LLM-based Coding Assistant

Abdul Moiz
CMS ID : 398353
Advisor : S/L Dr. Raja Farrukh Ali
Co-Advisor : W/C Dr. Nayyer Aafaq
College of Aeronautical Engineering
School of Avionics and Electrical Engineering

February 2025

Contents

1 Introduction

The rapid advancement in Artificial Intelligence has paved the way for innovative tools in software development. This project aims to develop a coding assistant powered by an open-source Large Language Model (LLM) that is specifically optimized for specified languages programming. The system is designed for secure, offline deployment in environments such as Pakistan Air Force (PAF) facilities, where data security and operational independence are paramount. The assistant will enhance productivity by providing real-time specified languages code generation, completion, debugging, and refactoring support through a Visual Studio Code (VS Code) extension, while adhering to strict security protocols. In addition to traditional code generation, the assistant will leverage LLM-based agent capabilities to enable autonomous debugging, iterative code refinement, and integration with external verification tools (e.g., static analysis, linting, and unit testing frameworks) to improve contextual understanding and adaptive code improvements.

1.1 Project Title

Development of Large Language Model (LLM) based Coding Assistant

1.2 Objectives

The primary objectives of this project are to:

- Develop an AI-based coding assistant using an open-source LLM optimized for multiple coding languages.
- Evaluate and select an appropriate LLM based on performance, power requirements, and resource constraints.
- Integrate the LLM with an extension/plugin for VS Code or similar IDEs to deliver seamless, real-time assistance within various development workflows.
- Ensure the solution is deployable in a secure, networked offline environment (intranet) to satisfy operational and security requirements.
- Provide a user-friendly interface and detailed documentation to facilitate adoption and maintenance.
- Incorporate autonomous debugging, iterative refinement, and external tool integration (e.g., static analysis, linting, and unit testing) to continuously enhance code accuracy and reliability.

1.3 Scope

The scope of the project includes:

- Evaluation and selection of a suitable open-source LLM (e.g., LLaMA, Code Llama, or DeepSeek) based on performance and resource requirements.
- Curation of multi-language code repositories and datasets to support diverse coding languages.
- Development of an inference engine that minimizes latency and efficiently manages computational resources for offline networked operation.
- Creation of an extension/plugin that integrates the assistant with popular IDEs such as VS Code.
- Comprehensive testing and validation, including unit, integration, remote testing, and user acceptance testing.
- Addressing potential bottlenecks in offline deployment, including computational resource limitations, power requirements, and network latency.

1.4 Motivation

In mission-critical and secure environments such as those encountered in the PAF, the ability to operate independently of the internet is essential. An AI-driven coding assistant offers significant benefits by reducing development time, increasing code quality, and lowering the risk of human error. By leveraging open-source LLM technology and utilizing a Linux-based development environment, the project ensures cost-effectiveness, system-level adaptability, and the flexibility needed for customization—all while providing specialized support for specified languages coding practices and standards.

2 Project Overview

This project focuses on delivering an LLM-based coding assistant that seamlessly integrates with popular IDEs. The assistant will run on a networked offline deployment (intranet), providing developers with intelligent code suggestions, automated code generation, and real-time debugging and refactoring support. Its modular architecture is designed for scalability and continuous improvement, ensuring that iterative refinements and efficient resource management can be incorporated without disrupting service.

3 Literature Review

3.1 Large Language Models in Software Development

Recent studies demonstrate that Large Language Models can transform software development by automating code generation, debugging, and error detection. Research by Jiang *et al.* [3] highlights how LLMs contribute to increased efficiency and improved code quality. Additionally, evidence-based beliefs and behaviors of LLM-based programming assistants have been explored by Brown and Cusati [4].

Advancement: Recent advancements differentiate traditional LLMs from LLM-based agents. While traditional LLMs assist with code generation and debugging, LLM-based agents integrate external tools (e.g., Retrieval-Augmented Generation, static analysis, autonomous refactoring) to provide adaptive debugging, software refactoring, and test case generation. This shift enables real-time, iterative improvements and better integration within software engineering workflows, especially in specified languages-centric development environments.

