Mid Semester Report

# Cover Page

**Title**: Code Summarization and Automatic Code Reviews with LLMs  
**Student Name**: Shalabh Raj  
**Roll Number**: 2022DC04190  
**Supervisor Name**: Sasidhar Venkata Naga  
**Department**: Data Science & Engineering  
**Institute**: Birla Institute of Technology and Science, Pilani  
**Date**: January 2025

# Dissertation Abstract

Code Summarization and Automatic Code Reviews with LLMs

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

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

For the **new people** joining the development group, it is 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 challenges** pose the problem for GenAI to solve it.

Large Language Models (**LLMs**) are **GenAI** models which would be **quickly learning** the language constructs of human and **interpret** the actions required. Since computer programming languages also follow specific syntax and construct, it would be easier if the **LLMs** are trained to learn these languages. Once they learn, they should be able to **translate** to plain English, **generate** new code, **understand** and **suggest** improvements on etc.

The **LLMs** like Gemini, GitHub CoPilot, OpenAI Codex, StarCoder2 etc. LLMs are trained learn multiples of programming languages and assist the developers to **summaries**, **explain**, **document** and **create new code** by assisting them during development phase in with a **chat window**.

**This project** is focused on building the **surrounding framework** to utilize **various models** and see how these models **perform** for **summarization**, **explanation**, **code review tasks** and which **model** is **more accurate** when given the **same context**. To prove the framework, this project also builds features to **evaluate** some of the **open-source models** and **train** some models for handling the **Q&A for assisting the developers**.

# List of Symbols & Abbreviations Used

|  |  |
| --- | --- |
| **Symbol/Abbreviation** | **Description** |
| LLMs | Large Language Models |
| HLD | High-Level Design |
| API | Application Programming Interface |
| UI | User Interface |

# List of Tables

1. [Table 1.1 Detailed Plan of Work](#_Table1.1_Detailed_Plan_1)

# List of Figures

1. [Figure 1.1: Architecture of Transformer-Based LLM Model](#_Figure_1.1_Architecture)
2. [Figure 1.2: Key Concept of LLM](#_Figure_1.2_Key)
3. [Figure 1.3: Lang chain Framework](#_Figure_1.3_Lang)
4. [Figure 1.4: HLD Design for the Chatbot Framework](#_Figure_1.4_HLD)

# Table of Contents

1. [Chapter 1: Introduction and Objectives](#_Chapter_1:_Introduction)  
2. [Chapter 2: Literature Review and Methodology](#_Chapter_2:_Literature)  
3. [Chapter 3: Work Completed So Far](#_Chapter_3:_Work)  
4. [Directions for Future Work](#_Directions_for_Future)  
5. [Bibliography/References](#_Bibliography_/_References)

# Chapter 1: Introduction and Objectives

## 1.1 Objectives

The objectives of my project are as follows:

1. To create a chatbot application designed to assist developers in various tasks. This chatbot will simplify code by translating it into plain English, making it more understandable.
2. It will also generate new code snippets, streamlining development processes and saving time. Additionally, the chatbot will provide recommendations for best coding practices, ensuring clean and efficient code.
3. It will further enhance security by identifying and highlighting vulnerabilities in the code, helping developers create safer applications. Overall, the chatbot will act as a comprehensive tool to improve productivity, code quality, and security in software development.

**Broad Area of Work:** LLMs, Open AI, Gemini, Ollama [mistral, gemma, llama], Hugging Face,Lang Chain, Lang Graph, Lang flow, Flowise

# Scope of Work

1. The project aims to develop an AI-powered chatbot using LLMs and frameworks like Lang Chain to assist developers. Key functionalities include translating code into plain English, generating code snippets, suggesting best practices, and identifying security vulnerabilities.
2. Lang Chain will handle workflow orchestration and context management.
3. The chatbot will feature a user-friendly interface with syntax-highlighted outputs and multi-language support.
4. After rigorous testing for accuracy and usability, the chatbot will be deployed on scalable platforms with API integration options.
5. Comprehensive documentation and a feedback loop will ensure continuous improvement, enhancing productivity, code quality, and security.

