

ESILV Smart Assistant Technical Report

LLM & Generative AI

Summary

1.	Introduction	2
2.	Problem Description	2
3.	System Architecture Overview.....	3
4.	Retrieval-Augmented Generation (RAG).....	3
a)	Initial Hallucination Problem.....	3
b)	Data Curation and Ingestion	4
5.	Model Selection Challenges.....	4
a)	Ollama and Gemini Attempt	4
b)	Transition to TinyLlama	5
6.	Encoding and Character Issues	5
a)	UTF-8 Encoding Problems	5
b)	Solution	5
7.	Vector Database and Embedding Issues	6
8.	Multi-Agent Coordination	6
a)	Orchestration Logic	6
b)	Role-based access	6
9.	User Interface and Streamlit Integration	7
10.	Evaluation and Results	7
11.	Challenges Summary Key challenges encountered:	8
12.	Conclusion & Future work	8

1. Introduction

The rapid progress of Large Language Models (LLMs) has enabled the development of conversational systems capable of understanding user requests, generating natural-language responses, and supporting complex workflows. However, deploying such systems in a real academic context raises practical challenges related to reliability, factual accuracy, latency, and responsible data handling. This project presents the design and implementation of the ESILV Smart Assistant, a multi-agent chatbot dedicated to the ESILV engineering school. The assistant provides document-grounded answers to questions about ESILV programs, admissions, and courses using a Retrieval-Augmented Generation (RAG) pipeline built on internal documentation, while also supporting student registration through structured form interactions. To remain compatible with academic constraints, the system is deployed locally using Streamlit for the interface and Ollama for on-device inference with an open-source model. In addition, the assistant includes role-based access control (admin/student) to restrict access to sensitive registration data and demonstrate responsible handling of personal information.

2. Problem Description

The core problem addressed by this project is the design of an institution-specific conversational assistant that avoids hallucinated responses while remaining flexible and interactive. Generic Large Language Models often generate incorrect, incomplete, or outdated information when queried about specific institutions, as they rely on probabilistic patterns learned during training rather than verified, authoritative sources. This limitation became evident early in the project when the chatbot was asked simple questions such as “What is ESILV?”. Instead of correctly identifying École Supérieure d’Ingénieurs Léonard de Vinci located in La Défense, the model incorrectly described ESILV as an institution based in Valenciennes, which is factually inaccurate. This behavior illustrates a fundamental weakness of standalone LLMs when applied to domain-specific or institutional contexts: without external grounding, the model tends to select the most statistically plausible answer rather than the correct one. Addressing this issue required a shift from pure generation to a system grounded in official ESILV documentation, combined with controlled interaction flows for sensitive use cases such as student registration. In addition, the project imposed academic constraints such as local deployment and limited computational resources, which further influenced architectural and technological choices. As a result, the system needed to ensure grounded responses, controlled behavior for structured interactions, a clear separation of responsibilities through multiple agents, and full local execution to comply with these constraints.

3. System Architecture Overview

The ESILV Smart Assistant is built on a modular multi-agent architecture coordinated by a central orchestration layer. This architectural choice was motivated by the need to separate responsibilities, control system behavior, and ensure long-term maintainability. Instead of relying on a single monolithic agent, the system decomposes its functionality into specialized agents, each designed to handle a specific category of tasks. This approach improves robustness and allows individual components to be modified or extended without affecting the entire system.

At the core of the architecture lies the orchestration agent, which is responsible for analyzing user intent and routing incoming requests to the appropriate agent. Depending on the nature of the user query, the orchestration agent delegates the task to one of several specialized components. The retrieval agent is responsible for answering factual questions related to ESILV by leveraging a Retrieval-Augmented Generation pipeline. It combines vector-based document retrieval with language generation to produce responses grounded in official ESILV documentation. The form agent handles structured interactions, such as student registration and contact collection, by guiding users through predefined data entry steps and ensuring consistent data formatting. In parallel, an admin logic component enforces role-based access control, restricting access to sensitive information such as registration data to authorized users only.

