OSSerCopilot: An LLM-Driven Tutoring System for Fostering Open Source Competency in Software Engineering Education

Xin Tan, Jingyi Tan, Weimiao Ren, Keqing Fan, Xiao Long, Fang Liu, Li Zhang

Abstract: In the context of large language model (LLM) reshaping software engineering education, this paper presents OSSerCopilot, a LLM-based tutoring system designed to address the critical challenge faced by newcomers (especially student contributors) in open source software (OSS) communities. Leveraging natural language processing, code semantic understanding, and learner profiling, the system functions as an intelligent tutor to scaffold three core competency domains: contribution guideline interpretation, project architecture comprehension, and personalized task matching. By transforming traditional onboarding barriers—such as complex contribution documentation and opaque project structures—into interactive learning journeys, OSSerCopilot enables newcomers to complete their first OSS contribution more easily and confidently. This paper highlights how LLM technologies can redefine software engineering education by bridging the gap between theoretical knowledge and practical OSS participation, offering implications for curriculum design, competency assessment, and sustainable OSS ecosystem cultivation. A demonstration video of the system is available at: https://figshare.com/articles/media/OSSerCopilot_Introduction_mp4/29510276.

Key words: Software Engineering Education; Open Source Software Education; Intelligent Tutoring Systems; Newcomer Onboarding; Large Language Models; AI-driven Educational Tools; OSS Contribution

1 Introduction

In recent years, the open source software (OSS) development model has emerged as a central driving force in the software industry. By breaking down traditional development barriers and aggregating global intelligence, it has significantly unleashed technological innovation and revitalized the software sector[4,8]. Currently, over 97% of developers and 99% of enterprises worldwide engage with OSS in some capacity[1]. In response to this growing trend, many countries have elevated the cultivation of OSS talents to national strategic importance[6]. Increasingly, OSS is being integrated into higher education curricula, particularly within software engineering programs, by establishing specialized courses.

These efforts aim not only to foster students' interest in OSS technologies but also to develop their practical competencies systematically. Embedding OSS practices into software engineering education is not merely a pedagogical enhancement, but also a strategic national imperative for cultivating high-level software talents.

However, despite the growing emphasis on OSS in software engineering education, a significant gap persists between the design of OSS-related curricula and their practical implementation. OSS projects are often inherently complex[9]. Due to limited instructional resources, instructors are typically unable to provide one-on-one guidance to every student throughout the course[11,13]. As a result, students are frequently expected to engage in self-directed learning by relying on resources provided by the OSS communities. However, for newcomers—particularly students lacking prior experience with real-world software projects—the OSS communities often presents a formidable "contribution barrier". Although many OSS communities have adopted beginner-friendly measures, students still frequently encounter a series of systemic barriers when making their first contributions[5]:

- Contribution guidelines are often lengthy, complicated, and written from a project management perspective, making it difficult for beginners to extract essential information quickly[2].

• Xin Tan, Xiao Long, Fang Liu* and Li Zhang are with the School of Computer Science and Technology, Beihang University, Beijing, 100191, China. E-mail: {xintan, longxiao, fangliu, lily}@buaa.edu.cn.

• Jingyi Tan, Weimiao Ren, Keqing Fan are with the School of Software, Beihang University, Beijing, 100191, China. E-mail: Tjy200408@126.com, 22373477@buaa.edu.cn, 1377426063@qq.com.

* To whom correspondence should be addressed.

Manuscript received: year-month-day; revised: year-month-day; accepted: year-month-day

- Many OSS projects—especially those with long development histories—possess large and intricate codebases, which make it challenging for students to form a coherent understanding of the overall system in a short period of time[14].
- Although communities often label certain tasks as “Good First Issues” (GFIs), these tasks frequently do not align well with the actual skill levels of newcomer contributors[12]. When students encounter difficulties that they cannot resolve on their own, they typically turn to community experts for assistance. However, such responses are often delayed or slow to arrive[10].

These barriers significantly dampen students’ interest and enthusiasm for learning OSS, hinder the progress of OSS-related instruction, and undermine the achievement of intended course objectives. More critically, they impede the influx of fresh contributors into the OSS ecosystem and contribute to a high attrition rate among potential contributors. Addressing the practical challenges of teaching OSS, lowering the barriers for student participation, and providing structured and personalized guidance have thus emerged as urgent and pivotal issues in contemporary software engineering education.