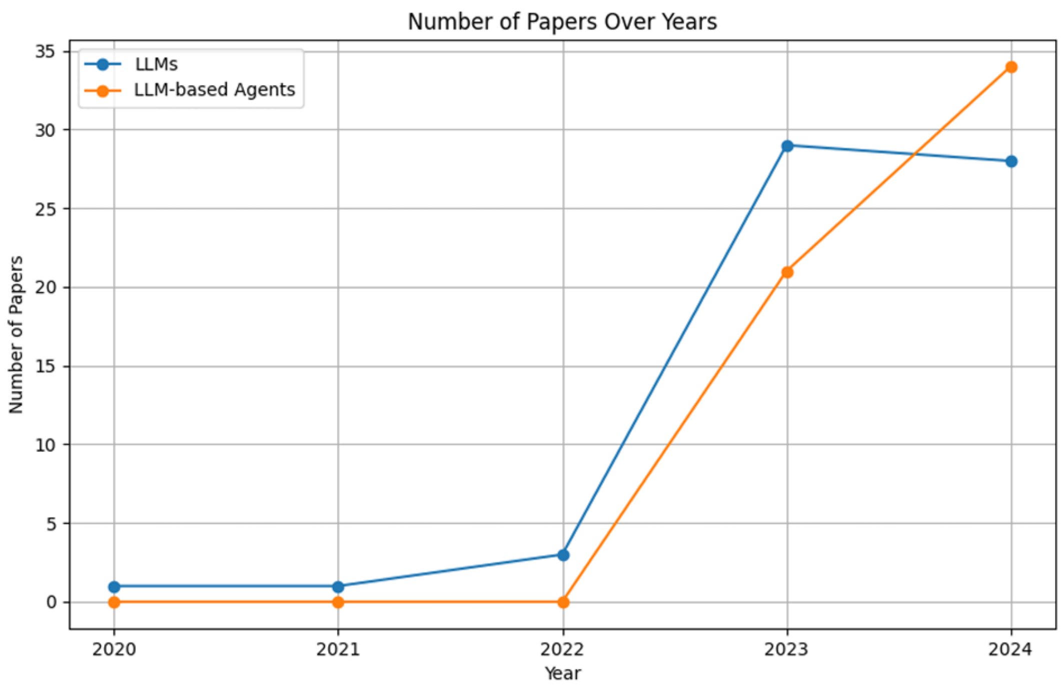


Figure 1: Adoption of LLM-based agents in software engineering (2020–2024) [4].

3.2 Open-Source LLMs

Open-source LLMs, such as LLaMA, Code Llama, and DeepSeek, have emerged as strong alternatives to proprietary models. These models offer robust architectures that can be fine-tuned for specialized tasks. Their open-source nature provides the transparency and adaptability required for secure, offline deployment in a specified languages development context.

3.3 Inference Engine for Offline Deployment

Offline deployment poses unique challenges, notably high computational demand and potential latency. Techniques such as model quantization and pruning will be considered to manage computational load in resource-constrained environments [7]. These strategies will be central to ensuring smooth and responsive specified languages code generation even in offline settings.

3.4 Integration with Development Environments

The integration of LLMs with integrated development environments (IDEs) is facilitated by standards such as the Language Server Protocol (LSP) [8]. This protocol enables the coding assistant to provide dynamic code suggestions, error detection, and debugging assistance directly within VS Code—streamlining the specified languages development workflow.

3.5 Linux in Model Creation and Adaptability

Linux will play a critical role throughout the development lifecycle. The Linux environment offers robust support for machine learning frameworks, efficient process management, and a wide range of performance monitoring tools. Specifically, Linux will be used to:

- Set up and configure the development environment for LLM selection, fine-tuning, and model retraining.
- Perform system-level adjustments and manage resource allocation for high-performance computing tasks.
- Execute automated scripts for model retraining using curated specified languages datasets, ensuring that the LLM is continuously refined for specified languages coding and applications.