This multi-agent architecture ensures modularity by isolating distinct responsibilities, enhances maintainability by simplifying debugging and updates, and improves extensibility by allowing new agents or functionalities to be integrated with minimal changes to the existing system. This architecture is then well suited for an academic prototype that aims to demonstrate best practices in system design while remaining adaptable to future improvements.

4. Retrieval-Augmented Generation (RAG)

a) Initial Hallucination Problem

In the early stages of the project, the chatbot relied solely on a standalone Large Language Model without any retrieval mechanism. Although the model produced fluent responses, it frequently generated incorrect institutional information. For example, when asked to describe ESILV, the model incorrectly identified it as a school located in Valenciennes instead of École Supérieure d'Ingénieurs Léonard de Vinci in La Défense. This behavior revealed a key limitation of generic LLMs when applied to institution-specific contexts.

Such errors occur because LLMs generate responses based on statistical patterns learned during training rather than verified sources. In the absence of external grounding, the model tends to select the most statistically probable answer, even when it is factually incorrect. This issue is amplified when institution names resemble acronyms or overlap with other entities. These observations demonstrated the necessity of grounding responses in authoritative documents, leading to the adoption of a Retrieval-Augmented Generation approach.

b) Data Curation and Ingestion

To address the hallucination issues observed during initial testing, an ESILV-specific dataset was created and stored in a dedicated text file (data/esilv.txt). This dataset contains curated institutional information, including the official school's name, geographic location in La Défense (Paris), accreditation details, academic programs and majors, and broader institutional context. The objective of this curation process was to provide a reliable and authoritative knowledge base aligned with official ESILV communication.

The document was put into a vector database using ChromaDB, with embeddings generated locally via Ollama. These embeddings enable semantic search over the institutional content and allow the retrieval of the most relevant information in response to user queries. Once the Retrieval-Augmented Generation pipeline was activated, the chatbot consistently produced accurate answers, correctly identifying ESILV as a school located in La Défense rather than Valenciennes. This validated the effectiveness of the data curation and ingestion process in grounding model responses.

5. Model Selection Challenges

a) Ollama and Gemini Attempt

During the initial phase of model selection, larger language models such as Gemini and heavier models available through Ollama were evaluated. While these models offered stronger reasoning capabilities, several practical limitations quickly emerged when deployed on local hardware. Inference times were excessively long, leading to noticeable latency in responses and, in some cases, causing the Streamlit interface to become unresponsive. In addition, these models exhibited high memory and computational resource consumption, as well as unstable behavior during embedding generation.

These limitations significantly degraded the user experience and made the system unsuitable for interactive use or live academic demonstrations. The use of larger models was deemed impractical within the constraints of local deployment and limited computational resources.

b) Transition to TinyLlama

To overcome the limitations observed with larger models, the project transitioned to TinyLlama, a lightweight open-source language model supported by Ollama. This model offered significantly faster inference times and more stable behavior when deployed locally, making it well suited for interactive use and academic demonstrations. In addition, its compatibility with LangChain and ChromaDB ensured seamless integration with the existing Retrieval-Augmented Generation pipeline.

The primary trade-off associated with this choice was a reduced reasoning depth compared to larger models. However, this limitation was mitigated by a stronger reliance on retrieval-based grounding, ensuring that factual correctness was maintained through authoritative ESILV documents. Given the project's focus on accuracy, reproducibility, and local execution rather than creative text generation, this trade-off was considered acceptable and aligned with the overall system objectives.

6. Encoding and Character Issues

a) UTF-8 Encoding Problems

A recurring technical issue involved incorrect rendering of French characters, resulting in outputs such as: Ã‰cole SupÃ©rieure dâ€™IngÃ©nieurs. This problem stemmed from inconsistent file encodings, conflicts with default Windows encodings, and improper decoding during the document ingestion process. These encoding issues negatively impacted both the chatbot responses and the user interface.

b) Solution

The issue was resolved by explicitly saving all data files in UTF-8 format and enforcing UTF-8 encoding during file loading. In addition, consistent encoding settings were applied across Python and Streamlit. Once corrected, French characters rendered correctly across the interface and chatbot responses.