## 1.2 Objectives Met till Midterm

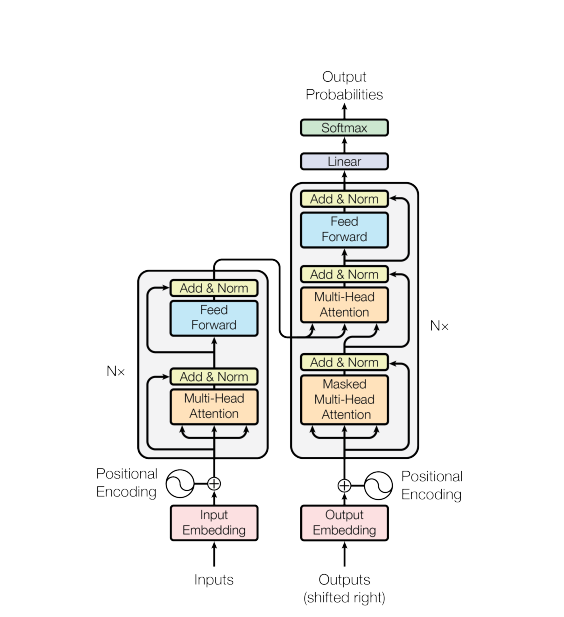
I have made a significant progress in understanding the architecture of LLMs , lang chain [and it’s surrounding frameworks like Lang graph , Lang flow , flowise ] and designing the high-level architecture for the chatbot framework.

# Chapter 2: Literature Review and Methodology

## 2.1 Literature Review

Relevant literature has been described below to understand the state-of-the-art in LLM applications for code summarization, generation, and review.

## Figure 1.1 Architecture of Transformer-Based LLM Model

****

(Source: https://proceedings.neurips.cc/paper\_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf ,” Attention Is All You Need”, by Vaswani et al.2017)

**Architecture Components**:

* Feed-forward layers, embedding layers, and attention layers form the backbone of LLMs.
* Input text is tokenized and embedded into vectors, then processed to generate predictions.

**Key Factors Influencing LLM Architecture**:

* Model size and parameter count.
* Input representations (how text is tokenized and embedded).
* Self-attention mechanisms for context-aware processing.
* Training objectives tailored to specific tasks.
* Computational efficiency for scaling and optimization.
* Decoding and output generation techniques.

**Transformer-Based LLM Components**:

* **Input Embeddings**: Convert tokens (words or sub-words) into dense vector representations, capturing semantic and syntactic information.
* **Positional Encoding**: Adds positional information to embeddings, enabling token order awareness.
* **Encoder**: Analyses input text using multiple encoder layers, each with:
  + **Self-Attention Mechanism**: Weighs the importance of tokens and captures relationships between them.
  + **Feed-Forward Neural Network**: Processes tokens independently to capture complex token interactions.
* **Decoder Layers**: (For autoregressive models) Generate sequential outputs by attending to previously generated tokens.
* **Multi-Head Attention**: Simultaneously applies attention with different learned weights to capture diverse relationships in the sequence.
* **Layer Normalization**: Stabilizes learning and improves generalization across inputs.
* **Output Layers**: Task-specific, such as SoftMax activation for language modelling.

