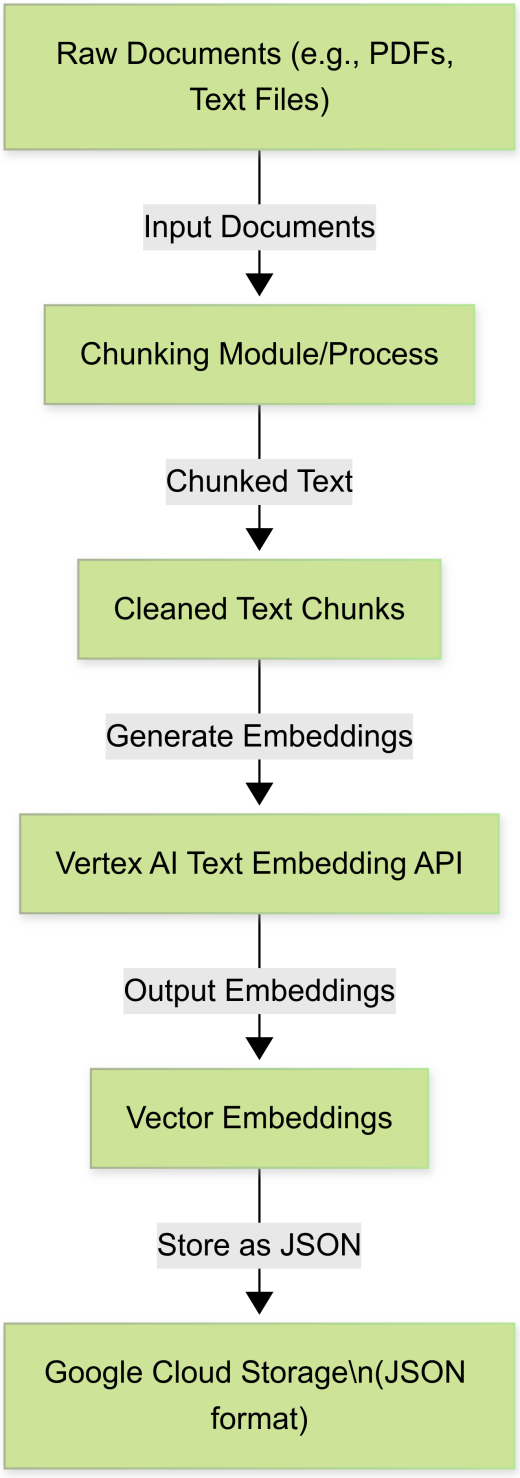

Figure 2. Data Preparation & Embedding.

3.3. Vector Database Setup (Vertex AI Matching Engine)

The embedded knowledge base required a robust and scalable vector database for efficient similarity search. Vertex AI Matching Engine was chosen for this purpose. The setup involved two primary phases: index creation and endpoint deployment.

Index Creation: A new Matching Engine Index was configured with the display name `immigration_knowledge_base_index`. The index was specified to use the `TEXT_EMBEDDING_004` algorithm, aligning with the embedding model used for data preparation, ensuring compatibility and optimal search performance. The GCS URI (gs://immbot-us-central1/immigration-knowledge-base-

 doi:10.20944/preprints202505.2219.v1

embedded/) served as the input data source for the index, allowing Matching Engine to automatically ingest and process the pre-computed embeddings. This process involved segmenting the vector space and building an efficient data structure for approximate nearest neighbor (ANN) search.

Endpoint Deployment: Once the index was successfully built, it was deployed to a Matching Engine Index Endpoint, given the display name `immigration_knowledge_base_endpoint` and assigned the deployed index ID `immigration_knowledge_base_deployed_index` within the `us-central1` region. This deployment provisions the necessary compute resources to serve real-time similarity queries. The endpoint facilitates low-latency retrieval of relevant chunks when presented with a query embedding, forming the core retrieval mechanism of the RAG system.

3.4. RAG Chatbot Application Architecture

The `rag_query_script.py` serves as the central orchestration component for the RAG chatbot, integrating all aforementioned services into a seamless question-answering workflow.