The rapid advancement of artificial intelligence technologies, particularly large language models (LLMs), has created unprecedented opportunities to address the aforementioned challenges in software engineering education. With their powerful capabilities in natural language understanding, code analysis, and generation, LLMs are now able to comprehend and process OSS resources—such as documentation and repository-level code—at a level comparable to, or even exceeding, that of human experts[7]. This positions LLMs as promising candidates for serving as intelligent teaching assistants. LLMs can dynamically learn individual learner profiles based on user information and conversational context, enabling precise analysis of differentiated learning needs and making personalized guidance a practical reality. Moreover, unlike traditional educational resources or delayed community support, LLMs offer real-time, interactive responses to learners’ queries, thereby overcoming both the latency of community-based assistance and the physical limitations of instructor availability. By incorporating “guided interaction” mechanisms, LLMs can systematically support students in navigating and exploring OSS projects, aligning closely with the pedagogical role of a mentor in educational settings. As such, LLMs demonstrate unique advantages in becoming powerful, responsive, adaptive, and continuously available tutors for OSS learning[11]. It lays a robust technical foundation and opens up new practical possibilities for building autonomous, efficient, and high-quality software engineering education environments.

This paper addresses the implementation challenges of OSS practices in software engineering education. Adopting a learner-centered perspective, we identify the core difficulties that students face when participating in OSS course activities. To tackle these issues, we designed and developed OSSerCopilot, a LLM-based tutoring system tailored for OSS learners. OSSerCopilot provides intelligent, end-to-end support throughout the learning process, effectively lowering the entry barriers for first-time contributors and systematically cultivating students’ core engineering skills. Our experiments demonstrate that OSSerCopilot not only enhances the efficiency of the teaching process but also advances the pedagogical goals of software engineering education. Moreover, it exemplifies how LLMs can serve as a transformative bridge between theory and practice, opening new directions for the evolution of OSS education ecosystems.

2 OSSerCopilot: Design Framework

OSSerCopilot is composed of three core functional modules: Contribution guideline analysis, Project structure analysis, and Issue recommendation. Figures 1 to 3 provide the interfaces of these three modules. In the following sections, we provide a detailed technical overview of each component.

2.1 Module 1: Contribution Guideline Analysis

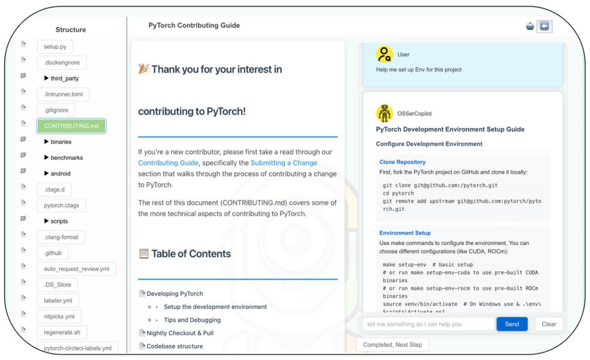
The goal of this module is to address the challenges posed by lengthy and complex contribution guidelines in OSS projects, which students often find difficult to understand. It helps learners quickly extract key information such as contribution requirements and environment setup procedures.

First, we apply regular expressions to match document filenames and retrieve critical documentation from the project repository, including files such as README.md and CONTRIBUTING.md. Next, we invoke the LLM (Qwen-Plus model. All modules in our system are powered by this model, except for Skill Matching in Module 3, which relies on the DeepSeek model for enhanced capability in structured skill extraction and mapping) to perform unsupervised clustering on the document contents, thereby identifying semantic labels such as "Project Overview", "Environment Setup", and other relevant sections. These labels are then combined with the original text to construct tailored prompts, enabling the LLM to generate concise and informative summaries of the contribution guidelines. To support further exploration, the module is designed to handle open-ended question answering. When students submit questions via the dialogue interface, the backend API integrates their queries into a predefined prompt template. This template embeds the previously extracted documents and section labels as contextual knowledge, ensuring more accurate and relevant responses from LLMs.

Moreover, recognizing that students—especially those lacking project experience—may struggle to formulate high-quality questions, we integrate a question suggestion mechanism into each module. Leveraging the LLM's context retention capabilities, we prompt the model to generate 3–5 follow-up questions at the end of each response, thereby guiding students toward deeper engagement with the project. For example, the following are recommended follow-up questions that the AI mentor might suggest after responding to a round of user queries:

- How can I verify that the environment is configured correctly? Is there integrated CI (e.g., GitHub Actions) to ensure environment consistency?
- Is it mandatory to pass all tests before submitting code?
- What is the contribution workflow defined by the project maintainers? Does it follow a Fork + PR model? Is there a required format for commit messages?
- Does the project provide a website, contributor meetings, discussion groups, or communication channels such as Slack or Discord?

Figure 1 provide the interface of this module.