3.6 Challenges and Limitations

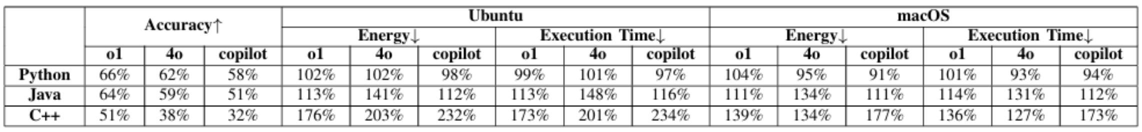
While promising, LLM-based coding assistants face several challenges:

- **Computational Overhead:** The resource-intensive nature of LLMs may lead to increased latency and necessitate significant hardware investments.
- **Model Reliability:** Variations in the generated code's accuracy and relevance require rigorous fine-tuning and continuous testing. Ambiguous outputs and a lack of credible sources necessitate the integration of verification mechanisms such as static analysis, linting, and unit testing.
- **Energy Inefficiencies:** Without targeted adjustments, LLM-generated code might consume more energy than desired in performance-critical applications.
- **Network Latency:** Even within an intranet, ensuring minimal latency during high-load periods is crucial.

4 Energy Efficiency of LLM-Generated Code

Evaluating the energy efficiency of LLM-generated code is essential, particularly when deploying AI models in resource-constrained, offline environments. A study by Solovyeva *et al.* [1] examines the energy performance of code generated by LLMs across different programming languages. Figure 2 shows a comparative analysis of energy efficiency for Python, Java, and C++.

Furthermore, our approach will incorporate energy-aware code evaluation metrics and fine-tuning techniques designed to optimize energy consumption in secure military or air force environments, ensuring that the generated specified languages code is both performance-critical and energy-efficient.



	Accuracy↑			Ubuntu						macOS					
				Energy↓			Execution Time↓			Energy↓			Execution Time↓		
	o1	4o	copilot	o1	4o	copilot	o1	4o	copilot	o1	4o	copilot	o1	4o	copilot
Python	66%	62%	58%	102%	102%	98%	99%	101%	97%	104%	95%	91%	101%	93%	94%
Java	64%	59%	51%	113%	141%	112%	113%	148%	116%	111%	134%	111%	114%	131%	112%
C++	51%	38%	32%	176%	203%	232%	173%	201%	234%	139%	134%	177%	136%	127%	173%

