**Code Summarization and Automatic Code Reviews with LLMs**

### BITS S1-24\_DSECLZG628T: Dissertation

**Broad Academic Area of Work**

**LLM’s, RAG**

by

Shalabh Raj

2022DC04190

# Dissertation work carried out at

## Verizon Data Services India pvt ltd, Hyderabad

Submitted in partial fulfillment of MTech Data Science & Engineering

Under the Supervision of

Venkata Naga Sasidhar Ghantasala, Assos Fellow-Software Devt

Verizon Data Services India pvt ltd, Hyderabad

****

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE**

**PILANI (RAJASTHAN)**

February2025

#### **CERTIFICATE**

This is to certify that the Dissertation entitled “Code Summarization and Automatic Code Reviews with LLMs” and submitted by Shalabh raj having ID-No. 2022DC04190 for the partial fulfillment of the requirements of the MTech Data Science & Engineering degree of BITS, embodies the Bonafide work done by him/her under my supervision.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature of the Supervisor

Place: Hyderabad

Date: 28th February 2025 Venkata Naga Sasidhar, Assos Fellow & HYD

Name, Designation & Organization &Location

**Birla Institute of Technology & Science, Pilani**

**Work-Integrated Learning Programmes Division**

**BITS ZG628T: Dissertation**

**ABSTRACT**

**BITS ID No. : 2022DC04190**

**NAME OF THE STUDENT : Shalabh Raj**

**EMAIL ADDRESS : shalabh.raj@verizon.com**

**STUDENT’S EMPLOYING : Verizon Data Services India pvt ltd, Hyderabad**

**ORGANIZATION & LOCATION**

**SUPERVISOR’S NAME : Venkata Naga Sasidhar Ghantasala**

**SUPERVISOR’S EMPLOYING : Verizon Data Services India pvt ltd, Hyderabad**

**ORGANIZATION & LOCATION**

**SUPERVISOR’S EMAIL ADDRESS: sasidhar.g.venkata.naga@verizon.com**

**DISSERTATION TITLE : Code Summarization and Automatic Code Reviews with LLMs**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ABSTRACT**

In the era of Generative AI, automation is transforming various aspects of human work, including software development. Agile software development demands rapid feature delivery, often leading to increased pressure on developers, which in turn affects code quality. To address this challenge, organizations are increasingly integrating Large Language Models (LLMs) into the software development lifecycle. This dissertation explores the use of LLMs for code summarization and automatic code reviews, aiming to enhance developer productivity and maintain software quality.

A significant portion of software development involves ongoing maintenance, where new features are continuously added to existing products. However, retaining Subject Matter Expert (SME) knowledge within a development team is difficult due to workforce volatility and team reshuffling. Traditional documentation approaches are costly and impractical for dynamic development environments. Additionally, onboarding new developers requires extensive time to understand legacy code, and manual code reviews demand skilled reviewers, adding to the overhead. This research investigates how LLMs can automate code summarization and reviews, enabling faster onboarding, efficient knowledge retention, and proactive issue detection, thereby addressing these challenges in software development.

**Broad Academic Area of Work:** LLMs, RAG, Lang Chain, Flowise, Hugging Face.

**List of Keywords & Abbreviations Used**:

|  |  |
| --- | --- |
| **Keywords** | **Description** |
| LLMs | Large Language Models |
| HLD  LLD | High-Level Design  Low-Level Design |
| API | Application Programming Interface |
| UI  KB  RAG | User Interface  Knowledge Base  Retrieval-Augmented Generation |

**ACKNOWLEDGMENTS**

I would like to thank my supervisor Venkata Naga Sasidhar Ghantasala for all the guidance and support in accomplishing this project. He is given a lot of feedback on how to proceed with developing this project. His feedback gave me thoughts on the scope of work design and the idea of what is missing in the current workplace.

I would like to thank my additional examiner and mentor Raghavendra G S, who is guiding me with a lot of thought-provoking questions to put me on the right line of thinking.

Lastly, I would like to express my deepest gratitude to all the faculty members at BITS, whose guidance and support have been instrumental in the completion of this dissertation. In particular, I extend my sincere appreciation to Prof. Parthasarathy P D, Prof. Srinath R Naidu, Prof. Y V K Ravi Kumar, Prof. Swarna Chaudhary, and Prof. Raja Vadhana P for introducing me to the core concepts of Data Science, Deep Learning, Machine Learning, and Natural Language Processing. The knowledge and insights gained from their teachings played a critical role in shaping my understanding and were essential in the successful development of this project.

**TABLE OF CONTENTS**

[List of Figures and Tables 1](#_Toc1724625)

[Introduction 2](#_Toc1724626)

[Solution Architecture and Design 4](#_Toc1724627)

[Future](#_Toc1724627) Release………………………………………………………………………………………………………………. 11

[References 1](#_Toc1724628)2

List of Figures and Tables

List of Figures

|  |  |
| --- | --- |
| Figure 1.1 HLD Architecture Diagram | Page Number 4 |
| Figure 1.2 LLD Architecture Diagram | Page Number 5 |
| Figure 2.1 VeriCode: UI Interface | Page Number 8 |
| Figure 2.2 VeriCode: Comparison Interface | Page Number 8 |
| Figure 2.3 VeriCode: Code Review | Page Number 9 |
| Figure 3.1 Flowise: Orchestration | Page Number 10 |

List of Table

|  |  |
| --- | --- |
| Table 1 LLMs Models Comparison | Page number 7 |

Introduction

In the current era of Generative AI, most of the human tasks are automatically taken care of by intelligent AI assistants. The software development life cycle is also one of the areas where software development organizations invest heavily to bring generative AI into the development life cycle. In agile software development, the turnaround time for delivering a software feature is much less, and speed-to-market matters a lot. At the same time, the more pressure put on developers to produce faster will result in quality issues. This is the most intriguing problem for many software companies for years. Thanks to GenAI, there is light at the end of the tunnel, to help enterprises to deal with this problem.