## Figure 1.2 Key Concept of LLM

## 

**Prompt Engineering**

* Clearly convey the desired response
* Provide context or background information
* Balance simplicity and complexity
* Iterative testing and refinement

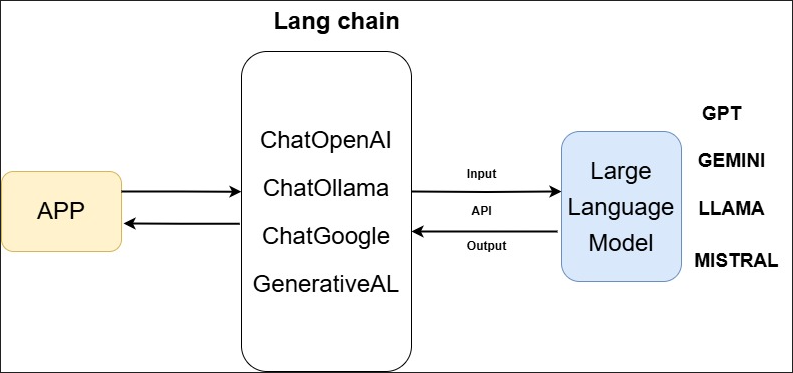
**Embeddings**

* Embeddings are numerical representation of text
* They are essential for AI models to understand and work with human language effectively

**Fine Tuning**

* Fine-tuning a LLM is the process of adapting a pre-trained model to perform specific task or to cater to a particular domain more effectively.

## Figure 1.3 Lang chain Framework



**What is Lang Chain?**

* Lang Chain is a framework that enables seamless integration of Large Language Models (LLMs) with external data, APIs, and logic to create advanced AI applications. It allows building workflows that leverage LLMs for tasks like question answering, summarization, and decision-making.
* Chains in Lang Chain are sequences of operations where the output of one step becomes the input for the next. They enable breaking down complex tasks into smaller, manageable processes.

**Prompts and Templates**

* Lang Chain supports custom prompt engineering, allowing developers to define reusable templates for specific tasks. Prompts guide LLMs' behavior and optimize their performance for use cases.

**Agents and Tools**

* Agents in Lang Chain dynamically decide which tools or actions to use during runtime. They interact with APIs, databases, or external systems to handle complex workflows.

**Applications**

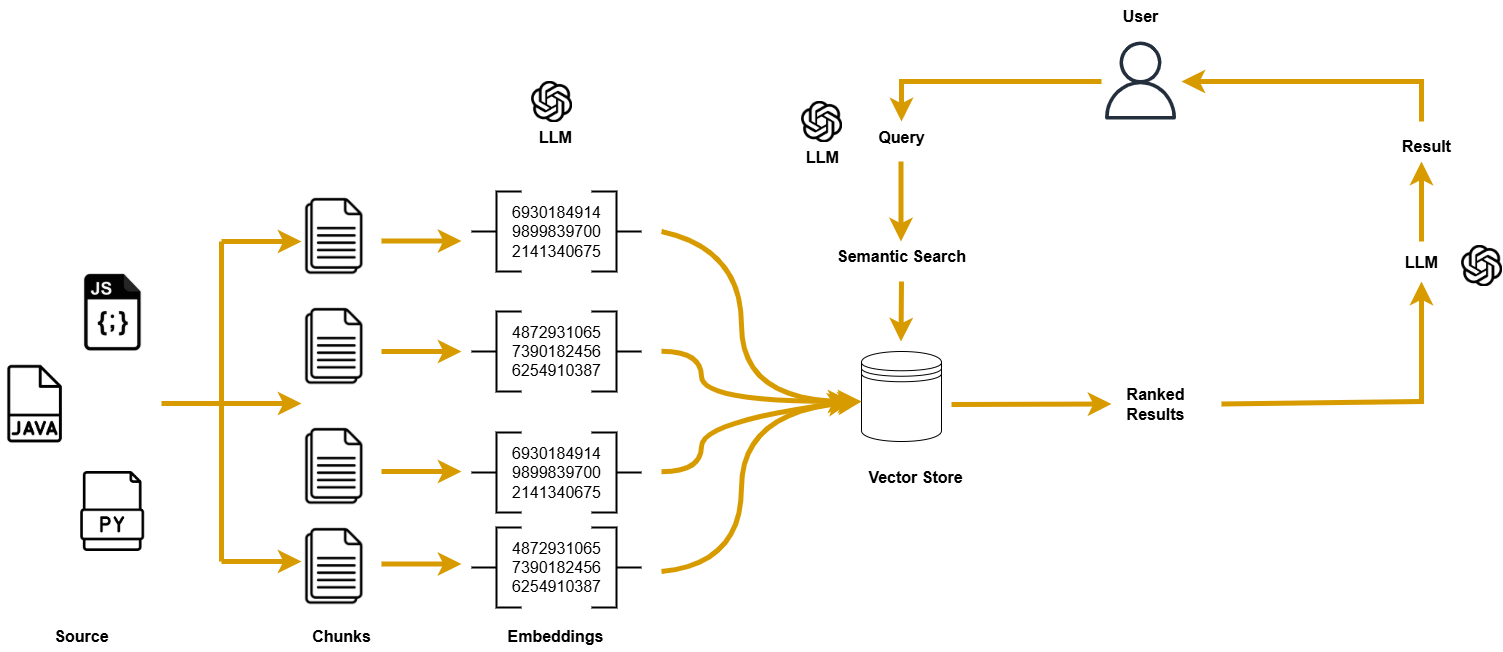
* Lang Chain is widely used for chatbots, data augmentation, document Q&A, and automating multi-step tasks with memory and scalable deployment options.