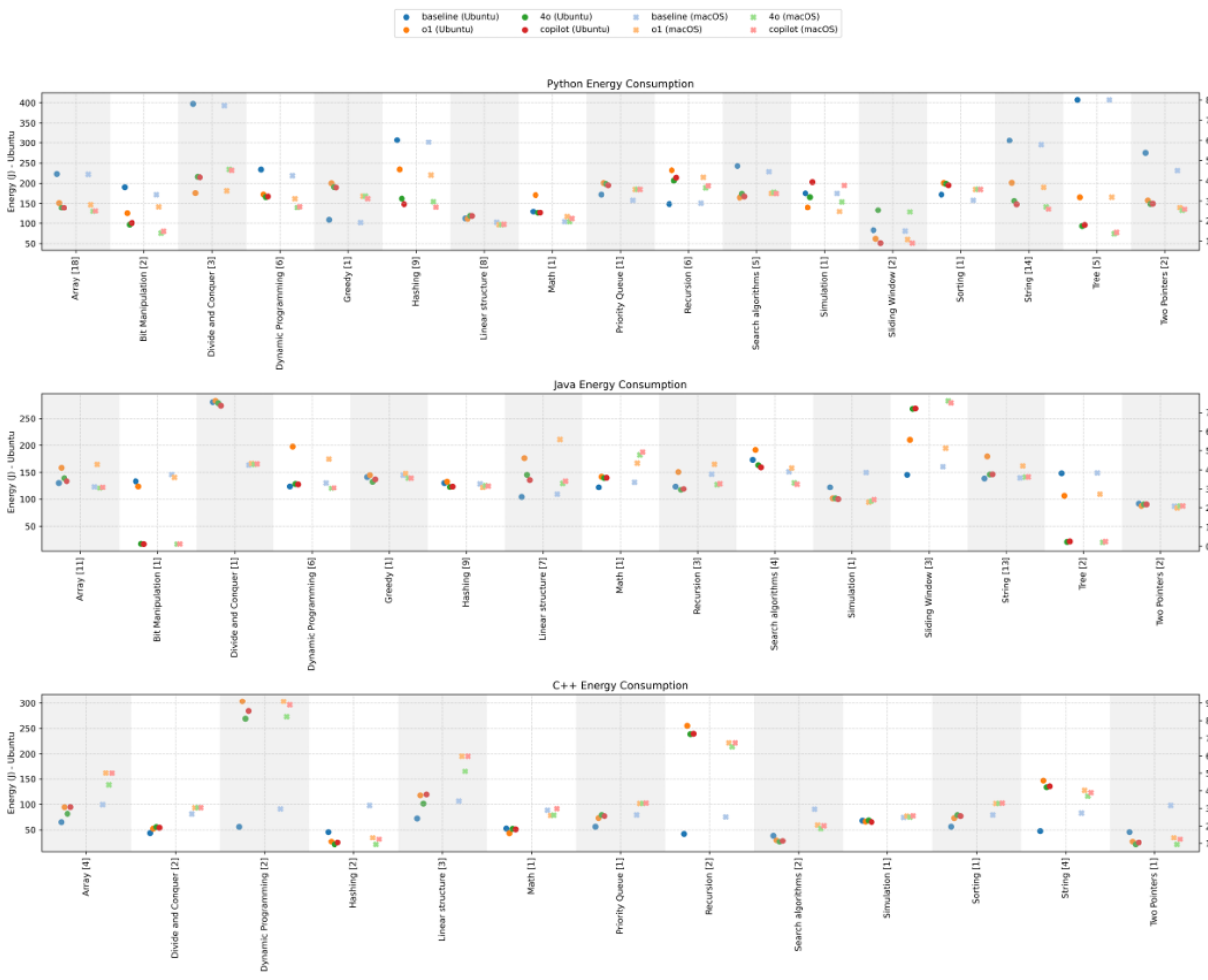


Figure 2: Energy efficiency of LLM-generated code across Python, Java, and C++ [1].

5 System Architecture and Methodology

The proposed system comprises the following key components:

1. **LLM Engine:** Serves as the core processing unit, utilizing an LLM-based agent approach (e.g., LLaMA, Code Llama, or DeepSeek) that integrates external tools for autonomous debugging, static code analysis, and iterative refinement through user feedback.

2. **Inference Engine:** A backend module designed to provide low-latency responses while efficiently managing computational resources during offline operation.

3. **IDE Extension:** Acts as the front-end interface, integrating the LLM's capabilities into the VS Code or any other IDE environment through the Language Server Protocol (LSP), and providing a user-friendly experience tailored for specified languages developers.

The project will follow an Agile development methodology with iterative cycles that allow for continuous testing, refinement, and integration.