The majority of software will undergo maintenance i.e. enterprises go on adding new features on top of existing software products. Developing a product from scratch is not as common as adding features on a weekly basis to an existing product. It could enable new ordering abilities, allowing new qualification rules for products/services offered by enterprises to their customers, AI-driven recommendations and dynamic offers, etc. Since the maintenance is an ongoing effort, it is very important to keep the SME knowledge in the development team. With volatile nature of people sticking to one company and the continuous shuffle of developers working on products will pose a challenge to keeping a manual knowledge base. At the same time, if the organization wants to document ongoing development, its extra cost and continuous documentation are not possible.

For the new people joining the development group, it is a difficult and time-consuming task to go through all the existing code and understand it, before enhancing it. Also to have a second eye of review for the functional/technical issues of the new code implemented, it requires additional persons who are technically and functionally strong in the area of that software application. These two organizational challenges pose a problem for GenAI to solve.

The Problem Statement

For Optimizing the productivity of the employees in the organization and creating a knowledge base while researching How Code Summarization and Automatic Code Reviews with LLMs enabled applications needed to cover below aspects:

* Collect and understand the different technologies used within the organization and corresponding code repositories.
* Convert these private codes into vector embeddings and store them in Vector Database thereby creating a knowledge base repository.
* Need to evaluate the available open and closed LLMs, their Architecture, and compute required. Need to identify which model is generalizing well.

Solution Approach

For the above problem statement, the solution will be of a 4-step approach as below:

1. A software program will need to be developed, whose objective is to download codes from the private code repositories and tokenization them using the Embedding Models and save these vectors in the Vector Database.
2. Another software program is to be developed which will act as a middleware.
3. The above program will be aware of the user context, connect with Vector DB or any configuration DB, and use appropriate LLM models to return the results.
4. A chat portal needs to be developed with a UI to view, select, show searched results, etc.

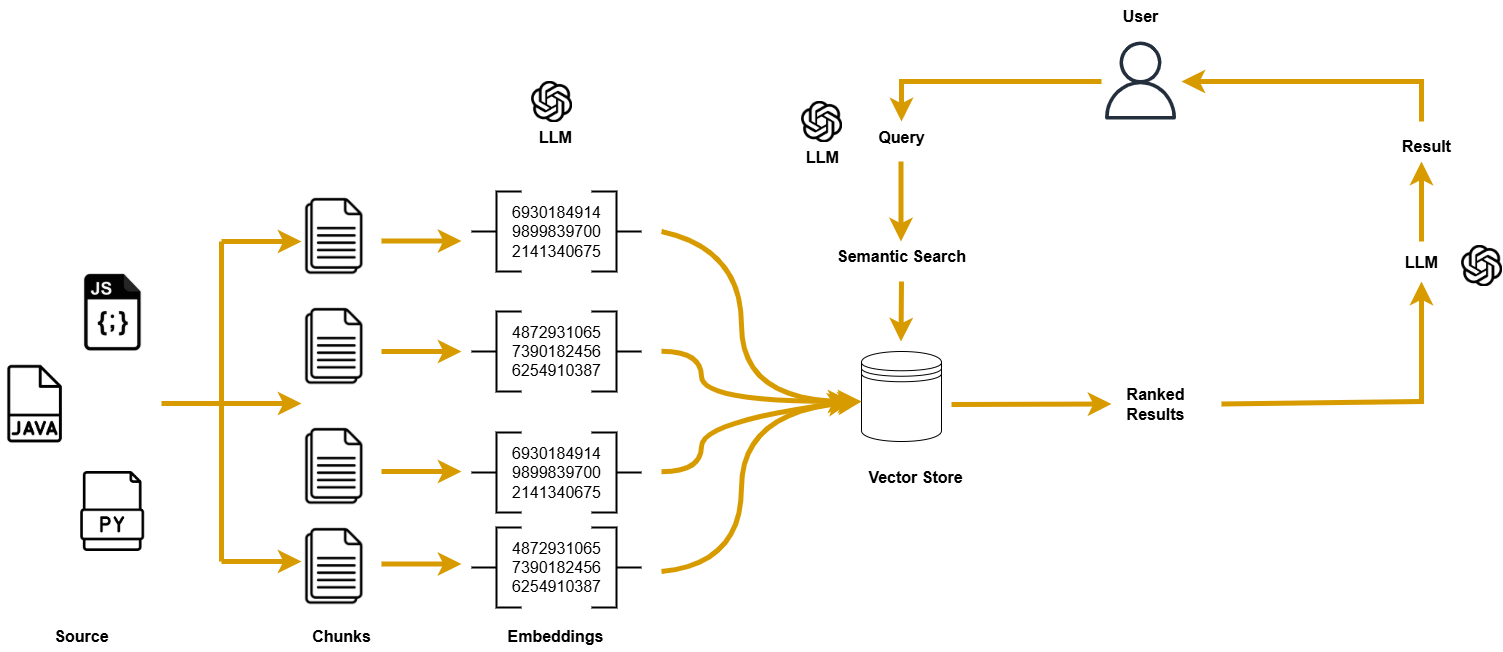
The software is to be built as various pluggable components like UI components, REST APIs, and script programs as appropriate comprising the full solution.