7. Vector Database and Embedding Issues

Another major technical challenge involved embedding dimension mismatches within the vector database. When switching between language models, ChromaDB produced errors such as “Collection expecting embedding with dimension of 4096, got 2048”. This issue occurred because the vector database had initially been created using one embedding model, while a different embedding model was later used during query time.

Resolving this problem required deleting the existing vector database and re-ingesting all documents using a single, consistent embedding model. In addition, embedding configurations were aligned across both the ingestion and retrieval stages. This ensured dimensional consistency and restored correct operation of the Retrieval-Augmented Generation pipeline.

8. Multi-Agent Coordination

a) Orchestration Logic

Initially, once a user entered the registration flow, the chatbot was no longer able to answer general questions. This limitation was caused by missing orchestration logic, which resulted in the system remaining locked in form mode. Therefore, the chatbot could not dynamically switch between conversational and structured interactions.

This issue was resolved by introducing a dedicated orchestration agent responsible for intent detection and agent routing. The orchestration agent enables dynamic transitions between normal chat and registration flows, allowing users to return to standard question-answer interactions after completing or interrupting the registration process. This improvement significantly enhanced system flexibility and user experience.

b) Role-based access

An admin and student login system was implemented to restrict access to sensitive registration data. This mechanism ensures that only authorized users can access and visualize collected information, thereby demonstrating responsible handling of personal data. For demonstration purposes, authentication credentials were hardcoded, which was sufficient to illustrate role-based access control and administrative functionalities within an academic context.

9. User Interface and Streamlit Integration

Streamlit was selected as the framework for the user interface due to its suitability for rapid prototyping, interactive design, and ease of deployment. It enabled the development of a simple yet functional interface supporting chat-based interactions, structured registration forms, and admin-only dashboards for data visualization.

Despite its simplicity, Streamlit introduced challenges related to session state management, particularly when handling multiple interaction flows such as chat and registration. These issues were addressed using controlled state variables, ensuring consistent system behavior and a smooth user experience across different user roles.

10. Evaluation and Results

The final system successfully meets the objectives defined at the beginning of the project. It provides accurate and document-grounded answers to questions related to ESILV, effectively avoiding hallucinated responses with a Retrieval-Augmented Generation pipeline. The assistant also supports structured registration flows while enforcing role-based access control, ensuring that sensitive information is handled appropriately.

In terms of performance, the system runs entirely locally with acceptable response times, making it suitable for interactive use and live academic demonstrations. Overall, the ESILV Smart Assistant fulfills all functional and technical requirements of the project and demonstrates the feasibility of deploying a reliable, institution-specific conversational assistant within academic constraints.

11. Challenges Summary Key challenges encountered:

The main challenges encountered during the development of the ESILV Smart Assistant include:

- Hallucinated responses generated by standalone Large Language Models when answering institution-specific questions
- Performance limitations of larger models when deployed on local hardware
- Character encoding issues affecting the correct rendering of French accents
- Vector database inconsistencies caused by embedding dimension mismatches
- Orchestration logic issues preventing smooth transitions between interaction flows
- Constraints related to full local deployment and limited computational resources

Each of these challenges contributed to a deeper understanding of the practical considerations involved in designing and deploying real-world LLM-based systems.

12. Conclusion & Future work

This project demonstrates that building a reliable Large Language Model-based assistant requires far more than simply calling an API. Through careful system architecture design, retrieval grounding, multi-agent coordination, and extensive debugging, the ESILV Smart Assistant successfully meets its objectives. The system provides accurate, institution-specific answers, avoids hallucinations through Retrieval-Augmented Generation, and supports structured interactions with appropriate access control, all while running entirely on local infrastructure.

Beyond its functional success, the project highlights the practical challenges involved in deploying Generative AI systems in real-world and academic contexts, including model selection constraints, data consistency issues, and orchestration complexity. These challenges provided valuable insights into responsible and reproducible AI system design.

Future work could focus on extending the system with a persistent and secure backend for data storage, improving authentication mechanisms, refining intent classification, and exploring cloud-based deployment to enhance scalability. Additional improvements such as multilingual support and real-time document updates could further increase the assistant's usefulness and adaptability.