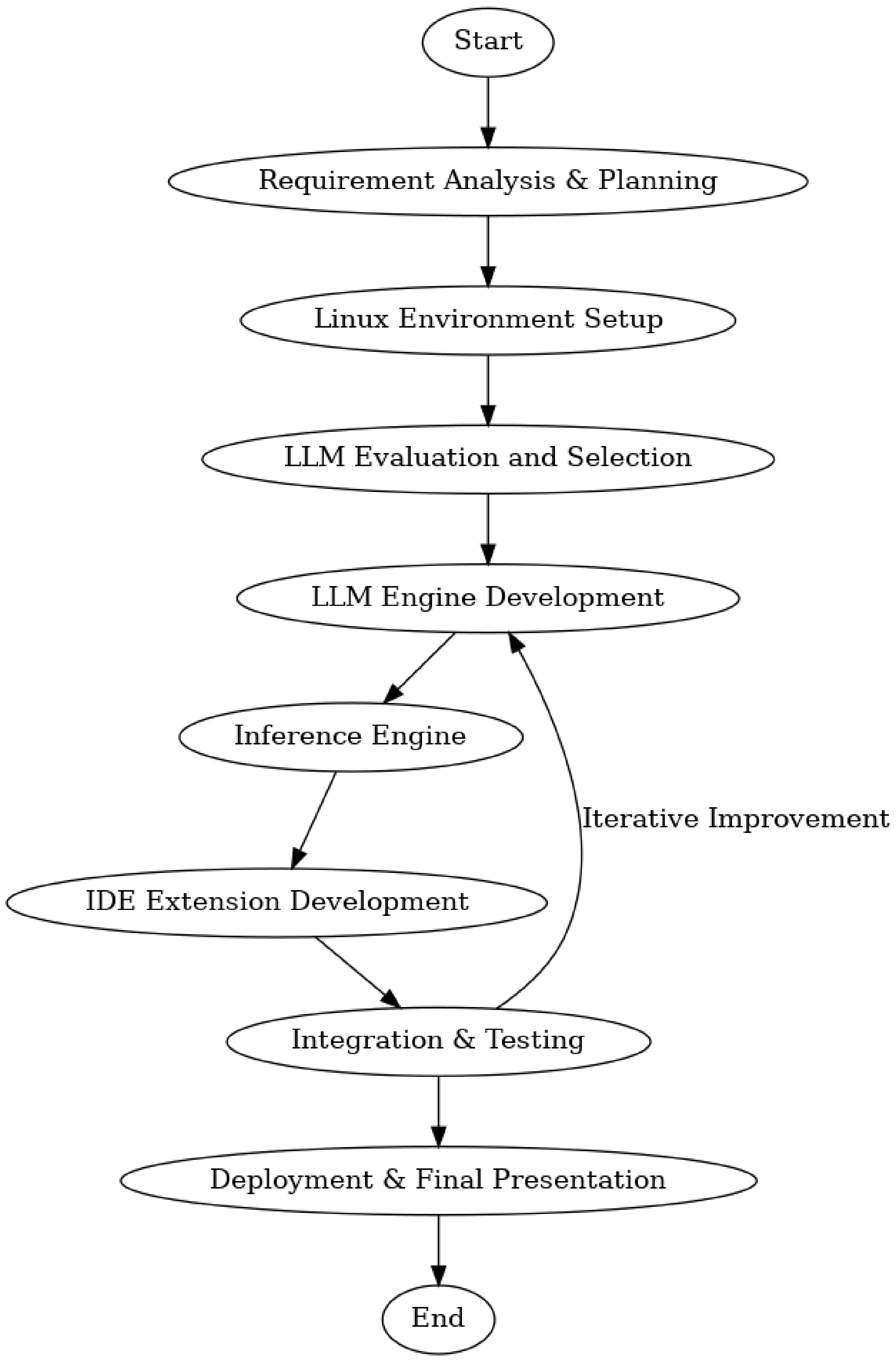

Start
Requirement Analysis & Planning
Linux Environment Setup
LLM Evaluation and Selection
LLM Engine Development
Inference Engine
IDE Extension Development
Integration & Testing
Iterative Improvement
Deployment & Final Presentation
End

Figure 3: Project Flowchart

6 Risk Analysis and Mitigation

6.1 Risk Identification

The project faces several key risks:

- **High Computational Demand:** The resource-intensive nature of LLMs may lead to increased latency and necessitate significant hardware investments.

- **Variability in Model Performance:** Inconsistent accuracy in code generation due to contextual variations may affect user trust and system reliability. Ambiguous outputs and potential hallucinations require robust verification and feedback mechanisms.

- **Energy Inefficiencies:** Without targeted adjustments, LLM-generated code might consume more energy than desired in performance-critical applications.

6.2 Mitigation Strategies

To address these risks, the following strategies will be implemented:

- **Robust Testing and Verification:** A thorough testing protocol, including unit, integration, and user acceptance tests, combined with static analysis and linting, will ensure consistent performance and reliability.

- **Enhanced Security Measures:** Best practices for local security, including regular vulnerability assessments and secure data handling procedures, will be strictly followed.

7 Expected Outcomes and Deliverables

Upon successful completion, the project will deliver:

- A stand-alone deployable LLM that can act as a Coding Assistant.
- A graphical user interface (GUI) and a backend inference engine to interact with the LLM.
- An IDE Extension that can be installed in a local PC to provide services.

