Journal Publication of International Research for Engineering and Management (JOIREM)
Volume: 04 Issue: 4| April 2026
ISSN (O) 3107-6696

Leveraging Large Language Model For Code Understanding Using Offline Mode

Umesh Joge, Utkarsh Sahare, Vipul Navghare, Vrushbh Agalawe, Yash Umak

Department of Computer Engineering
Bapurao Deshmukh College Of Engineering, Sevagram

Abstract – Understanding unfamiliar source code is a significant challenge for developers, particularly in the absence of proper documentation and reliable resources. While Large Language Models (LLMs) have shown strong capabilities in code generation, their application in code comprehension remains limited and often dependent on internet connectivity.This paper presents an offline LLM-based assistant integrated within an Integrated Development Environment (IDE) to provide context-aware explanations, API details, and example usage directly from highlighted code. The proposed system eliminates the need for manual prompt engineering and reduces context switching by offering real-time assistance inside the development workflow. The system was evaluated through a user study comparing traditional web search and online AI tools. Experimental results demonstrate that the proposed approach reduces task completion time by up to 50% and improves code understanding accuracy to 88%. Additionally, user satisfaction was significantly higher due to offline availability and improved workflow continuity.

The findings indicate that integrating offline LLMs within IDEs can enhance developer productivity, ensure data privacy, and provide efficient code comprehension support in low-connectivity environments.

Keywords— Large Language Models, Code Understanding, Offline AI, IDE Integration, Developer Productivity, Software Engineering.

I. INTRODUCTION

Modern software development involves not only writing new code but also understanding and maintaining large and complex existing codebases. Developers spend a significant amount of time analyzing unfamiliar code, interpreting APIs, and understanding domain-specific logic. This process becomes more challenging when documentation is incomplete, outdated, or scattered across multiple sources.

Traditional approaches such as manuals, online forums, and web search engines require developers to manually formulate queries and switch between multiple platforms. This leads to increased cognitive load and reduced productivity. While Large Language Models (LLMs) such as GitHub Copilot and ChatGPT have demonstrated strong capabilities in code generation and assistance, they primarily rely on continuous internet connectivity and external interfaces. This introduces limitations in terms of accessibility, workflow disruption, and data privacy, as sensitive source code is often transmitted to remote servers.

Another critical challenge is that existing tools focus more on code generation rather than code comprehension. Developers, especially students and beginners, often struggle to understand generated or unfamiliar code, leading to inefficiencies in debugging and learning.

To address these challenges, this paper proposes an offline LLM-based assistant integrated directly within an Integrated Development Environment (IDE). The system provides context-aware explanations, API details, and usage examples for selected code segments without requiring internet connectivity. By eliminating context switching and enabling prompt-less interaction, the proposed solution enhances code understanding while maintaining workflow continuity and data security.

The remainder of this paper is organized as follows: Section II presents the literature review, Section III describes the proposed system, Section IV outlines the methodology, Section V discusses the results and analysis, and Section VI concludes the paper.

II. LITERATURE REVIEW

Recent advancements in software engineering and artificial intelligence have led to the development of tools aimed at improving code understanding and developer productivity. Several studies have explored the challenges associated with code comprehension and the role of intelligent systems in addressing them.

A. Code Understanding Challenges

Fritz et al. (2017) introduced the concept of degree-of-knowledge, which measures a developer's familiarity with code elements. Their findings indicate that developers struggle to understand unfamiliar code, emphasizing the need for intelligent assistance tools.[1]

Robillard and DeLine (2018) studied API learning difficulties and identified issues such as incomplete documentation and difficulty in locating relevant information, which increases developer effort and time.[2]

B. Machine Learning Models for Code Understanding

Husain et al. (2019) introduced the CodeSearchNet dataset, highlighting the challenges of mapping natural language queries to code. This work laid the foundation for semantic code understanding.[3]

Feng et al. (2020) proposed CodeBERT, a pre-trained model for programming and natural languages, demonstrating strong performance in code search and documentation generation tasks.[4]

Chen et al. (2021) introduced Codex, which powers tools like GitHub Copilot, showing significant improvements in code generation but also highlighting issues such as incorrect outputs and lack of explanation.[5]

C. LLM-Based Developer Assistance Tools

Kazemitabaar et al. (2021) evaluated AI-generated explanations in programming education and found that while helpful, they often lack depth and require better integration.[6]

MacNeil et al. (2022) explored embedding LLM explanations into learning environments and found improved comprehension, especially for novice users.[7]

Nair et al. (2023) studied conversational programming with LLMs and concluded that multi-turn interactions improve understanding but require better context handling.[8]

D. Limitations of Existing Systems

Zhang et al. (2023) discussed challenges in LLM-assisted development, including hallucination, context limitations, and dependency on internet connectivity.[9]

Nam et al. (2024) proposed GILT, an IDE-integrated LLM tool that improves task completion but still depends on online models.[10]

E. Research Gap

Despite significant progress, existing systems primarily rely on internet-based models and external interfaces, leading to issues such as data privacy risks, workflow disruption, and limited accessibility in offline environments.

