ESILV Smart Assistant Technical Report – LLM & Generative AI

1. Introduction

The rapid development of Large Language Models (LLMs) has enabled the creation of intelligent conversational systems capable of reasoning, retrieving information, and interacting with users in a natural way. However, deploying such systems in a real-world academic context introduces significant technical challenges related to accuracy, performance, data grounding, and system orchestration. This project presents the design and implementation of the ESILV Smart Assistant, an intelligent chatbot dedicated to the ESILV engineering school. The objective is to provide a reliable assistant capable of answering questions about ESILV programs, admissions, and courses, while also handling structured interactions such as student registration. The system relies on Retrieval-Augmented Generation (RAG) and a multi-agent architecture, and is deployed locally using Ollama and open-source models. Throughout the project, several non-trivial challenges were encountered, including model selection constraints, encoding issues, hallucinated institutional knowledge, vector database inconsistencies, and orchestration logic conflicts. This report documents not only the final system but also the reasoning, debugging process, and technical decisions behind each solution.

2. Problem Description

The core problem addressed by this project is the creation of an institution-specific chatbot that avoids hallucinations while remaining flexible and interactive. Generic LLMs often provide incorrect or outdated information when asked about specific institutions. This issue was observed early in the project when asking the chatbot simple questions such as “What is ESILV?”. Instead of referencing École Supérieure d’Ingénieurs Léonard de Vinci in La Défense, the model incorrectly described ESILV as an institution located in Valenciennes, which is factually incorrect. This behavior highlighted a fundamental limitation of standalone LLMs: they rely on probabilistic training data rather than authoritative sources. The project therefore required: Grounded answers based on official ESILV data Controlled behavior for sensitive interactions (registration) A clear separation of responsibilities via agents Local deployment due to academic constraints

3. System Architecture Overview

The ESILV Smart Assistant uses a multi-agent architecture coordinated by an orchestration layer. Each agent has a clearly defined responsibility: Retrieval Agent: Answers factual questions using RAG Form Agent: Collects structured user information Orchestration Agent: Routes queries to the appropriate agent Admin Logic: Controls access to sensitive data This architecture ensures modularity, maintainability, and extensibility.

4. Retrieval-Augmented Generation (RAG)

4.1 Initial Hallucination Problem

Initially, the chatbot relied solely on an LLM without retrieval. This resulted in incorrect institutional descriptions, such as identifying ESILV as a school in Valenciennes. This issue occurred because: The LLM had incomplete or ambiguous training data ESILV shares an acronym-like structure with other institutions The model defaulted to the most statistically probable answer This behavior demonstrated the necessity of grounding responses in authoritative documents.

4.2 Data Curation and Ingestion

To solve this, an ESILV-specific dataset was created (data/esilv.txt) containing: Official school name Location (La Défense, Paris) Accreditation details Programs and majors Institutional context This document was ingested into ChromaDB, using embeddings generated via Ollama. Once the RAG pipeline was active, the chatbot began answering correctly, referencing the ESILV school in La Défense instead of Valenciennes.

5. Model Selection Challenges

5.1 Ollama and Gemini Attempt

Initially, larger models such as Gemini and heavier Ollama models were considered. However, several practical limitations emerged: Extremely slow inference times on local hardware Long response latency causing Streamlit to freeze Excessive resource usage Unstable behavior during embedding generation These constraints made the system impractical for live demonstrations.

5.2 Transition to TinyLlama

To address these issues, the project transitioned to TinyLlama, a lightweight open-source model supported by Ollama. Advantages: Significantly faster inference Stable local deployment Suitable for academic demonstration Compatible with LangChain and ChromaDB Trade-offs: Lower reasoning depth Requires stronger reliance on RAG for correctness This trade-off was acceptable given the project’s emphasis on retrieval accuracy rather than creative generation.

6. Encoding and Character Issues

6.1 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 Windows default encoding conflicts Improper decoding during ingestion

6.2 Solution

The issue was resolved by: Explicitly saving data files in UTF-8 Forcing UTF-8 encoding during file loading Ensuring Streamlit and Python used consistent encodings Once corrected, French characters rendered correctly across the UI and responses.

7. Vector Database and Embedding Issues

Another major challenge involved embedding dimension mismatches. When changing models, ChromaDB produced errors such as: Collection expecting embedding with dimension of 4096, got 2048 This occurred because: The vector database was created with one embedding model A different embedding model was later used for querying The fix required: Deleting the existing vector database Re-ingesting all documents using the same model Ensuring consistent embedding configuration across ingestion and retrieval

8. Multi-Agent Coordination

8.1 Orchestration Logic

Initially, once a user entered the registration flow, the chatbot could no longer answer questions. This was due to: Missing orchestration logic The system remaining locked in form mode This issue was solved by: Introducing an orchestration agent Allowing dynamic intent detection Enabling users to return to normal chat after registration

8.2 Role-Based Access Control

An admin/student login system was added to: Restrict access to registration data Demonstrate ethical handling of personal information Support administrative visualization Hardcoded credentials were used for demonstration purposes.

9. User Interface and Streamlit Integration

Streamlit was chosen for: Rapid prototyping Interactive UI Ease of deployment The interface supports: Chat-based interaction Registration forms Admin-only dashboards Despite its simplicity, Streamlit introduced challenges related to session state management, which were addressed using controlled state variables.

10. Evaluation and Results

The final system successfully: Provides accurate, grounded answers about ESILV Avoids hallucinations through RAG Handles structured registration flows Enforces role-based access Runs fully locally with acceptable performance The system is suitable for live demonstration and meets all academic project requirements.

11. Challenges Summary Key challenges encountered:

LLM hallucinations

Model performance limitations

Encoding issues

Vector database inconsistencies

Orchestration logic bugs

Local deployment constraints

Each challenge contributed to a deeper understanding of real-world LLM system design.

12. Future Work Potential improvements

include: Persistent database for registrations

Secure authentication system

Improved intent classification

Deployment on cloud infrastructure

Multilingual support Real-time document updates

13. Conclusion

This project demonstrates that building a reliable LLM-based assistant requires far more than simply calling an API. Through careful architecture design, retrieval grounding, agent coordination, and extensive debugging, the ESILV Smart Assistant achieves its objectives while highlighting the practical realities of deploying Generative AI systems.