🎉 Thank you for your interest in

contributing to PyTorch!

If you're a new contributor, please first take a read through our Contributing Guide, specifically the Submitting a Change section that walks through the process of contributing a change to PyTorch.

The rest of this document (CONTRIBUTING.md) covers some of the more technical aspects of contributing to PyTorch.

📋 Table of Contents

- Developing PyTorch
  - Setup the development environment
  - Tips and Debugging
- Nightly Checkout & Pull
- Codebase structure

Help me set up Env for this project

PyTorch Development Environment Setup Guide

Configure Development Environment

Clone Repository

First, fork the PyTorch project on GitHub and clone it locally:

```
git clone git@github.com:/pytorch.git
cd pytorch
git remote add upstream git@github.com:pytorch/pyto
rch.git
```

Environment Setup

Use make commands to configure the environment. You can choose different configurations (like CUDA, ROCm):

```
make setup-env  # basic setup
# or run make setup-env-cuda to use pre-built CUDA
binaries
# or run make setup-env-rocm to use pre-built ROCm
binaries
source venv/bin/activate  # On Windows use & .\env\
```

tell me something so I can help you.

Fig.1 OSSerCopilot's Interface: Contribution Guideline Analysis

2.2 Module 2: Project Structure Analysis

This module is designed to help students understand both the high-level architecture and the low-level implementation of large-scale software systems. It aims to reduce the cognitive overload often experienced when navigating massive codebases and to foster students' abilities in code reading and analysis.

Considering that students may approach OSS projects with dual needs—top-down exploration of system architecture and bottom-up inspection of implementation details—we introduce a layered analysis approach to support both perspectives:

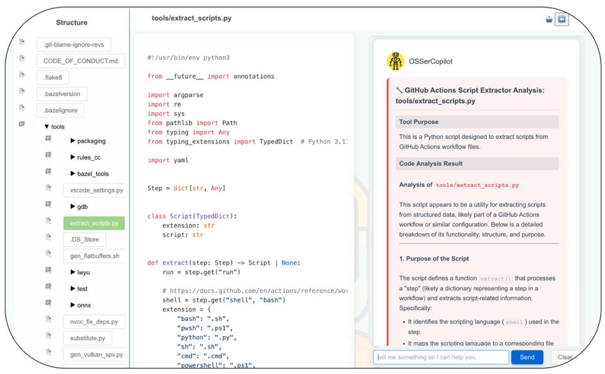
2.2.1 Code Decomposition

We utilize language-specific parsers (e.g., Python's ast library) to perform syntax analysis on all code files within a project, generating their corresponding Abstract Syntax Trees (ASTs). A depth-first traversal is then conducted over the ASTs to extract the smallest semantically meaningful code units, such as functions, classes, and interfaces. For each unit, we extract its signature, associated comments, and code body, thereby enabling fine-grained code decomposition.

2.2.2 Bottom-up Recursive Summarization Algorithm

This algorithm constructs hierarchical summaries of a codebase through a recursive, bottom-up process. Starting with the smallest logical units, it progressively synthesizes higher-level summaries by aggregating and abstracting lower-level explanations, culminating in a comprehensive repository-level overview.

Figure 2 provide the interface of this module.



```
#!/usr/bin/env python3

from __future__ import annotations

import argparse
import re
import sys
from pathlib import Path
from typing import Any
from typing_extensions import TypedDict  # Python 3.1

import yaml


Step = dict[str, Any]


class Script(TypedDict):
    extension: str
    script: str


def extract(step: Step) -> Script | None:
    run = step.get("run")

    # https://docs.github.com/en/actions/reference/wo
    shell = step.get("shell", "bash")
    extension = {
        "bash": ".sh",
        "pwsh": ".ps1",
        "python": ".py",
        "sh": ".sh",
        "cmd": ".cmd",
        "powershell": ".ps1",
```

🔧 GitHub Actions Script Extractor Analysis: tools/extract_scripts.py

Tool Purpose

This is a Python script designed to extract scripts from GitHub Actions workflow files.

Code Analysis Result

Analysis of `tools/extract_scripts.py`

This script appears to be a utility for extracting scripts from structured data, likely part of a GitHub Actions workflow or similar configuration. Below is a detailed breakdown of its functionality, structure, and purpose.

1. Purpose of the Script

The script defines a function `extract()` that processes a "step" (likely a dictionary representing a step in a workflow) and extracts script-related information. Specifically:

- It identifies the scripting language (`shell`) used in the step.
- It maps the scripting language to a corresponding file

tell me something so I can help you.