8 Resources Required

The execution of this project requires the following resources:

- **Hardware:** High-performance servers equipped with GPUs to support intensive model inference. Recommended GPU specifications include:
  - For full-scale LLMs (e.g., LLaMA): NVIDIA RTX 3090 or NVIDIA A6000 with at least 24GB VRAM.
- **Software:**
  - Python3, pip3, and relevant libraries (e.g., PyTorch, Transformers) for development, testing, and deployment.
  - Visual Studio Code with specified languages extensions for extension development.
  - Linux OS for a stable and adaptable development environment.
- **Data:** Access to curated specified languages code repositories and datasets for effective fine-tuning and retraining of the LLM.
- **Budget:** Financial resources for hardware procurement, software licensing, personnel training, and ongoing maintenance.

9 Milestones and Timeline

The project is structured over a total of 32 weeks, divided into two semesters. The timeline is outlined in Table 1.

Milestone	**Duration (Weeks)**	**Description**
First Semester (Weeks 1–16)		
Project Definition Document	4	Finalize the detailed project definition document and plan for the coding assistant.
LLM Evaluation and Selection	6	Evaluate and select an appropriate open-source LLM (e.g., LLaMA, Code Llama, or DeepSeek) for specified languages code generation.
Dataset Curation	6	Curate code repositories and fine-tune the LLM for tasks such as code generation, debugging, and refactoring.
7th Semester Presentation	14	Present intermediate results at Week 14.
Second Semester (Weeks 17–32)		
IDE Extension Development	4	Design and develop an extension integrating the LLM for use in IDEs.
System Integration and Verification	8	Integrate the developed modules and verify system functionality through testing and feedback.
Testing & Final Report Submission	4	Conduct comprehensive testing and prepare the final project report.
8th Semester Mid Presentation	24	Present intermediate results at Week 24.
Final Presentation	30	Deliver the comprehensive final project presentation at Week 30.

Table 1: Project Milestones and Implementation Timeline (32 Weeks Total)

