DebugTA: An LLM-Based Agent for Simplifying Debugging and Teaching in Programming Education

Extended Abstract

Lingyue Fu
Shanghai Jiao Tong University
Shanghai, China
fulingyue@sjtu.edu.cn

Datong Chen
Fudan University
Shanghai, China
dtchen22@m.fudan.edu.cn

Haowei Yuan
Stevens Institute of Technology
Hoboken, NJ, USA
hyuan13@stevens.edu

Xinyi Dai
Tencent
Shanghai, China
xinyidai@tencent.com

Qingyao Li
Shanghai Jiao Tong University
Shanghai, China
ly890306@sjtu.edu.cn

Weinan Zhang
Shanghai Jiao Tong University
Shanghai, China
wnzhang@sjtu.edu.cn

Weiwen Liu*
Shanghai Jiao Tong University
Shanghai, China
wwliu@sjtu.edu.cn

Yong Yu*
Shanghai Jiao Tong University
Shanghai, China
yyu@sjtu.edu.cn

ABSTRACT

In programming education, Debugging and Teaching (DT) task is a common scenario which requires generating modification suggestions from erroneous code, error messages, reference solutions, and problem descriptions. Existing approaches struggle with complex multi-source reasoning and underutilize available reference code, limiting the effectiveness of large language models (LLMs) in DT tasks. To address these challenges, we propose DebugTA, a novel LLM-based debugging and teaching agent with specialized tools for standard code retrieval, variable substitution to align reference code, and an external compiler for real-time code analysis. Guided by pedagogical and debugging principles, DebugTA decomposes complex DT tasks into structured LLM–tool interactions that reduce reasoning complexity. By aligning reference code with erroneous code, DebugTA enables the LLM to focus on logical errors and improves suggestion accuracy. To rigorously assess the quality of modification suggestions, we introduce a student simulator-teacher interaction paradigm. Experimental results on three real-world code datasets demonstrate that DebugTA consistently improves teaching effectiveness while significantly reducing computational costs.

KEYWORDS

Debugging and Teaching; LLM Agents; Programming Education

ACM Reference Format:
Lingyue Fu, Datong Chen, Haowei Yuan, Xinyi Dai, Qingyao Li, Weinan Zhang, Weiwen Liu*, and Yong Yu*. 2026. DebugTA: An LLM-Based Agent for Simplifying Debugging and Teaching in Programming Education: Extended Abstract. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 3 pages. https://doi.org/10.65109/GYMB4283

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/GYMB4283

1 INTRODUCTION

In programming education, the Debugging and Teaching (DT) task goes beyond merely fixing code; it involves identifying logic errors and providing pedagogical guidance to help students correct their work [2, 10]. While Large Language Models (LLMs) [6] have revolutionized code generation, they still face significant challenges in DT scenarios. Direct debugging with LLMs is challenged by the complexity and heterogeneity of multi-source inputs, leading to ineffective feedback [4, 11, 12].

A key missed opportunity in existing approaches is the underutilization of *Standard Code* (reference solutions), which are readily available in educational contexts (e.g., LeetCode, Codeforces). Directly providing standard code to the LLM can lead to answer leakage or confusion due to variable naming mismatches.

To address these challenges, we propose **DebugTA**, an LLM agent that decomposes the DT task into manageable steps. Our core contributions are:

- We propose DebugTA, the first LLM agent tailored for DT tasks that effectively utilizes reference solutions.
- We introduce a novel variable substitution tool that aligns standard code variables with student code, reducing reasoning burden while preventing plagiarism.
- We design a Student Simulator evaluation paradigm to rigorously assess teaching effectiveness.

A full technical version of this paper is available on arXiv[1].

2 DEBUGTA FRAMEWORK

DebugTA operates as an intelligent agent equipped with a memory module and three specialized tools: *Standard Code Retrieval*, *Variable Substitution*, and a *Compiler*.

[1] https://arxiv.org/abs/2510.11076

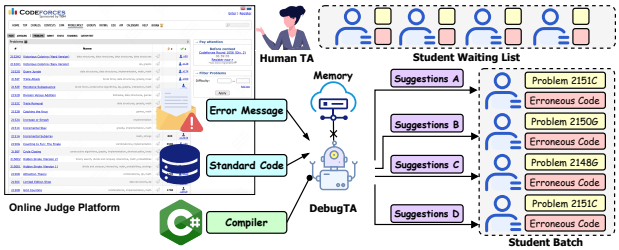

Figure 1: Demonstration of the Debugging and Teaching task.

2.1 Standard Code Retrieval

Given a student's erroneous submission, DebugTA first retrieves the most relevant correct solution from a repository. We utilize the BM25 algorithm to choose most similar solution based on structural and textual similarity, providing DebugTA with a reference.