(Source: https://python.langchain.com/docs/introduction/,” Lang chain)

## 2.2 Architecture of Chatbot

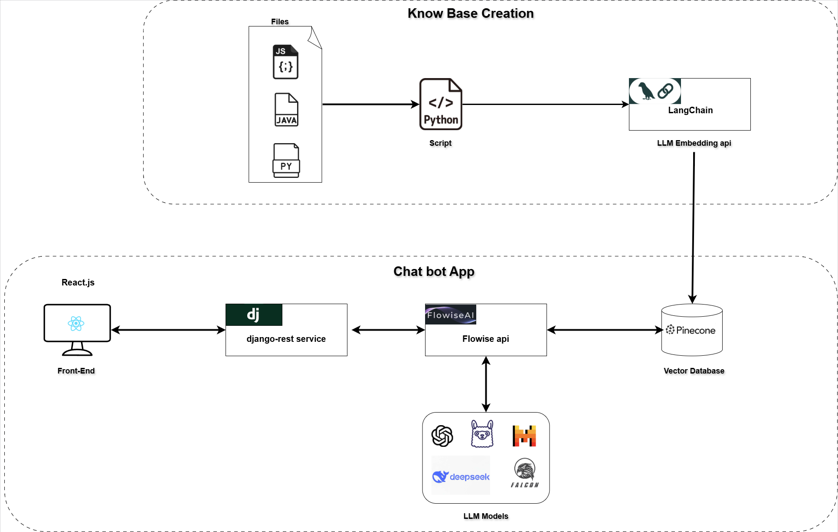
**Pre-trained LLM** models like ChatGPT are built using **publicly available data** and cannot access **or answer questions** about **private** or company-specific codes since they were **not trained** on them. Organizations often need AI to provide insights from their private knowledge repositories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