Solution Architecture and Design

As discussed in the solution approach, the overall solution has various components connected as mentioned in the below diagram.

Pre-trained LLM models like GitHub Copilot are built using publicly available data and cannot access or answer questions about private or company-specific codes since they were not trained on them. Organizations often need AI to provide insights from their private knowledge repositories instead of relying solely on public information. To address this, the solution involves RAG-based design to understand and respond based on private source codes.

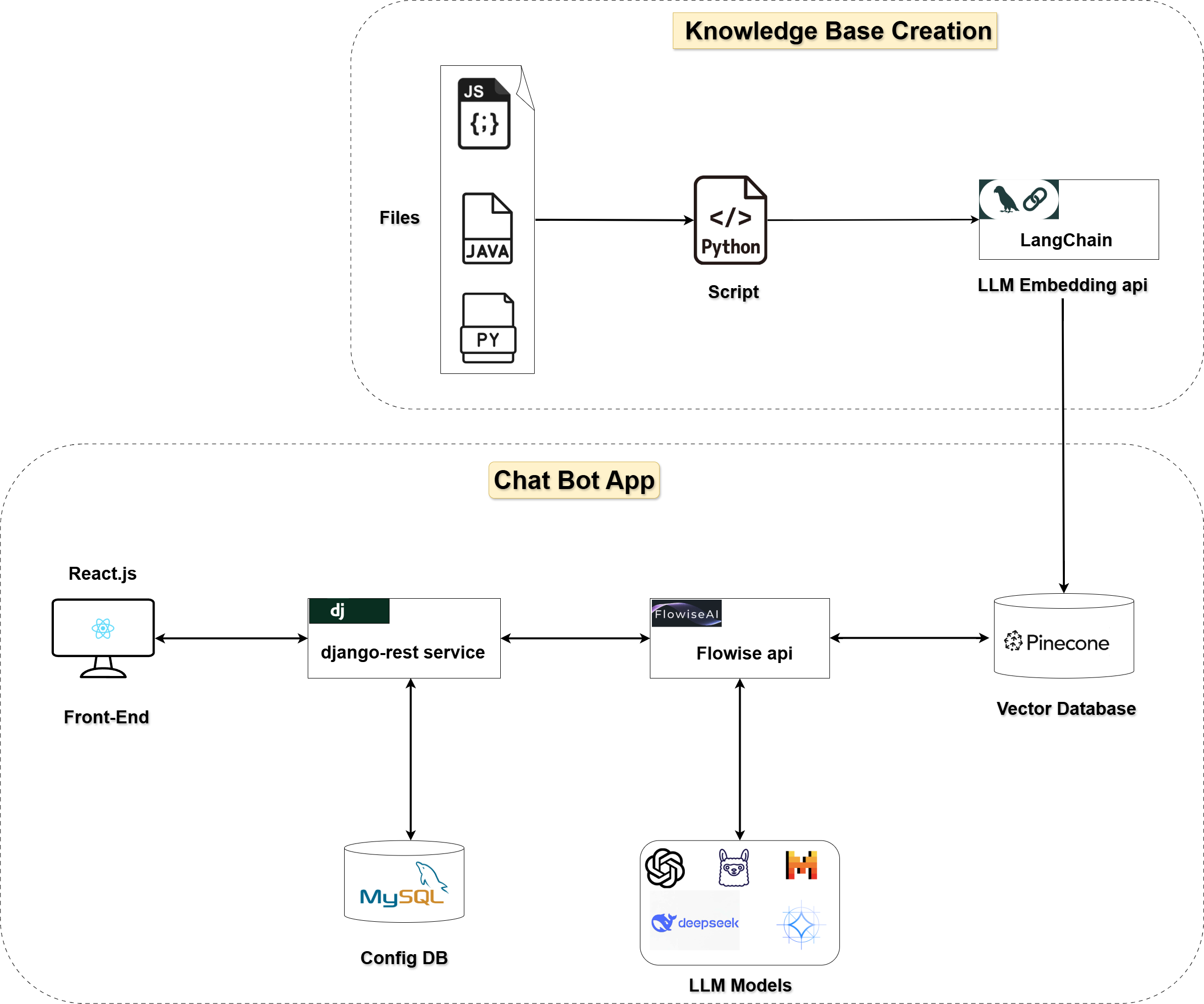
Figure 1.1 HLD Architecture Diagram Design for the Chatbot Framework



The architecture for this solution begins with building a knowledge repository from private code repository data, such as code repositories like GitHub, GitLab etc. These codes are broken into smaller chunks for easier processing and are then converted into embeddings, numerical representations of text, using LLM models API’s. These embeddings are stored in a specialized database called a vector store, enabling efficient retrieval later. This process forms the ingestion pipeline for the model's training data.

When a user submits a question, it is also converted into an embedding using the same API. A semantic search is then performed on the vector store to find the closest matching embeddings to the question. The retrieved embeddings are ranked to identify the best match, and the top result is passed to LLM API to generate a human-readable response. This final answer is presented to the user, providing precise and relevant insights based on the organization's private code.