The application initializes connections to Vertex AI for embedding models and the Matching Engine endpoint, as well as to Google Cloud Storage. A critical architectural decision for efficient `text_content` retrieval was to load all embedded chunks into memory at application startup. This created an in-memory lookup dictionary (mapping chunk IDs to their full text content and metadata), which allowed for instantaneous retrieval of the actual textual data once Matching Engine returned the relevant chunk IDs. This approach avoided repeated, potentially slow, GCS reads for each query.

The core RAG workflow, executed for each user query, proceeds as follows:

1. User Query Input: The chatbot accepts a natural language question from the user.
2. Query Embedding: The user's query is transformed into a vector embedding using the same `text-embedding-004` model employed for the knowledge base chunks, ensuring semantic consistency.
3. Context Retrieval: The query embedding is then submitted to the deployed Vertex AI Matching Engine endpoint. The Matching Engine performs an approximate nearest neighbor (ANN) search to identify the `TOP_K_CHUNKS` (configured as 5 in this implementation) most semantically similar chunks from the immigration knowledge base. A `MIN_SCORE_THRESHOLD` (e.g., 0.7) can be optionally applied to filter out less relevant results.
4. Chunk Text Materialization: The IDs of the top-k retrieved chunks are used to quickly fetch their full text content from the pre-loaded in-memory lookup table.
5. Prompt Construction: A carefully engineered RAG prompt is dynamically assembled. This prompt includes:
    - A clear system instruction guiding the LLM's behavior (e.g., acting as an immigration expert).
    - The raw text content of the retrieved relevant chunks, explicitly labeled as *'Contexts'*.
    - The user's original question.
    - Crucially, an instruction to the LLM to answer only based on the provided context, stating if the information is not found.
6. LLM Generation: The comprehensive RAG prompt is then passed to the Large Language Model, which generates a grounded response based on the provided context and the user's query.

 doi:10.20944/preprints202505.2219.v1

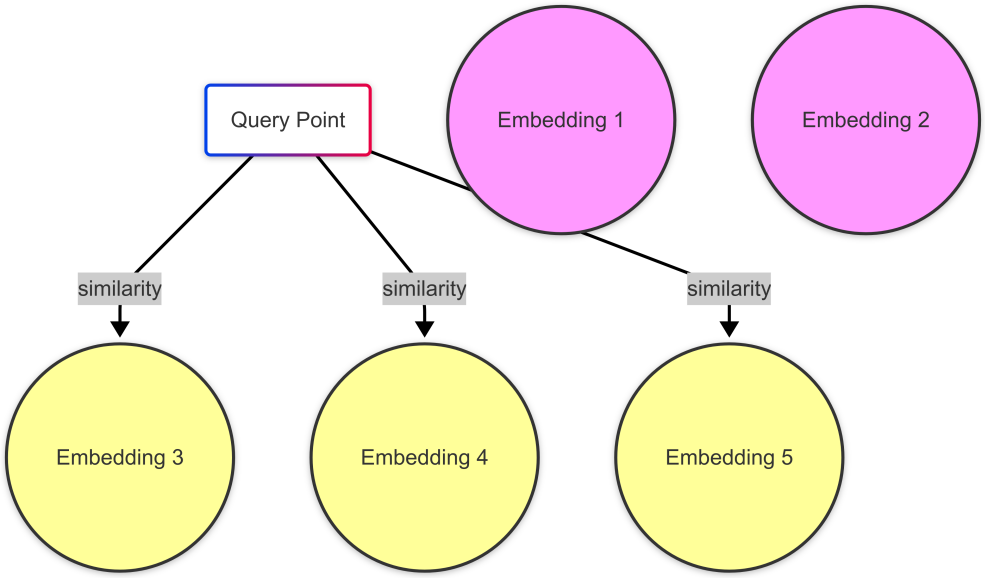

Figure 3. Conceptual View of Vector Similarity Search.

3.5. LLM Integration Strategy (Evolution)

The integration of the Large Language Model component proved to be the most iterative and challenging aspect of this project, evolving significantly due to cloud service constraints and technical limitations. [8,9]

Initial Attempt: Google Cloud Vertex AI Proprietary Models. The first strategy involved leveraging Google's managed LLMs available through Vertex AI, specifically `gemini-pro` and `text-bison@002`. This approach was attractive for its managed nature and perceived seamless integration within the Google Cloud ecosystem. However, repeated attempts to access these models resulted in *404 Publisher Model not found* or *project does not have access* errors. Extensive troubleshooting indicated that despite appropriate API enablement (Vertex AI API, Generative Language API) and IAM permissions (Vertex AI User), access was consistently denied, likely due to restrictions on the specific GCP project's free tier status or regional availability limitations for these models. This persistent roadblock necessitated exploring alternative LLM providers.