Fig.2 OSSerCopilot's Interface: Project Structure Analysis

Unit-Level Analysis (L0):

For each type of logical unit, we design specialized explanation-oriented prompts that focus on its role and functionality within the code. We employed a few-shot prompting strategy to guide the LLM in generating concise summaries for these basic code units.

File-, Package-, and Repository-Level Summarization (L1):

At each higher level of abstraction, we aggregate the summaries generated from all lower-level elements within that scope. These aggregated texts are then provided to the LLM as contextual input, prompting it to produce a higher-level summary. This process is applied recursively in a bottom-up manner, culminating in a global summarization of the entire repository at the root directory level. All intermediate summaries are stored in *JSON* format, organized in a directory structure that mirrors the project's original file hierarchy.

Finally, students can interact with a visual file tree interface, where they can access analysis summaries for any file or package with a single click, and freely ask questions about the code contained within. The "Project Structure Analysis" module thus provides students with a scalable "cognitive map" of the codebase, enabling flexible navigation and deep understanding of complex software systems.

2.3 Module 3: Issue Recommendation

This module is designed to address the mismatch between "Good First Issues" and the actual interests and capabilities of novice contributors. We introduce a user profiling mechanism to enable more accurate task matching.

2.3.1 User Profiling

This module utilizes GitHub's official API endpoint "*GET /users/{username}/events*" to retrieve a user's public event stream. These events include 12 core types, such as "*PushEvent*" (code commits), "*PullRequestEvent*" (PR submissions), and "*IssueCommentEvent*" (comments on issues), which collectively reflect the user's historical contribution activity.

We define a six-dimensional keyword schema to represent user skills, covering areas such as "Programming Languages", "Frameworks & Libraries", and "Development Practices". Leveraging the LLM, we semantically analyze the user's event records and match them against this keyword schema to infer skill tags. The resulting structured user profile is represented as a *JSON* object and serves as the foundation for personalized task recommendations.

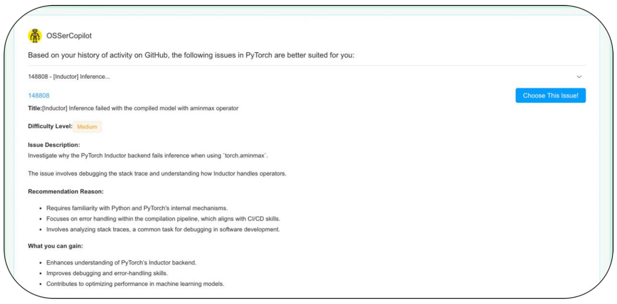
2.3.2 Issue Matching Based on User Profiles

The backend constructs HTTP requests to retrieve issue data from specific GitHub repositories, using filters such as "state=open" and "labels=good first issue". The titles and descriptions of the issues are then normalized and preprocessed through data cleaning and visualization steps. Both the previously generated user profile and the processed issue descriptions are passed to the LLM, which is prompted to recommend issues that align with the user's skills and interests.

A set of parsing rules is applied to extract structured information from the LLM's response, resulting in a recommendation list that includes attributes such as issue ID, URL, difficulty level, and justification for the recommendation. The front end displays these results in a card-based layout, allowing users to view details and navigate to the corresponding GitHub issue with a single click.

This module embodies OSSerCopilot's core philosophy of personalized learning. By deriving precise technical tags from students' GitHub histories and leveraging LLMs for semantic reasoning, the system recommends tasks matching both skill level and interest. This enables learners to engage in OSS practice with clear goals and appropriate challenges, fostering competence and significantly boosting intrinsic motivation to contribute.

Figure 3 provide the interface of this module.



Based on your history of activity on GitHub, the following issues in PyTorch are better suited for you:

148808

Title:[Inductor] Inference failed with the compiled model with aminmax operator

Difficulty Level: Medium

Issue Description:

Investigate why the PyTorch Inductor backend fails inference when using `torch.aminmax`.

The issue involves debugging the stack trace and understanding how Inductor handles operators.

Recommendation Reason:

- Requires familiarity with Python and PyTorch's internal mechanisms.
- Focuses on error handling within the compilation pipeline, which aligns with CI/CD skills.
- Involves analyzing stack traces, a common task for debugging in software development.

What you can gain:

- Enhances understanding of PyTorch's Inductor backend.
- Improves debugging and error-handling skills.
- Contributes to optimizing performance in machine learning models.

Fig.3 OSSerCopilot's Interface: Issue Recommendation

2.4 System Architecture

The system adopts a front-end-back-end separation architecture, which is efficient and easily extensible. Figure 4 illustrates the system architecture of OSSerCopilot.