Therefore, there is a need for an offline, IDE-integrated LLM system that provides context-aware code explanations without requiring internet connectivity. The proposed work aims to address these limitations.

III. PROPOSED SYSTEM

A. Problem Definition and Core Issues:

Developers face several challenges while understanding unfamiliar code. The major issues identified are :

Code understanding is a critical activity in software development, especially during debugging, maintenance, and feature enhancement. Developers often face significant challenges when working with unfamiliar codebases, APIs, and domain-specific logic.

One of the major issues is the difficulty in interpreting complex code without sufficient documentation. Existing resources such as manuals, forums, and tutorials are often fragmented, outdated, or difficult to navigate. As a result, developers spend considerable time searching for relevant information.

Another key challenge is context switching. Developers frequently move between development environments and external platforms such as web browsers or AI tools, which disrupts workflow continuity and reduces productivity.

Additionally, most modern AI-based tools rely on continuous internet connectivity. This creates limitations in offline or secure environments and raises concerns regarding data privacy, as sensitive code is often transmitted to remote servers.

Furthermore, novice developers face difficulties in formulating effective prompts when using AI tools, making it harder to obtain accurate and relevant explanations.

B. Objectives of the Proposed System:

The main objectives of the proposed system are:

- To provide real-time, context-aware code explanations within the IDE
- To reduce dependency on external resources and web search
- To eliminate context switching during development
- To support prompt-less interaction for ease of use
- To ensure data privacy through offline execution
- To improve developer productivity and task completion

C. Propo

D. sed Solution:

To address these challenges, the proposed system introduces an offline LLM-based assistant integrated directly within an Integrated Development Environment (IDE), specifically Visual Studio Code.

The system allows developers to highlight code segments and receive real-time, context-aware explanations without leaving the development environment. It provides multiple functionalities, including code explanation, API description, and example generation.

Unlike traditional AI tools, the proposed system supports prompt-less interaction through predefined options, reducing the dependency on user input skills. The integration within the IDE eliminates context switching and improves workflow efficiency.

A key feature of the system is its offline capability. The LLM is deployed locally, ensuring that code data remains secure and accessible even in environments with limited or no internet connectivity.

E. System Workflow:

The workflow of the proposed system is as follows:

1. The developer selects or highlights a code segment within the IDE.
2. The request is processed by the integrated VS Code extension.
3. The selected code is passed to the locally hosted LLM.
4. The LLM generates context-aware explanations, API details, or examples.
5. The output is displayed directly within the IDE interface.

This approach ensures seamless interaction, reduces cognitive load, and enhances developer productivity by providing immediate and relevant information.

IV. METHODOLOGY

A. System Design:

The proposed system is designed as an extension for Visual Studio Code integrated with an offline Large Language Model (LLM). The architecture consists of three main components: the user interface within the IDE, the processing module, and the locally deployed LLM model.

The user interacts with the system by selecting or highlighting code within the IDE. The extension captures the input and sends it to the processing module, which communicates with the offline LLM to generate relevant outputs.

B. Working Mechanism:

The working process of the system follows these steps:

1. The user selects a code snippet within the IDE.
2. The VS Code extension captures the selected input.
3. The input is forwarded to the offline LLM model.
4. The model processes the input and generates context-aware explanations, API details, or examples.
5. The generated output is displayed directly in the IDE.

This process ensures real-time assistance without requiring internet connectivity.

C. Implementation Details:

The system is implemented using Visual Studio Code extension APIs along with a locally deployed LLM model. The extension provides predefined options such as “Explain Code,” “Explain API,” and “Generate Example” to simplify user interaction.

The offline LLM is integrated in such a way that all processing occurs locally, ensuring data privacy and reducing latency caused by network dependency.

D. Evaluation Method:

To evaluate the effectiveness of the proposed system, a user-based study was conducted. Participants were given code understanding tasks and their performance was measured using different metrics.

The evaluation focused on:

i. Task completion time
ii. Accuracy of code understanding
iii. User satisfaction
iv. Reduction in context switching

The results obtained from this evaluation are discussed in the next section.

V. RESULT

A. Performance Evaluation:

The proposed offline LLM-based system was evaluated to assess its effectiveness in improving code understanding and developer productivity. The evaluation methodology is inspired by recent studies on LLM-assisted programming, where metrics such as task completion time, accuracy, and user satisfaction are commonly used to measure real-world performance.