## Figure 1.4 HLD 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.

# Chapter 4: Low level Design Flow



## 4.1 Overview

Know Base creation Steps

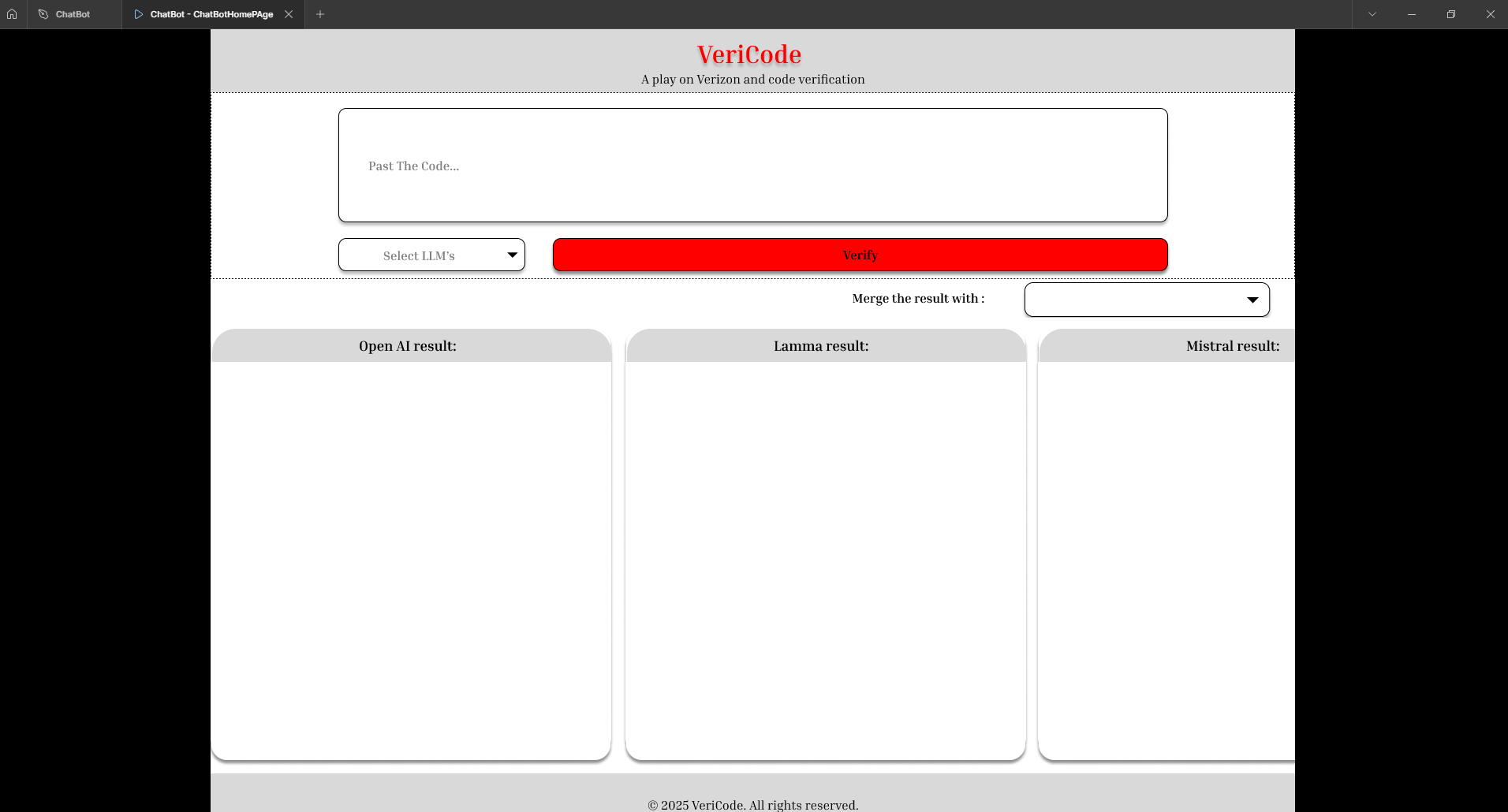
Chat bot App

## Models Selection

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **LLM Base Models** | | | | |
| Model | Parameters | Architecture | Context Length | Embedding Length |
| gpt-4o-mini | 8 billion | GPT-4 | 128 K tokens | 16 K tokens |
| 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 |
|  |  |  |  |  |