Figure 1.2 LLD Design Flow



The Framework

The entire solution is designed as a framework model, which will enable the customizations as per the organization needs. The individual components of the framework are designed to handle such customizations. Phase 1 of the solution is to build the Most Viable Product (MVP) to evaluate the overall concept and enhance or customize it in later phases.

The key components of the framework as shown in the above architecture diagram are:

1. Knowledge Base Creation layer
2. Chat Bot Application layer

This design ensures an efficient chatbot system with knowledge retrieval, LLM-based responses, and an interactive front end.

**1) Knowledge Base Creation**

* **Input Files**: Supports JavaScript (.js), Java (.java), and Python (.py) files and can be extended further based on organization requirements.
* **Processing**: Python script to process these files which uses LangChain framework and an embedding API to generate the vector embeddings.
* **Storage**: The generated embeddings are stored in **Pinecone** (a vector database)

**2)** **Chatbot Application**

**Front-End**

* Technology: Built using **React.js**.
* User Interaction: Created a UI/UX **Figma** design for user experience and Provided a UI for users to interact with the chatbot.
* Figma Layout URL: <https://www.figma.com/design/FfhDNe03elOlaOfwtmHj4e/ChatBot?node-id=30-6&p=f&t=LRDJ1bGcGg1IkaG7-0>

**Backend & Services**

* **Django** REST Framework (dj):
  + Acts as a middleware between the frontend and backend services.
  + Fetches configuration data from **MySQL** (Config DB).
* **Flowise** API:
  + Handles **orchestration** of LLM queries.
  + Integrates with multiple LLM models (e.g., OpenAI, DeepSeek, and others).
  + Communicates with Pinecone for retrieving vector-based knowledge.
  + Enhances chatbot responses using relevant knowledge from stored embeddings

**Storage & Retrieval**

* Technology: **Pinecone** Vector Database.
* Stores and retrieves embeddings for similarity searches.

Table 1: LLMs Used in the Project: General, Code, and Embedding Models Comparison

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **LLM Base Models** | | | | |
| Model | Parameters | Architecture | Context Length | Embedding Length |
| llama3 | 8 billion | Llama | 131K tokens | 4 K tokens |
| gemma2 | 9.2 billion | gemma2 | 8 K tokens | 3.5 K tokens |
| mistral | 7.2 billion | llama | 32 K tokens | 4 K tokens |
| deepseek-r1 | 8 billion | Llama | 131K tokens | 4 K tokens |
| falcon3 | 10 billion | Llama | 32 K tokens | 3 K tokens |
|  |  |  |  |  |
| **Finetuned Code Models** | | | | |
| codellama | 13 billion | Llama | 16 K tokens | 5 K tokens |
| codegemma | 8.5 billion | Gemma | 8 K tokens | 3 K tokens |
| starcoder2 | 16 billion | starcoder2 | 16 K tokens | 6 K tokens |
| qwen2.5 | 14.8 billion | qwen2 | 32 K tokens | 5 K tokens |
| codegeex4 | 9.4 billion | chatglm | 131K tokens | 4 K tokens |
| codeqwen | 7.3 billion | qwen2 | 65 K tokens | 4 K tokens |
| codeup | 13 billion | llama | 4 K tokens | 5 K tokens |
|  |  |  |  |  |
| **Embedding Models** | | | | |
| nomic-embed-text | 136.73M | nomic-bert | 2 K tokens | 768 tokens |
| mxbai-embed-large | 334.09B | bert | 512 | 1024 tokens |
| bge-m3 | 566.70B | bert | 8 K tokens | 1024 tokens |
|  |  |  |  |  |

The table categorizes models into LLM Base Models, Finetuned Code Models, and Embedding Models, with variations in parameter size, architecture, context length, and embedding length.

Key features of using these models:

* Open source
* Required less compute
* Generalized well to response

The lightweight **open-sourced models** requiring less computation, Lama3 (8B), Gemma2(9.2B) Mistral (7.2B), LLaMA3 (8B), and DeepSeek-r1 (8B) offer a balance of efficiency and capability for general NLP code summarization tasks.

In the **finetuned** models, Code Llama (13B), Code Gemma (8.5B), qwen2.5(14.8B), codegeex4(9.4B), and codeqwen(7.3B) stand out as efficient options for code generation with manageable resource needs.

For **embedding** tasks, Nomic-Embed-Text (136.73M) is the lightest and most efficient model, making it ideal for low-latency applications.

The below diagram explains the application screens.