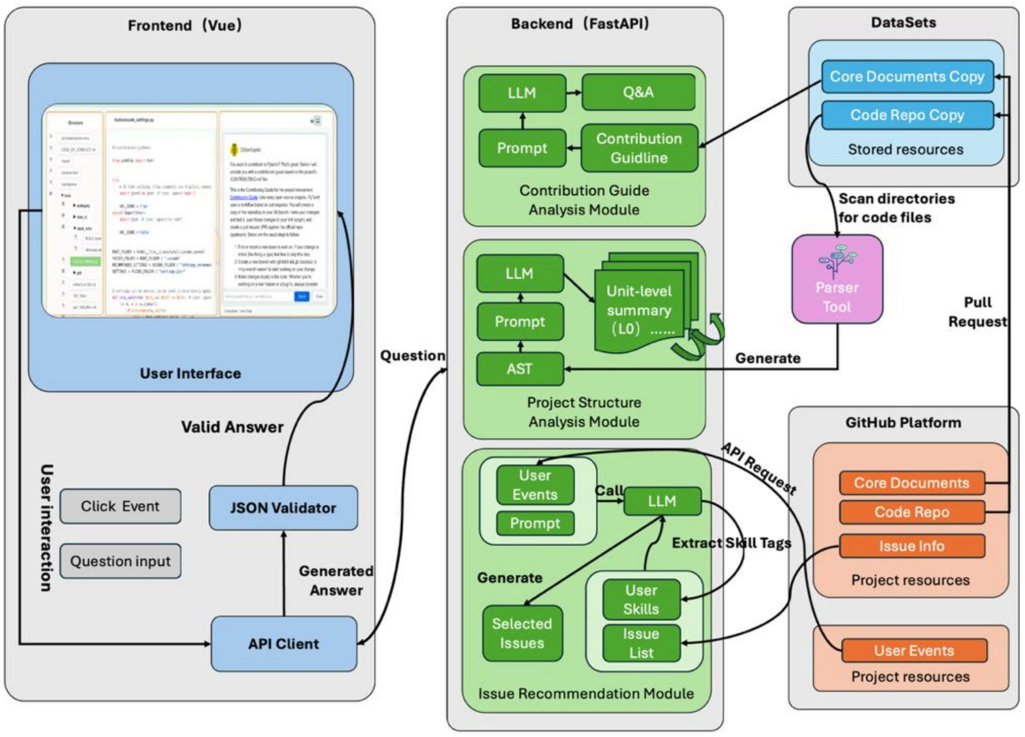


Fig.4 Framework of OSSerCopilot

The front-end uses the Vue framework, leveraging its component-based and reactive design to enable users to interact with the tool through an intuitive interface. User actions, such as submitting questions or clicking on the file tree, are detected by the front-end and sent to the back-end via API calls. Upon receiving *JSON* data responses from the back-end, the front-end is responsible for parsing the data and rendering the results on the user interface.

The back-end is implemented using the FastAPI framework, which is primarily responsible for handling user requests, executing algorithms, constructing high-quality prompts, and interfacing with the LLM.

3 Evaluation

To validate the effectiveness of OSSerCopilot as an OSS tutoring system, we conducted a learner-centered empirical study. The evaluation focused on assessing the actual impact of OSSerCopilot's design on learners in real-world learning scenarios.

3.1 Evaluation Experimental Methodology

The participation of novice OSS learners serves as a crucial criterion for evaluating the success of OSSerCopilot's design. To this end, we recruited 19 OSS beginners for the study. Notably, 10 of these participants were students enrolled in a university course on OSS development, ensuring a high degree of relevance to real-world software engineering education scenarios.

A mixed-methods approach was employed for evaluation. Prior to system development, we conducted interviews and surveys to investigate the difficulties novices encountered at various stages of OSS contribution, as well as their expectations for an AI tutor. These insights directly informed our feature design.

After participants experienced the OSSerCopilot prototype to complete a simulated contribution task, feedback was collected through multiple means: satisfaction ratings on the system's three core modules were gathered using a 5-point Likert scale and compared against their pre-development expectations. Additionally, the classic Technology Acceptance Model (TAM)[3] was employed to quantitatively assess the system across three dimensions: Perceived Usefulness, Perceived Ease of Use, and Self-predicted Future Use.

3.2 Evaluation Results and Findings

The study results strongly validate the effectiveness of OSSerCopilot in the context of OSS education.

As shown in Figure 5, participants' satisfaction scores for the three core functions of OSSerCopilot closely matched their prior expectations, indicating that our design accurately addressed key challenging points in teaching OSS projects.