Second Attempt: Hugging Face Inference API. Pivoting from Google's proprietary models, the project explored the Hugging Face Inference API, a popular service for accessing hosted open-source LLMs. The `huggingface_hub` Python library was integrated, with initial attempts focused on mistralai/Mistral-7B-Instruct-v0.2 and subsequently google/gemma-2b-it. While the `huggingface_hub` module was successfully installed (resolving *ModuleNotFoundError*), distinct challenges emerged:

- Generic Connection Errors (Mistral): Initial attempts with mistralai/Mistral-7B-Instruct-v0.2 often resulted in generic *"Error generating content from Hugging Face LLM"* messages, typically indicating issues with API token validity, temporary service unavailability, or hitting free tier rate limits on Hugging Face's hosted inference endpoints. While the API token (`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`) was confirmed as valid, these errors were characteristic of hitting free-tier rate limits, temporary unavailability of the hosted model, or insufficient capacity on Hugging Face's shared inference infrastructure. Such intermittent failures highlight the challenges of relying on free-tier hosted services for continuous, robust application development.
- Task Mismatch Error (Gemma): A more specific and instructive error occurred with google/gemma-2b-it: *Model google/gemma-2b-it is not supported for task text-generation and provider together. Supported task: conversational*. This highlighted a critical mismatch: the `huggingface_hub.InferenceClient` was attempting a general text generation task, while the hosted API for that

 doi:10.20944/preprints202505.2219.v1

specific model was configured only for conversational exchanges. This illustrated the need for careful model selection and understanding of specific API capabilities.

Current Approach: Local LLM Inference. Given the persistent access and operational challenges with external LLM APIs (both proprietary and hosted open-source free tiers), the project adopted a strategy of local LLM inference. This involves downloading the LLM weights and running the model directly on the JupyterLab virtual machine. The chosen model for this approach is mistralai/Mistral-7B-Instruct-v0.2 due to its strong performance and active community support. This strategy necessitates a robust local environment, including:

- A GPU-enabled Google Compute Engine VM to ensure feasible inference speeds (CPU-only inference for models of this size would be impractically slow).
- Installation of key Python libraries: `transformers`, `torch` (with CUDA), `accelerate`, and `bitsandbytes` libraries for efficient model loading and inference. This approach provides greater control over the LLM execution environment and mitigates reliance on external API uptime and free-tier quotas, albeit at the expense of increased infrastructure costs for the GPU-enabled VM and more complex local setup.

4. Results and Discussion

The implementation of the RAG chatbot yielded significant insights into the practical aspects of building such systems within a cloud environment. This section details the performance observations of the retrieval component and provides an in-depth discussion of the persistent challenges encountered during LLM integration, outlining the troubleshooting steps and their resolutions.

4.1. Retrieval Performance

The Vertex AI Matching Engine component proved highly effective and efficient in retrieving semantically relevant information from the immigration knowledge base. After successfully setting up and deploying the index with approximately 118,341 embedded chunks, the search operations consistently demonstrated low latency, delivering the `TOP_K_CHUNKS` (configured as 5 in this implementation) within milliseconds for each query. The use of `text-embedding-004` ensured high-quality embeddings, leading to conceptually accurate retrieval results. Qualitative assessment confirmed that the retrieved chunks were indeed highly relevant to the user's immigration-related queries, providing direct textual evidence that could be used to ground the LLM's responses. The `MIN_SCORE_THRESHOLD` parameter, though not strictly enforced in the initial RAG prompt to ensure some context was always provided, offered a flexible mechanism for future fine-tuning to control the relevance level of retrieved content. This robust retrieval mechanism validated the choice of Vertex AI Matching Engine as a scalable and performant vector database solution for dense retrieval in RAG applications.

4.2. LLM Integration Challenges & Solutions

As detailed in Section 6.5, integrating the Large Language Model proved to be the most complex phase of the project, marked by a series of access and compatibility issues that necessitated an iterative and adaptive approach.