Figure 2.1 VeriCode: UI Interface

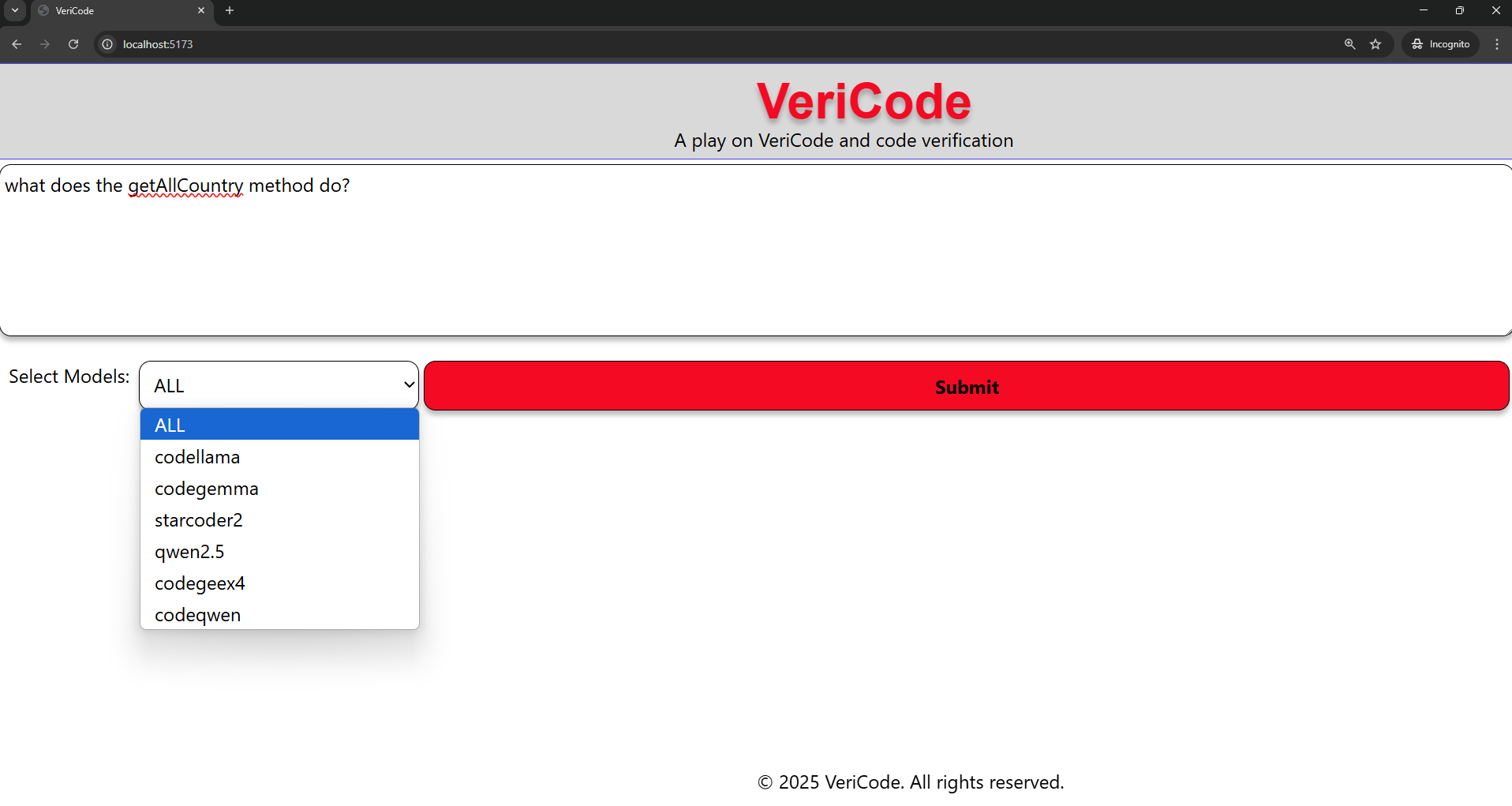
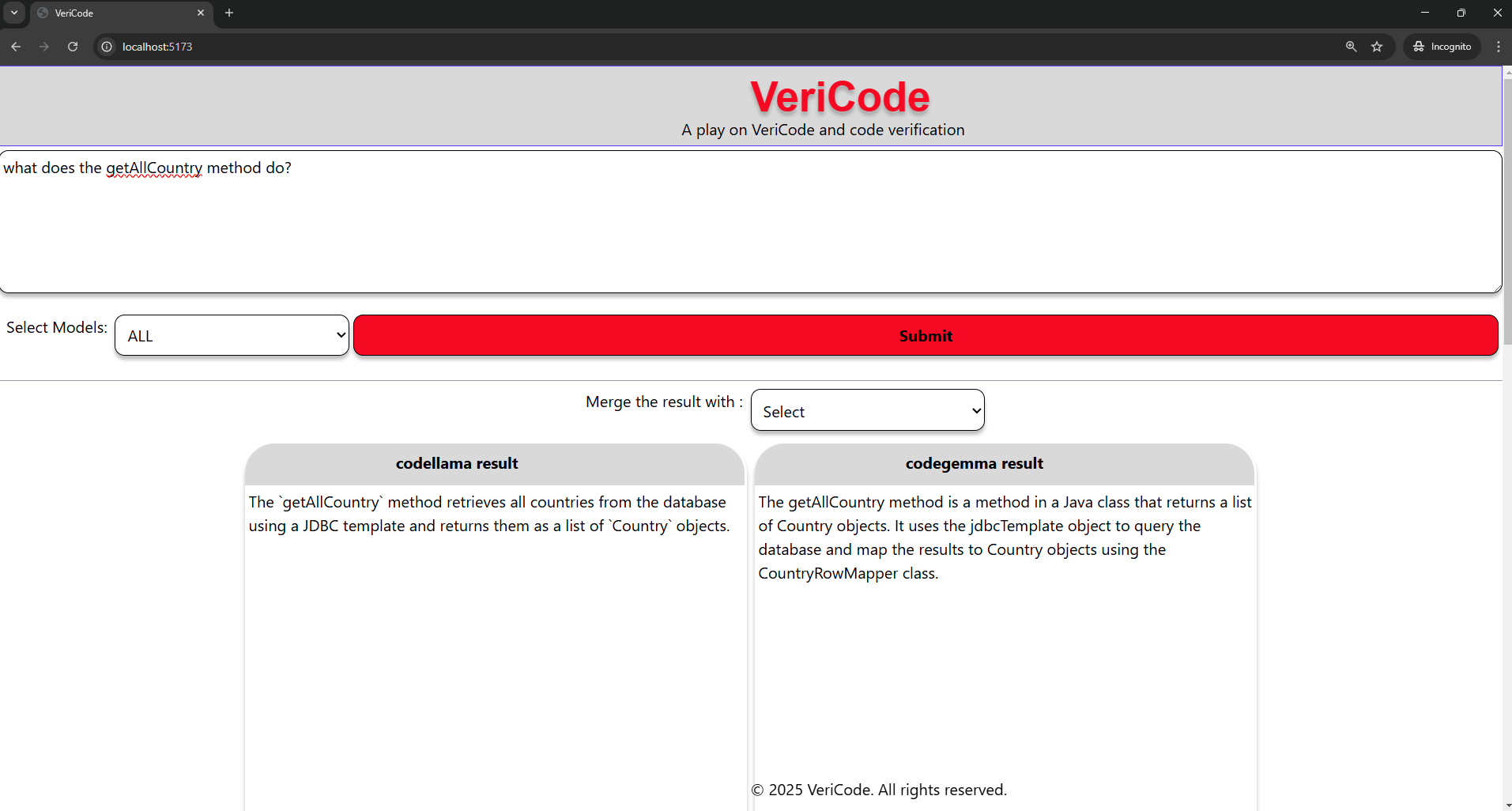