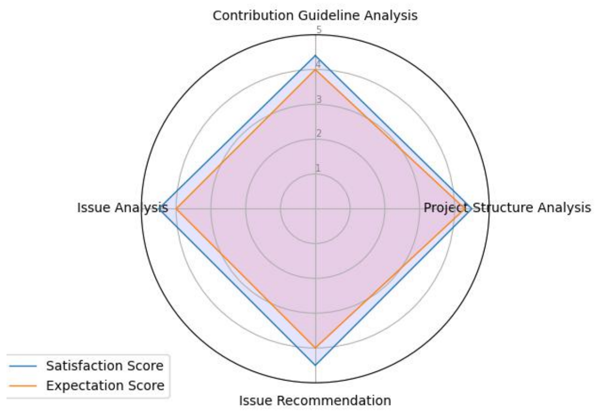


Fig.5 Participants' Average Likert Scale Ratings on Expectations and Satisfaction with OSSerCopilot

In the dimension of Perceived Usefulness (as shown in Figure 6), all metrics received over 88% of "Agree" or "Strongly Agree" responses. All participants reported that OSSerCopilot made their contribution tasks "faster" and helped "improve work performance and efficiency," demonstrating that the tool effectively enhances students' learning efficiency and practical outcomes.

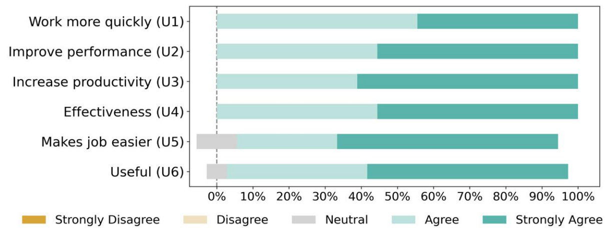


Fig.6 Perceived Usefulness

As shown in Figure 7, more than 83% of participants found OSSerCopilot "easy to learn and operate," a critical factor for educational tools, as it ensures students can focus cognitive resources on learning OSS knowledge itself rather than struggling with tool usage.

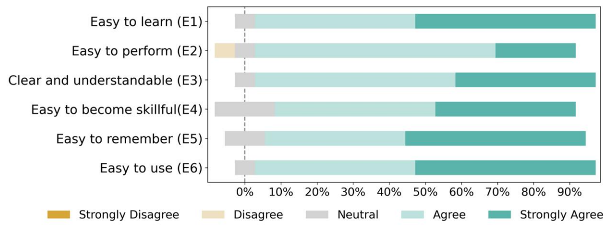


Fig.7 Perceived Ease of Use

Finally, as shown in figure 8, all participants expressed willingness to use OSSerCopilot in future OSS practice.

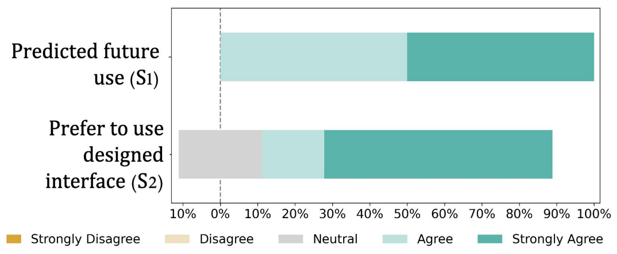


Fig.8 Self-predicted Future Use

More convincingly, 78% of participants believed that compared to traditional approaches (such as independently reading documentation or waiting for community expert responses), OSSerCopilot provides "a more effective form of support." This result directly reflects the superiority of an AI tutor over conventional learning resources.

In summary, the positive feedback from students confirms that OSSerCopilot, as an LLM-driven tutoring tool for OSS onboarding, effectively lowers the learning curve, improves the learning experience and confidence, and successfully serves as a "digital guide" for learners.

4 Implications for Software Engineering Education

In prior software engineering education practices, the high entry barriers that students face when engaging with OSS projects have often been overlooked or inadequately addressed. Traditional instructional approaches are constrained by limited resources and time, making it difficult to provide timely, detailed, and personalized one-on-one guidance. These barriers have hindered a large number of newcomers from contributing to OSS and have resulted in significant talent attrition. The successful implementation of OSSerCopilot demonstrates the immense potential of LLMs to overcome these traditional limitations and to transform software engineering education, particularly in reshaping the training model for OSS contributors.

The most direct contribution of OSSerCopilot lies in its effective response to the key pain points faced by novices during their first OSS contribution. It transforms traditional entry barriers — mismatching issue recommendations and the difficulty of understanding intricate project structures—into an efficient and engaging interactive learning journey. Leveraging the powerful natural language understanding and generation capabilities of LLMs, OSSerCopilot accurately interprets contribution guidelines, provides a clear analysis of project architecture, and recommends personalized tasks based on students' skills and interests.