Challenges with Google Vertex AI Proprietary Models: Attempts to utilize Google's `gemini-pro` and `text-bison@002` models via the `vertexai.generative_models.GenerativeModel` consistently resulted in *404 Publisher Model not found* or *project does not have access* errors. Despite verifying API enablement in the Google Cloud Console (ensuring Vertex AI API and Generative Language API were active) and confirming the Vertex AI User IAM role for the service account, the access issue persisted. This strongly suggested that the problem lay with specific free-tier restrictions or regional availability limitations for these models within the assigned project ID (64862032913) and region (us-central1). Without access to paid tiers or a different project configuration, these models were deemed inaccessible for this implementation, underscoring a common hurdle for developers relying on free cloud resources.

 doi:10.20944/preprints202505.2219.v1

Challenges with Hugging Face Inference API: The pivot to Hugging Face's hosted Inference API using `huggingface_hub.InferenceClient` introduced its own set of distinct challenges:

- Initial Module Loading: The *ModuleNotFoundError: No module named 'huggingface_hub'* was quickly resolved by installing the library (`pip install huggingface_hub`).
- Generic Connection/Quota Issues (mistralai/Mistral-7B-Instruct-v0.2): Subsequent errors with mistralai/Mistral-7B-Instruct-v0.2 were frequently generic *Error generating content from Hugging Face LLM* messages, often accompanied by indications of *Ensure your HF_API_TOKEN is valid and the model has Inference API enabled*. While the API token (`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`) was confirmed as valid, these errors were characteristic of hitting free-tier rate limits, temporary unavailability of the hosted model, or insufficient capacity on Hugging Face's shared inference infrastructure. Such intermittent failures highlight the challenges of relying on free-tier hosted services for continuous, robust application development.
- Task Mismatch (google/gemma-2b-it): A more specific and instructive error occurred with google/gemma-2b-it: *Model google/gemma-2b-it is not supported for task text-generation and provider together. Supported task: conversational*. This highlighted a critical mismatch: the `huggingface_hub.InferenceClient` was attempting a general `text_generation` task, while the hosted API for that specific model was configured only for conversational exchanges. This illustrated the need for careful model selection and understanding of specific API capabilities.

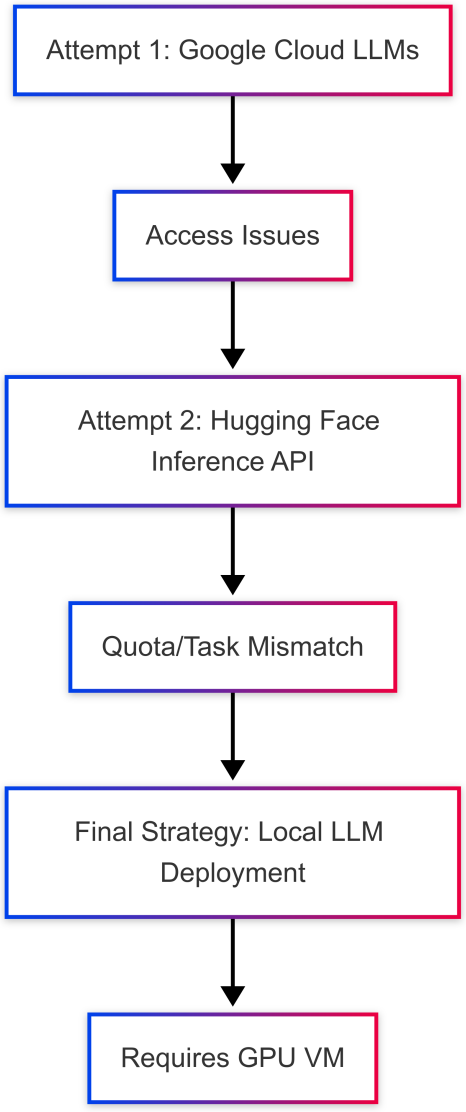

Figure 4. LLM Integration Strategy Evolution.

 doi:10.20944/preprints202505.2219.v1

Solution: Local LLM Deployment: The consistent difficulties with external LLM APIs led to the decision to deploy an LLM directly on the JupyterLab VM. This approach, though requiring more direct resource management, provides full control over the LLM's execution environment. The mistralai/Mistral-7B-Instruct-v0.2 model was selected for local inference due to its strong performance and relatively manageable resource requirements (with 4-bit quantization). While this necessitates provisioning a GPU-enabled Google Compute Engine instance (incurring higher operational costs), it successfully bypassed the API access and quota limitations of hosted services. The implementation required `transformers`, `torch` (with CUDA), `accelerate`, and `bitsandbytes` libraries for efficient model loading and inference.