Figure 2.2 VeriCode: Multi-Model Code Explanation and Comparison Interface

****

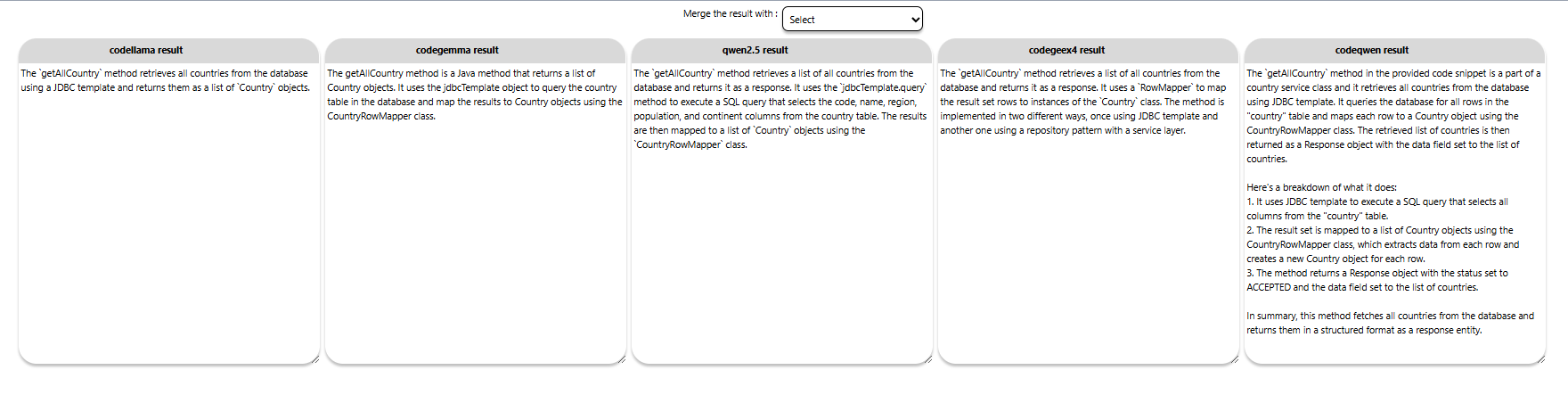