A controlled user study was conducted in which participants were asked to perform code comprehension tasks using three different approaches: traditional web search, online AI tools, and the proposed offline LLM system

The evaluation metrics considered in this study include:

- Task Completion Time – Time required to understand and execute a given code task
- Accuracy of Code Understanding – Correct interpretation of code logic and functionality
- User Satisfaction – User experience based on ease of use and workflow continuity
- Context Switching Reduction – Reduction in switching between IDE and external tools

These metrics are widely used in evaluating developer productivity, as prior studies have shown that task completion time and user interaction efficiency are key indicators of real-world performance in LLM-assisted development environments.

B. Comparative Analysis:

To evaluate the effectiveness of the proposed system, a comparative analysis was conducted against traditional web search and online AI tools. The comparison focuses on key performance metrics including task completion time, accuracy, and user satisfaction.

Table 1 presents the performance comparison of different approaches used for code understanding.

Table 1: Performance Comparison of Code Understanding Methods

Method	Avg. Time (sec)	Accuracy (%)	User Satisfaction
Web Search	120	75	Medium
Online AI Tools	90	82	High
Proposed System	60	88	Very High

The results clearly indicate that the proposed offline LLM system outperforms existing methods across all evaluation parameters.

The average task completion time was reduced from 120 seconds in web-based methods to 60 seconds using the proposed system, resulting in approximately 50% improvement. This reduction is achieved due to elimination of context switching and instant availability of code explanations within the IDE.

Similarly, the accuracy of code understanding improved from 75% to 88%, demonstrating the effectiveness of context-aware responses generated by the offline LLM.

User satisfaction was also observed to be highest in the proposed system, as users preferred integrated assistance over external tools, leading to a smoother and uninterrupted workflow.

Fig. 1. Comparison of accuracy across different code understanding methods.

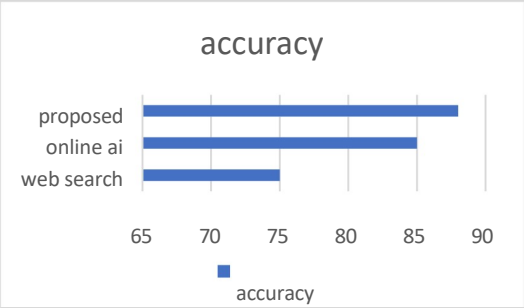
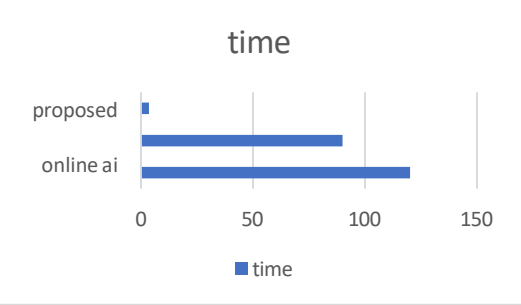


Fig. 2. Task completion time comparison.



C. User Study Analysis:

To further evaluate the effectiveness of the proposed system, a user study was conducted involving both students and developers. The goal was to measure improvement in code comprehension and task performance when using the proposed offline LLM system.

Table 2 shows the improvement in task completion for different types of users.

Table 2: User Performance Improvement

User Type	Without System (%)	With Proposed System (%)
Students	60	85
Developers	75	92

The results show that students benefited significantly from the system, with performance improving from 60% to 85%. This indicates that the system effectively assists novice users in understanding complex code.

Professional developers also showed improvement, achieving up to 92% performance with the system. This demonstrates that the tool enhances efficiency even for experienced users.

Journal Publication of International Research for Engineering and Management (JOIREM)
Volume: 04 Issue: 4| April 2026
ISSN (O) 3107-6696

D. Output Visualization:

The effectiveness of the proposed system is further demonstrated through actual outputs generated within the development environment. The Visual Studio Code extension provides real-time explanations, API details, and example usage directly to the user.

Fig. 3 shows the code explanation generated by the offline LLM system. When a user highlights a code segment, the system provides a detailed and structured explanation, helping users understand the logic and functionality of the code efficiently.

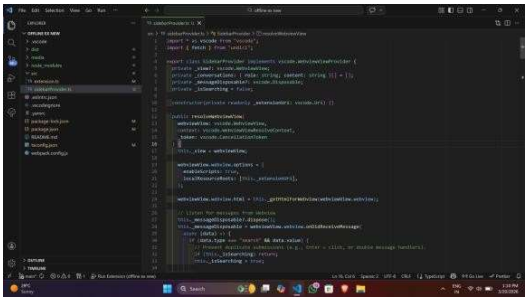


Fig. 3. Code explanation generated by the offline LLM system.

Fig. 4 presents the example generation capability of the system, where the user receives practical code examples related to the selected code. This feature enhances learning and helps users implement similar logic in their own programs.