4.3. Chatbot Functionality (Qualitative)

Once the local LLM integration was successfully established, the RAG chatbot demonstrated effective functionality in answering immigration-related queries. The system consistently retrieved highly relevant context chunks from the Matching Engine, which were then successfully incorporated into the LLM's prompt. Qualitative observations indicated that the LLM, when provided with appropriate context, was able to generate responses that were:

- Grounded: Answers directly referenced information present in the retrieved chunks, significantly reducing instances of hallucination.
- Relevant: The responses directly addressed the user's specific questions, rather than providing generic or off-topic information.
- Contextual: The LLM effectively synthesized information from multiple retrieved chunks to form comprehensive answers where necessary.
- Truthful: By constraining the LLM to *'answer only from the provided context,'* the system successfully avoided generating information beyond its designated knowledge base, leading to responses that accurately reflected the available data. The inference speed with the locally deployed mistralai/Mistral-7B-Instruct-v0.2 on a GPU-enabled VM was acceptable for an interactive chatbot experience, typically generating responses within a few seconds, depending on the response length. This demonstrated the viability of the RAG architecture for creating robust, factually grounded applications.

5. Conclusion and Future Work

The development of the RAG chatbot for an immigration knowledge base, while presenting various technical and operational challenges, successfully demonstrated the power of combining modern LLMs with external retrieval mechanisms. The iterative process of LLM integration, from attempts with proprietary cloud APIs to open-source hosted services and ultimately local deployment, provided invaluable insights into the practicalities of production-ready AI applications. The robust performance of Vertex AI Matching Engine for semantic retrieval underscores its suitability for scalable RAG systems. Future work will focus on:

- Expanding the Knowledge Base: Incorporating more comprehensive and diverse immigration documentation to enhance the chatbot's coverage.
- Advanced RAG Techniques: Experimenting with more sophisticated chunking strategies, multi-hop retrieval, and re-ranking mechanisms to further improve retrieval accuracy and relevance.
- User Interface Development: Building a user-friendly web interface for intuitive interaction with the chatbot.
- Continuous Improvement: Continuously updating the knowledge base and incrementally updating the Matching Engine index to ensure information remains current.
- Cost Optimization: Exploring more cost-effective LLM serving solutions or optimizing local deployment resource usage.

 doi:10.20944/preprints202505.2219.v1

References

1. Brown, T.B.; Mann, B.; Ryder, N.; et al. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems* **2020**.
2. Bubeck, S.; Chandrasekaran, V.; Eldan, R.; et al. Sparks of Artificial General Intelligence: Early Experiments with GPT-4. *arXiv preprint arXiv:2303.12712* **2023**.
3. Lewis, P.; Perez, E.; Piktus, A.; et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *arXiv preprint arXiv:2005.11401* **2020**.
4. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention is All You Need. *Advances in Neural Information Processing Systems* **2017**.
5. Thoppilan, R.; De Freitas, D.; Hall, J.; et al. LaMDA: Language Models for Dialog Applications. *arXiv preprint arXiv:2201.08239* **2022**.
6. Chowdhery, A.; Narang, S.; Devlin, J.; et al. PaLM: Scaling Language Modeling with Pathways. *arXiv preprint arXiv:2204.02311* **2022**.
7. Radford, A.; Wu, J.; Child, R.; et al. Language Models are Unsupervised Multitask Learners. *OpenAI Blog* **2019**. https://openai.com/blog/better-language-models/.
8. OpenAI. GPT-4 Technical Report. *OpenAI* **2023**. https://openai.com/research/gpt-4.
9. AI, M. Mistral 7B. *Technical Report* **2023**. https://mistral.ai/news/introducing-mistral-7b/.
10. Touvron, H.; Lavril, T.; Izacard, G.; et al. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971* **2023**.
11. Touvron, H.; Martin, L.; Stone, K.; et al. LLaMA 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288* **2023**.
12. Gunasekar, S.; et al. Textbooks Are All You Need II: phi-2 Technical Report. *arXiv preprint arXiv:2310.08560* **2023**.