10 Gantt Chart

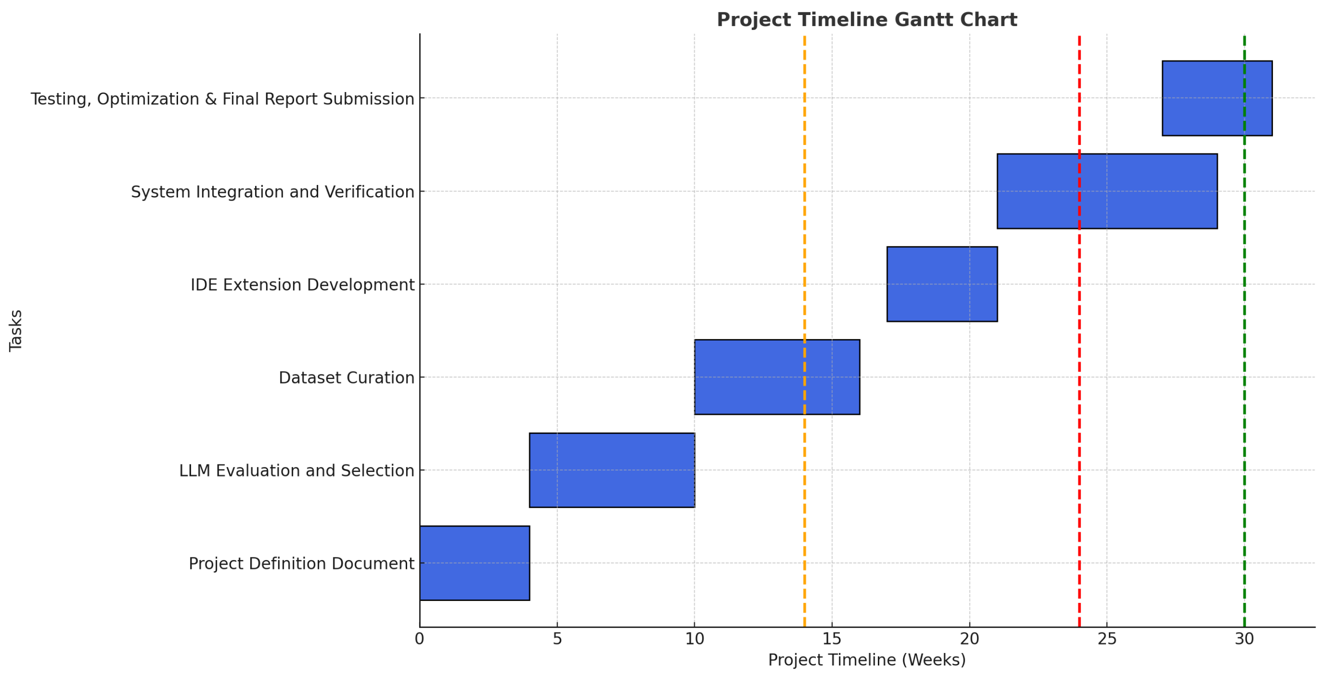


Figure 4: Project Gantt Chart

11 Evaluation

The system will be evaluated based on the following key performance metrics:

- **Accuracy:** The correctness and contextual relevance of specified languages code suggestions generated by the LLM.
- **Latency:** The response time for code generation and debugging support during interactive sessions.
- **Resource Utilization:** Efficiency in the usage of CPU, GPU, memory, and storage resources during offline operations.

12 Conclusion and Future Work

This project aims to create a state-of-the-art LLM-based coding assistant that significantly enhances the productivity and quality of software development in secure, offline environments. By leveraging open-source LLMs, a Linux-based development environment, and integrating these capabilities with a user-friendly extension for IDEs, the project sets a strong foundation for future innovations. The incorporation of advanced

LLM-based agent capabilities, rigorous verification mechanisms, and a system integration module that iteratively refines code outputs will ensure that the system remains robust and adaptable to evolving software engineering needs.

Future work may involve:

- Extending support to additional specified languages frameworks and libraries.
- Incorporating advanced context-aware debugging features.
- Continuously refining the model based on user feedback and evolving development practices.

13 References

References

[1] L. Solovyeva, S. Weidmann, and F. Castor, “AI-Powered, But Power-Hungry? Energy Efficiency of LLM-Generated Code,” *University of Twente*, 2025.

[2] L. Tao *et al.*, “CODELUTRA: Boosting LLM Code Generation via Preference-Guided Refinement,” *arXiv preprint arXiv:2411.05199*, 2024.

[3] J. Jiang *et al.*, “A survey on large language models for code generation,” *arXiv preprint arXiv:2406.00515*, 2023.

[4] A. Brown and B. Cusati, “From LLMs to LLM-based Agents in Software Development: Evolution and Impact,” *International Journal of Advanced Software Engineering*, vol. 10, no. 1, pp. 34–49, 2024.

[5] Y. Li *et al.*, “A review on edge large language models: Design, execution, and adaptability,” *arXiv preprint arXiv:2410.11845*, 2023.

[6] J. Wang *et al.*, “A review on code generation with LLMs: Application and evaluation,” in *Proc. IEEE/ACM Int. Conf. Software Engineering*, 2023, pp. 1234–1245.

[7] A. Kumar *et al.*, “Deploying AI models in offline environments: Challenges and techniques,” *IEEE Trans. Ind. Informatics*, vol. 17, no. 5, pp. 3211–3221, May 2021.

[8] Xlang AI, “Paper collection for LLM code generation,” 2023. [Online]. Available: `https://github.com/xlang-ai/xlang-paper-reading/blob/main/llm-code-generation.md`

Advisor:

Date:____________________________ **S/L Dr. Raja Farrukh Ali**

Co-Advisor:

Date:____________________________ **W/C Dr. Nayyer Aafaq**

Student Bio Data

- **Name:** GC Abdul Moiz
- **Father's Name:** M. Javaid Afzal
- **CNIC:** 3-84031412922-1
- **Current Address:** Room 417, Cadets Mess Block 4
- **Permanent Address:** House no. 691 B Gulberg city Sargodha
- **Academic Record:**
  - **Matric:** 1013/1100
  - **FSC:** 906/1100
  - **CGPA:** 2.26/4
- **School:** Govt. Comprehensive High School Sargodha
- **College:** Superior College Sargodha