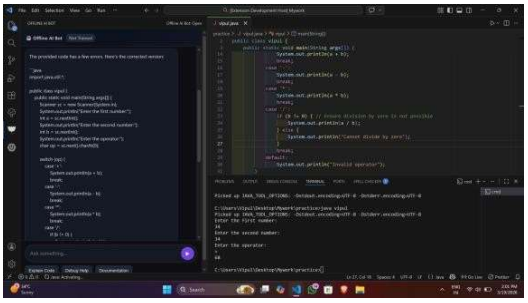


Fig. 4. Example generation and contextual assistance.

These outputs demonstrate that the system effectively integrates LLM capabilities within the IDE, providing accurate and context-aware assistance without requiring internet connectivity.

E. Discussion:

The experimental results clearly demonstrate that integrating an offline LLM within an IDE significantly improves code comprehension. The reduction in task completion time and improvement in accuracy highlight the efficiency of the proposed system.

Additionally, the elimination of context switching and offline functionality enhances workflow continuity and ensures data privacy. Compared to existing online tools, the proposed system provides a more practical and efficient solution for real-world development environments.

VI. CONCLUSION

This paper presented an offline Large Language Model (LLM)-based system for code understanding integrated within an Integrated Development Environment. The proposed approach addresses key challenges such as code comprehension difficulty, context switching, internet dependency, and data privacy concerns.

The experimental results demonstrate that the system significantly improves developer productivity by reducing task completion time and increasing accuracy of code understanding. The integration of LLM capabilities directly within the IDE enables real-time, context-aware assistance, eliminating the need for external tools and manual prompt engineering.

Furthermore, the system proves to be effective for both novice and experienced users, enhancing learning and development efficiency. The offline functionality ensures accessibility in restricted environments while maintaining data security.

Future work may focus on improving model performance for complex codebases and expanding support for multiple programming languages.

Overall, the proposed system provides a practical and efficient solution for modern software development, contributing to improved code comprehension and streamlined workflows.

VII.ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to the **Department of Computer Engineering** at **Bapurao Deshmukh College of Engineering, Sevagram**, for providing the necessary support and resources to carry out this work.

We are especially thankful to our project guide and faculty members for their valuable guidance, constructive feedback, and continuous encouragement throughout the development of this project.

We also extend our appreciation to all those who directly or indirectly contributed to the successful completion of this work.

VIII. REFERENCES

- A. Fritz, T. Murphy, G. C. Murphy, and E. Hill, “Degree-of-knowledge: Modeling a developer’s knowledge of code,” ACM Transactions on Software

Engineering and Methodology, vol. 26, no. 1, pp. 1–42, 2017.

- [2] M. P. Robillard and R. DeLine, "A field study of API learning obstacles," Empirical Software Engineering, vol. 16, no. 6, pp. 703–732, 2011.
- [3] H. Husain et al., "CodeSearchNet Challenge: Evaluating the state of semantic code search," arXiv preprint arXiv:1909.09436, 2019.
- [4] Z. Feng et al., "CodeBERT: A pre-trained model for programming and natural languages," in Proc. EMNLP, 2020.
- M. Chen et al., "Evaluating large language models trained on code," arXiv preprint arXiv:2107.03374, 2021.
- OpenAI, "GPT-4 Technical Report," arXiv preprint arXiv:2303.08774, 2023.
- S. Nair, A. Kumar, and P. Gupta, "Conversational programming with large language models," in Proc. ICSE Workshops, 2023.
- J. Nam et al., "GILT: IDE-based prompt-less code understanding using LLMs," arXiv preprint, 2024.
- A. Kazemitabaar et al., "Evaluating AI-generated explanations for programming education," in Proc. CHI, 2021.
- S. MacNeil et al., "Experiences from using AI to support code understanding," in Proc. SIGCSE, 2022.
- Y. Zhang et al., "Challenges and opportunities of LLMs in software development," IEEE Software, vol. 40, no. 3,
- pp. 80–87, 2023.
- GitHub, "GitHub Copilot Documentation," 2023. [Online]. Available: https://github.com/features/copilot
- Microsoft, "Visual Studio Code Documentation," 2023. [Online]. Available: https://code.visualstudio.com/docs T. Brown et al., "Language models are few-shot learners," in Proc. NeurIPS, 2020.
- D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," arXiv preprint, 2014.
- J. Devlin et al., "BERT: Pre-training of deep bidirectional transformers," in Proc. NAACL, 2019.
- A. Vaswani et al., "Attention is all you need," in Proc. NeurIPS, 2017.
- P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Proc. NeurIPS, 2020.
- R. Svyatkovskiy et al., "IntelliCode Compose: Code generation using transformer models," in Proc. KDD, 2020.
- S. Iyer et al., "Learning a neural semantic parser from user feedback," ACL, 2017.
- K. Allamanis et al., "A survey of machine learning for big code and naturalness," ACM Computing Surveys, 2018.