This real-time, one-on-one assistance significantly lowers both the cognitive and technical thresholds for student participation, enabling them to complete their first OSS contributions successfully. This improvement, in turn, enhances their sense of achievement, helping to prevent the loss of motivation and talent that often results from a difficult onboarding experience. By attracting more learners into the OSS ecosystem and offering structured guidance that eases the burden on community maintainers, OSSerCopilot contributes to the development of a healthier, more vibrant, and sustainable OSS ecosystem.

LLM technologies offer an effective means to bridge the significant gap between theoretical knowledge and real-world OSS participation. Within OSSerCopilot, LLMs are grounded in authentic project contexts, helping learners connect abstract engineering and disciplinary concepts with concrete code implementations and engineering practices in real OSS projects. This improvement addresses the long-standing disconnect between theory and practice in software engineering education, fosters students' OSS competencies, and contributes to a redefinition of how software engineering is taught.

The successful application of OSSerCopilot provides valuable insights for future curriculum design, competency assessment, and talent development in software engineering education. As a newly introduced instructional resource, AI tutors offer scalable, context-aware, and hands-on support for practice-based teaching. In terms of assessment, OSSerCopilot enables educators to leverage detailed student activity data to build a more holistic and process-oriented evaluation system—one that assesses learners' engineering literacy based on the quality and depth of their actual contributions to real-world projects.

In conclusion, the exploration and practice of OSSerCopilot confirm the significant potential of LLMs in enhancing software engineering education, fostering OSS talents, and bridging the gap between theory and practice. It offers valuable insights into how software engineering education can be reimagined and restructured in the era of AI.

5 Conclusion

This study addresses a critical challenge in current software engineering education—teaching OSS practices—and explores the application potential of LLMs in this context. Guided by this vision, we designed OSSerCopilot, a prototype system designed to cultivate students' OSS competencies. The system targets three core challenges faced by OSS novices: understanding the contribution process, comprehending project architecture, and identifying suitable tasks. It introduces an intelligent tutoring system that integrates natural language understanding, semantic analysis, and user profiling. Empirical results demonstrate that OSSerCopilot significantly enhances the learning experience, lowers entry barriers, and supports effective competency development.

From a practical standpoint, OSSerCopilot not only improves students' efficiency and confidence in completing their first OSS contribution but also significantly enhances their understanding of large-scale software architectures and their engineering thinking abilities. These outcomes provide strong evidence for the feasibility and necessity of AI tutors in delivering structured, continuous support within educational settings.

As the capabilities of LLMs continue to evolve, OSSerCopilot holds promise for further extending its application scope. On the functional level, the system can be enhanced with improved code generation and refactoring suggestions, as well as expanded support for advanced OSS tasks such as collaborative development and code review. On the pedagogical level, OSSerCopilot can be more deeply integrated into project-based training, course design, and academic assessment frameworks, enabling instructors to deliver personalized teaching interventions and evaluate learning outcomes. These directions pave the way for scalable, high-quality cultivation of OSS competencies.

In summary, the OSSerCopilot project offers a practical solution for cultivating OSS competencies in software engineering education. It provides a valuable case study for exploring AI-powered transformations in educational paradigms. Looking ahead, we envision broader educational scenarios in which LLM-driven "AI tutors" can guide learners into the complex world of real-world systems, enabling deeper capability development and more meaningful educational outcomes.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant Nos. 62202022, 62572030), the Fundamental Research Funds for the Central Universities, and the exploratory elective projects of the State Key Laboratory of Complex and Critical Software Environments.