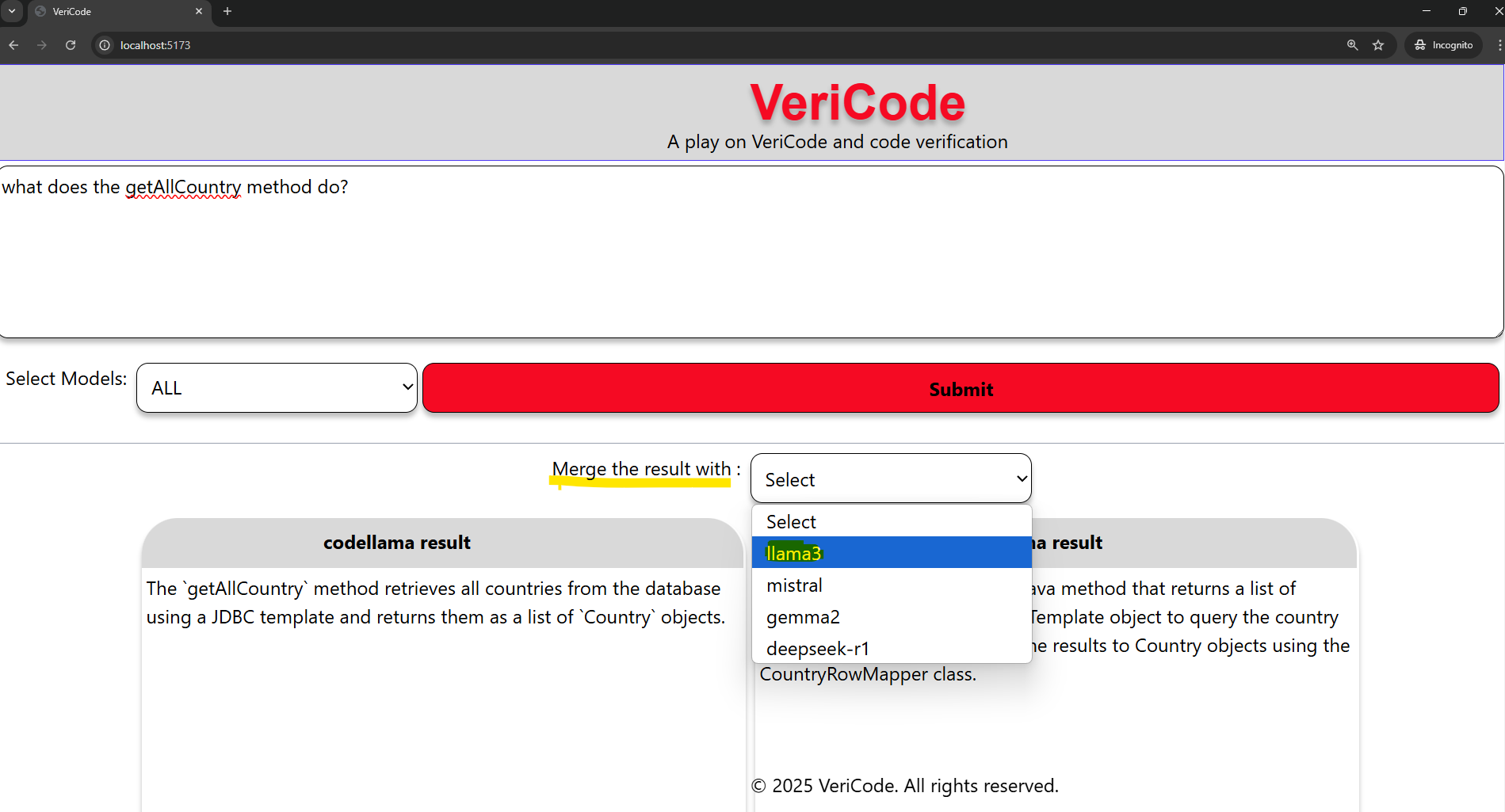
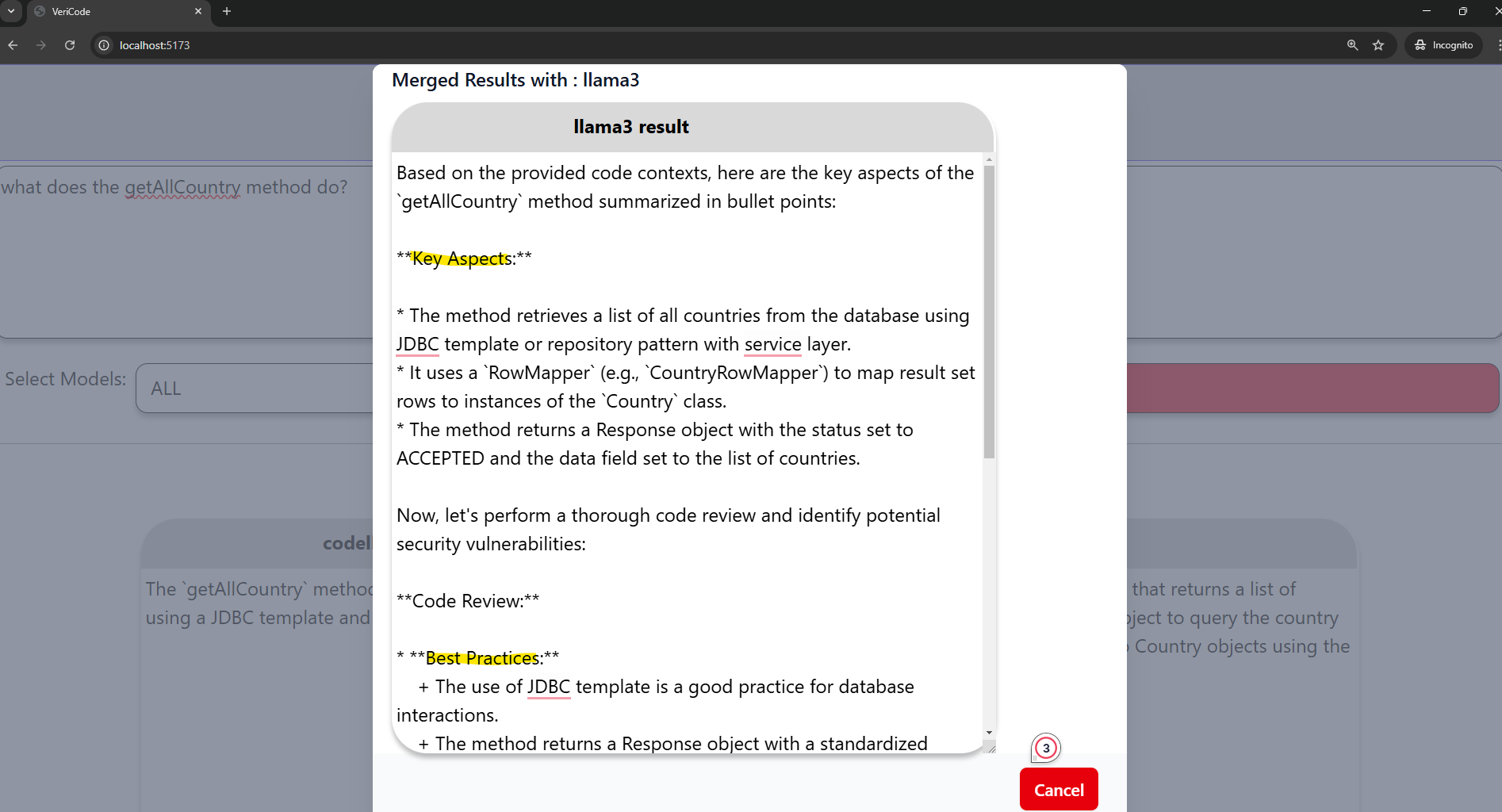