## 4.2 UI/UX Figma Design

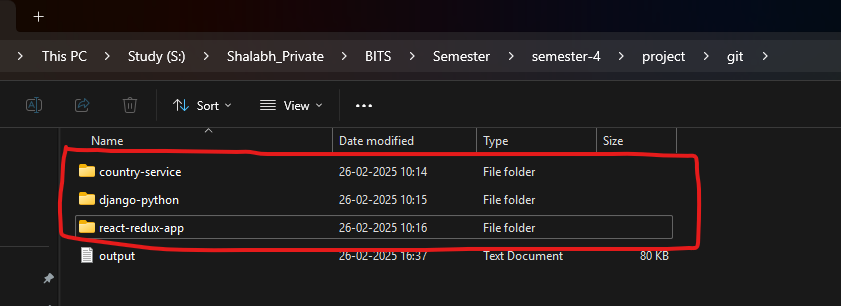
Figma Layout Url : <https://www.figma.com/design/FfhDNe03elOlaOfwtmHj4e/ChatBot?node-id=30-6&p=f&t=LRDJ1bGcGg1IkaG7-0>

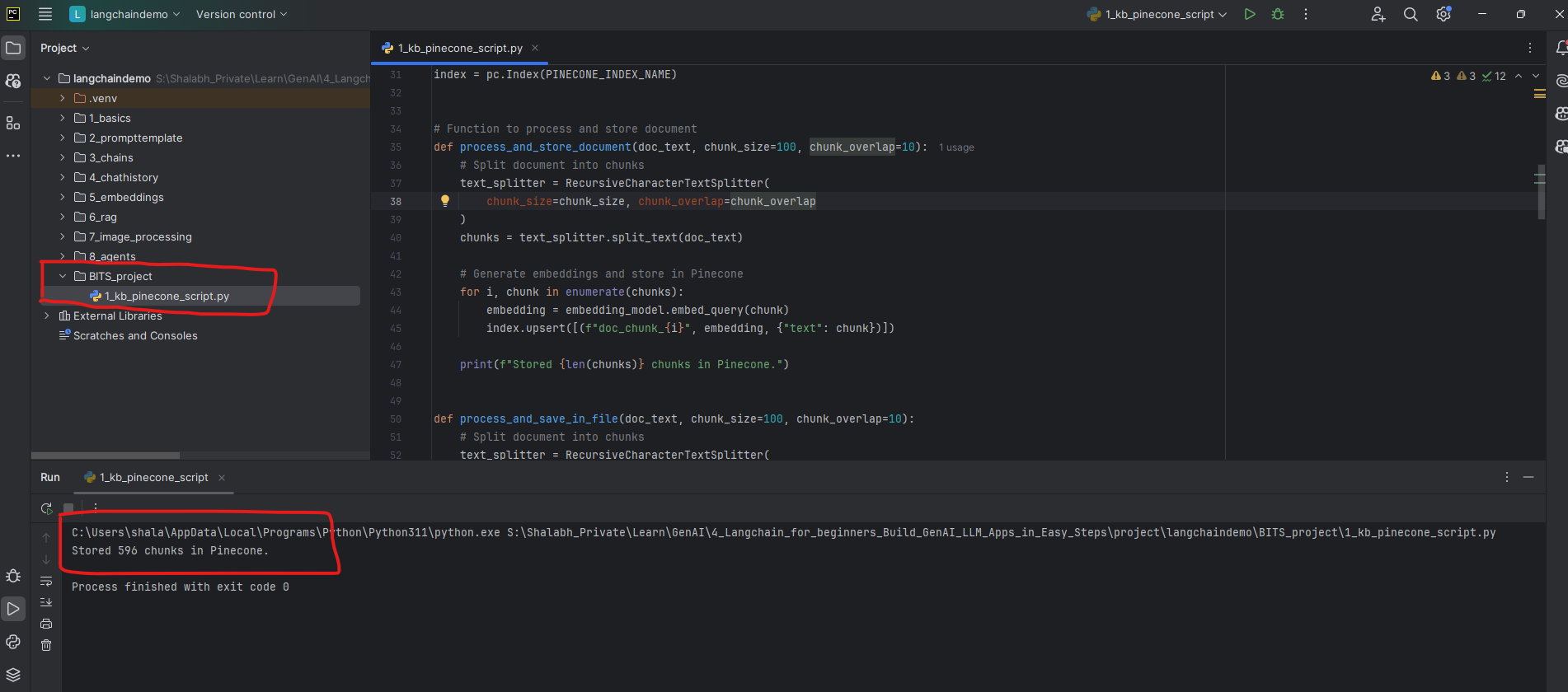


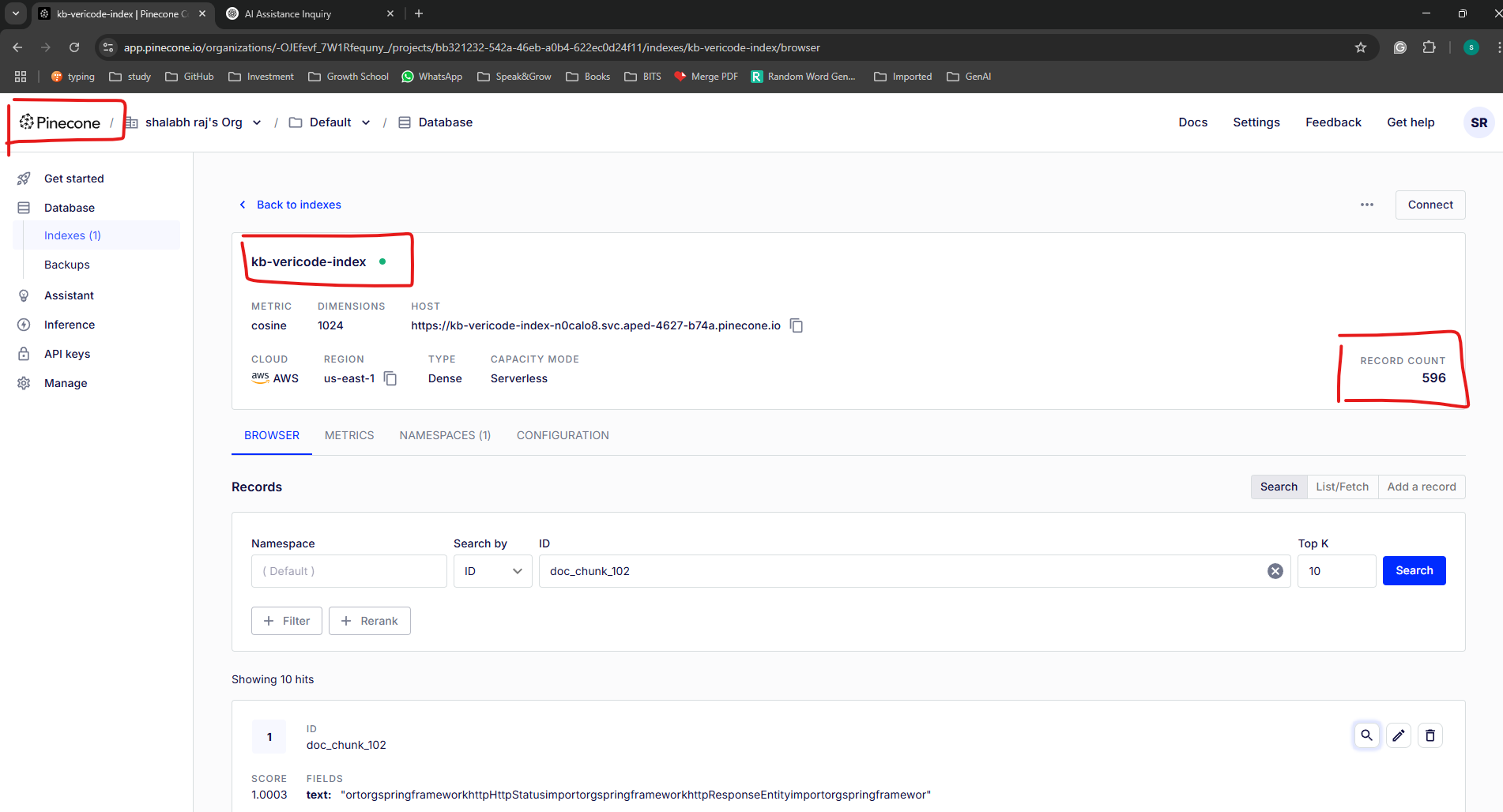
## 4.3 Knowledge Base Creation

Add here

Documents







Pinacone top 3 result gives better result

Chucnk 500 and chuck overlap 50

Add Video here

## 4.4 Flow Wise Sequence Diagram

## 4.5 End To End Flow

Add Video here

# Chapter 5: Future Release

Use Langhian to avoid dependency on flatwise

Use Agent AI

Create the IDE plugin for VS-code, IntelliJ

# Chapter 3: Work Completed So Far

## 3.1 Progress Overview

Progress includes designing the LangFlow pipeline, implementing API calls to invoke multiple LLMs, and initiating comparisons of outputs from different models.

## Table1.1 Detailed Plan of Work

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S. No | Taks | Start and End Date | Planned Weeks | Deliverables | Status |
| 1 | Understand Design and Architecture of LLMS and Lang chain frameworks | 09/12/2024– 30/12/2024 | 3 | HLD design for the chatbot framework | done |
| 2 | Build Lang chain frameworks-based pipelines to invoke LLMS from the chatbot framework | 30/12/2024-20/01/2025 | 3 | Expose Lang Flow/flowise Api which internally uses agents to invoke multiple LLMS | WIP |