References

[1] Andersen-Gott M, Ghinea G G, Bygstad B. Why do commercial companies contribute to open source software?[J]. International journal of of information management, 2012(2): 106-117.
[2] Elazhary O, Storey M A, Ernst N, et al. Do as i do, not as i say: Do contribution guidelines match the github contribution process?[C]// 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2019: 286-290.
[3] Granić A, Marangunić N. Technology acceptance model in educational context: A systematic literature review[J]. British Journal of Educational Technology, 2019(5): 2572-2593.
[4] Levine S S, Prietula M J. Open collaboration for innovation: Principles and performance[J]. Organization Science, 2014(5): 1414-1433.
[5] Mendez C, Padala H S, Steine-Hanson Z, et al. Open source barriers to entry, revisited: A sociotechnical perspective[C]// Proceedings of the 40th International conference on software engineering. 2018: 1004-1015.
[6] Ministry of Industry and Information Technology of China. 14th Five-Year Plan for Software and Information Technology Service Industry Development[EB/OL]. (2021-03-13)[2025-10-09]. https: // www.gov.cn/xinwen/2021-03/13/content_5592681.htm.
[7] Nam D, Macvean A, Hellendoorn V, et al. Using an llm to help with code understanding[C]// Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. 2024: 1-13.
[8] Scacchi W. Free/open source software development[C]// Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. 2007: 459-468.
[9] Steinmacher I, Silva M A G, Gerosa M A. Barriers faced by newcomers to open source projects: a systematic review[C]// Open Source Software: Mobile Open Source Technologies: 10th IFIP WG 2.13 International Conference on Open Source Systems, OSS 2014, San José, Costa Rica, May 6-9, 2014. Springer, 2014: 153-163.
[10] Tan X, Chen Y, Wu H, et al. Is it enough to recommend tasks to newcomers? understanding mentoring on good first issues[C]// 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). IEEE, 2023: 653-664.
[11] Tan X, Long X, Zhu Y. Revolutionizing Newcomers' Onboarding Process in OSS Communities: The Future AI Mentor[J]. Proceedings of the ACM on Software Engineering, 2025(FSE): 1091-1113.
[12] Tan X, Zhou M, Sun Z. A first look at good first issues on GitHub[C]// Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2020: 398-409.
[13] Tan X, Zhou M, Zhang L. Understanding mentors' engagement in OSS communities via google summer of code[J]. IEEE Transactions on Software Engineering, 2023(5): 3106-3130.
[14] Tu Q, et al. Evolution in open source software: A case study[C]// Proceedings 2000 International Conference on Software Maintenance. IEEE, 2000: 131-142.

Xin Tan obtained her PhD. from Peking University in 2021; She is currently an Associate Professor at the School of Computer Science, Beihang University. Her major research interests focus on empirical software engineering, open source software development, and human-computer interaction. She has published more than 30 papers in top journals and conferences in the fields of software engineering and human-computer interaction, including ICSE, FSE, ASE, CSCW, TSE, and TOSEM. She has won the First Prize of Beijing Science and Technology Progress Award and the ACM/IEEE Distinguished Paper Award, and has been selected into the Beijing "High-Innovation Plan" Young Talent Support Project.

Weimiao Ren is pursuing a Bachelor's degree in Software Engineering at Beihang University. Her research centers on multimodal interaction in Extended Reality (XR) and AI-driven software systems. She has developed an end-to-end MR object interaction system, contributed to an AI-powered open-source mentorship platform, and engineered a high-traffic e-commerce platform. She aspires to further integrate large models into intelligent development tools and advanced interactive paradigms.

Jingyi Tan is currently pursuing her Bachelor's degree in the School of Software, Beihang University. Majoring in software engineering, her academic and practical work primarily focuses on exploring the application of Artificial Intelligence (AI) within the software industry, with a strong emphasis on Large Language Models (LLMs). Specifically, her experience involves developing education-assisted AI tools and AI-driven unit test code generation tools.

Keqing Fan is currently pursuing her Bachelor's degree in Software Engineering at Beihang University, China. Her current research mainly focuses on intelligent software engineering, particularly in intelligent requirements acquisition and analysis. She is interested in applying artificial intelligence techniques to improve software development processes and hopes to continue exploring advanced methods in intelligent software systems in the future.

Xiao Long is a postgraduate student at the School of Computer Science and Engineering, Beihang University. Before that, he received a B.S. degree from the Shen Yuan Honors College, Beihang University in 2023. His research interests include empirical software engineering and open source ecosystems. His current research focuses on designing AI Agent to promote OSS newcomers' onboarding.

Fang Liu received the Ph.D. degree from the School of Computer Science, Peking University, Beijing, China, in 2022. She is an Assistant Professor with the School of Computer Science and Engineering, Beihang University. Her research interests mainly focus on the fields of intelligent software engineering, including code understanding, code generation, program repair, etc. Her current research focuses on advancing the application and optimization of large language models (LLMs) for software engineering tasks, particularly in code-related domains such as code generation, program repair, program translation,and code review.

Li Zhang received the B.S., M.S., and Ph.D. degrees from the School of Computer Science and Engineering, Beihang University, Beijing, China, in 1989, 1992, and 1996, respectively. She is currently a Professor with the School of Computer Science and Engineering, Beihang University, where she is leading the expertise areas of system and software modeling. She has around 20 years of experience of conducting industry oriented research in various application domains such as avionics and ships, and communications in several countries, including America, Norway, and France. Her current research interests include software engineering, with specific interest in requirements engineering, software/system architectures, model-based engineering, model-based product line engineering, and empirical software engineering.