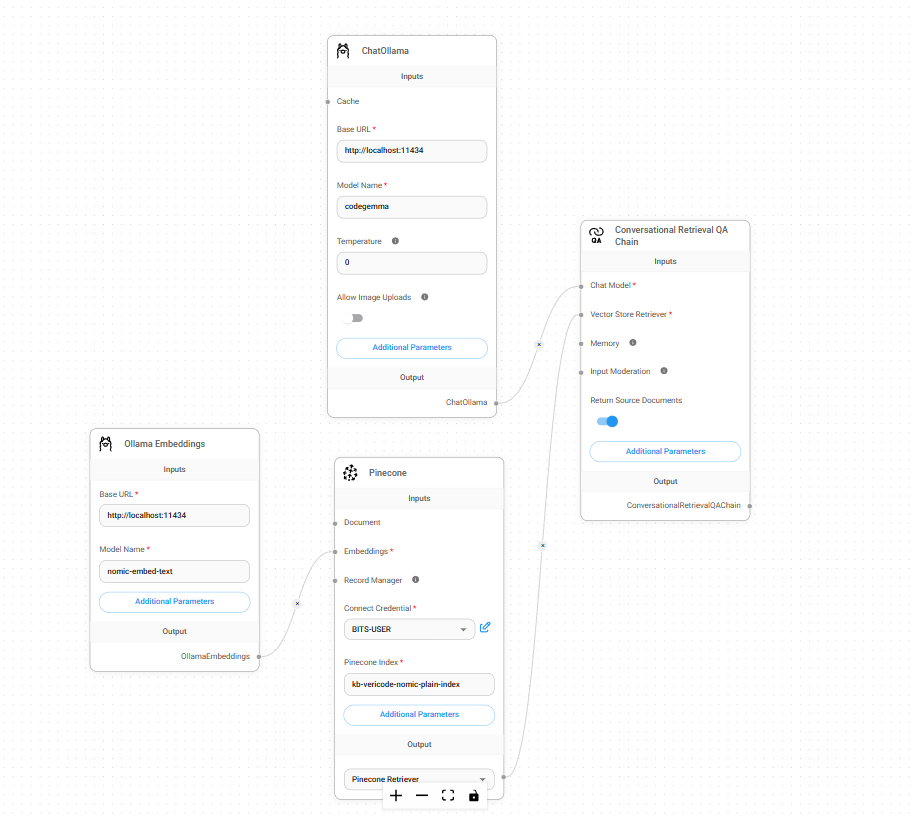
Figure 2.3 VeriCode: Merging Multi-Model Response and Code Review





The below diagram explains the Flowise screens.

Figure 3.1 Flowise: Orchestration of LLM components



**Conclusion**:

As depicted in the above screen, this tool empowers end-users by providing functionalities for both code summarization and code review. By utilizing these capabilities, developers can quickly gain insights into complex code, identify potential issues, and ensure adherence to best practices. This streamlined approach not only enhances code quality but also significantly reduces the time required for manual review processes. As a result, developers can work more efficiently, maintain consistency in their codebase, and ultimately improve overall productivity in software development.

Future Release/Scope

**Beta**

Beta 1.1 Beta release: [**Current State**]

collect feedback from actual end-users.

Beta 1.2 a) Create a supervised data set and check which model generalizes well.

b) Remove the dependency with Flowise, and use the lang-chain framework.

Beta 1.3 Qualified Models, need to be fine-tuned with KB to create custom models tailored for the organization's use case.

Beta 1.4 Create an Agentic Model around these models.

**Release**

Release 1.1 Develop a vs-code and IntelliJ plugin around the Agentic Model

Release 1.2 fine-tuning and any other performance improvement.

References

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30. <https://arxiv.org/abs/1706.03762>

OpenAI — OpenAI Platform documentation. <https://platform.openai.com/docs/overview>

Gemini AI — Gemini API documentation. <https://ai.google.dev/gemini-api/docs>

Hugging Face documentation. <https://huggingface.co/docs>

Lang Chain documentation. <https://python.langchain.com/docs/introduction/>

Ollama documentation. <https://github.com/ollama/ollama/blob/main/docs/api.md>

Flowise documentation. <https://docs.flowiseai.com/>

React.js documentation. <https://react.dev/reference/react>

Tailwinds documentation. https://tailwindcss.com/docs/

Udemy <https://www.udemy.com/course/generative-ai-for-beginners-b/?kw=Generative+AI+for+Beginners&couponCode=NVDIN35>