| 3 | build chart bot framework and UI interface | 20/01/2025-03/02/2025 | 2 | Build a chatbot interface using any JavaScript library | WIP |
| 4 | built comparison AL/ML models to do comparisons of LLM outputs | 03/02/2025-17/02/2025 | 2 | Comparison between the various LLMS output and conclude which one is Best |  |
| 5 | testing and documentation | 17/02/2025-02/03/2025 | 2 | Documentations And Reports |  |

## 3.2 Challenges Encountered

* Challenges include selecting the LLMs for required tasks, as in market there are many options and selecting the list of LLM is one of the kinds of challenge.

**Models**: OpenAI, Gemini, Llama, gemma, mistral.

* Choosing the right farmwork for LLM orchestration and integrations

**Farmwork**: Lang chain, Lang graph, Lang flow(python), flowise(node), vector shift

* Choosing the right library/framework for build chatbot UI interface

**Farmwork**: React.js, Next.js, streamlit

## 3.3 Results Achieved

* I have developed a clear understanding of how to interact with LLM APIs, including making requests to process text inputs and receiving structured outputs. This includes learning how to fine-tune models, generate embeddings, and utilize various capabilities like text completion, summarization, and semantic search to solve specific problems.
* I have acquired a understanding of Lang Chain framework. This knowledge will allow me to build core functionalities for the chatbot application.
* I have studied and reviewed various courses and documentations related to LLMs in order to develop the architecture.

# Directions for Future Work

* I need to build and expose an API using the Lang Chain framework that utilizes multiple LLM models to analyze and compare the results.
* I need to develop a UI interface that will allow end users to submit code for review or ask questions related to Q&A.
* I need to Conduct rigorous testing to evaluate model accuracy and usability.
* Deploy the chatbot framework on scalable platforms with API integration options.

# Bibliography / References

* <https://en.wikipedia.org/wiki/Attention_Is_All_You_Need>
* Paper (Attention\_Is\_All\_You\_Need ) <https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>
* https://platform.openai.com/docs/api-reference/introduction
* <https://ai.google.dev/gemini-api/docs>
* <https://huggingface.co/docs>
* https://ollama.com/
* <https://www.udemy.com/course/generative-ai-for-beginners-b/?kw=Generative+AI+for+Beginners>
* <https://python.langchain.com/docs/introduction/>
* <https://langchain-ai.github.io/langgraph/tutorials/introduction/>
* <https://docs.langflow.org/>
* <https://docs.flowiseai.com/>
* <https://docs.streamlit.io/>
* https://react.dev/reference/react
* https://nextjs.org/docs