2.2 Variable Substitution Mechanism

A major challenge in using reference code is the discrepancy in variable naming conventions between the student's code and the standard solution. Direct comparison confuses the LLM. To solve this, we develop the **Variable Substitution** tool. It employs an LLM-based aligner to map variables from the standard code to the student's code without altering the underlying logic. This aligned code serves as a perfect hint for the backbone model, allowing it to compare logic flows directly without the noise of mismatched identifiers. Crucially, this alignment process ensures that the agent can guide the student based on their own code structure.

2.3 Adaptive Reasoning & Memory

DebugTA follows a stepwise reasoning process. A memory module (implemented as a dictionary) stores the retrieval results, compiler feedback, and intermediate reasoning steps. This allows the agent to maintain context across multiple turns of interaction, selectively retrieving information to avoid context overflow.

3 EXPERIMENTS

3.1 Experimental Setup

We evaluated DebugTA on three datasets: CodeApex [5], ACMOJ[2], and Code4Bench [8], covering varying difficulty levels. We utilize three backbone models: GPT-4o-mini (API) [9], Qwen2.5-Coder-Instruct (7B) [7], and DeepSeek-Coder-V2-Lite-Instruct (16B) [3].

To simulate the interactive teaching process, we introduce a Student Simulator (StuBot). StuBot acts as a student who receives suggestions from DebugTA and attempts to fix their code. We measure the AC@1 of the code fixed by StuBot.

3.2 Results

To rigorously evaluate the effectiveness of our framework, we compared DebugTA against five baselines categorized into three groups: (1) **Direct Prompting** methods (Direct Debugging and Direct Teaching), which rely solely on the LLM's internal reasoning; (2) **Debug with Standard Code**, which directly provides the reference solution to the LLM; and (3) **Self-Debugging** methods (Explanation and Trace) [1], which guide the model to explain code or trace execution. As shown in Table 1, DebugTA significantly outperforms baseline methods across all datasets. Notably, DebugTA narrows the performance gap between smaller models and larger models, suggesting that our tool-use framework effectively offloads the reasoning burden from the backbone model.

In Figure 2, our ablation studies further confirm that removing either the retrieval mechanism or the variable alignment module significantly impacts DebugTA's performance. Providing LLMs with structurally and semantically similar correct code substantially reduces the complexity of the DT task.

[2]https://acm.sjtu.edu.cn/OnlineJudge/

Table 1: Results of DebugTA and baselines on three datasets with three backbone LLMs.

Backbone	Teacher	CodeApex	ACMOJ	Code4Bench
GPT-4o-mini	Direct Debug	78.70	12.00	18.39
	Debug with S	75.93	14.00	10.34
	Self Debug (Expl.)	71.30	16.00	10.34
	Self Debug (Trac.)	83.33	18.00	14.94
	Direct Teach	68.52	14.00	11.49
	DebugTA	**94.44**	**42.00**	**26.44**
Qwen (7B)	Direct Debug	65.74	7.00	10.34
	Debug with S	61.11	4.00	6.90
	Self Debug (Expl.)	58.33	6.00	2.30
	Self Debug (Trac.)	65.74	3.00	3.45
	Direct Teach	70.37	12.00	14.94
	DebugTA	**86.11**	**25.00**	**18.39**
DeepSeek (16B)	Direct Debug	70.37	13.00	6.90
	Debug with S	46.30	4.00	10.34
	Self Debug (Expl.)	59.26	18.00	8.05
	Self Debug (Trac.)	65.74	13.00	9.20
	Direct Teach	61.11	11.00	12.64
	DebugTA	**90.74**	**24.00**	**20.69**

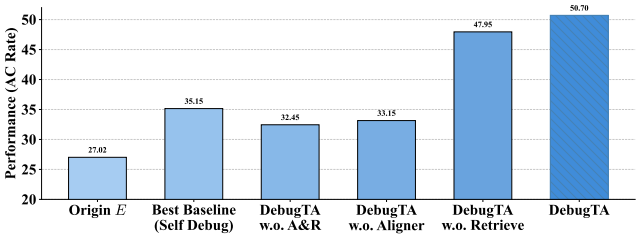

Figure 2: Ablation Study Results on the ACMOJ.

4 CONCLUSION

In this work, we proposed **DebugTA**, a LLM agent tailored for the debugging and teaching task. By innovatively combining standard code retrieval with a variable substitution mechanism, DebugTA effectively leverages reference solutions to provide accurate guidance without answer leakage. Experimental results demonstrate that DebugTA achieves state-of-the-art performance and significantly reduces the dependency on the backbone model's reasoning capabilities.

ACKNOWLEDGMENTS

This work is partially supported by National Natural Science Foundation of China (62322603, 62177033, 62502310).

REFERENCES

[1] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching Large Language Models to Self-Debug. arXiv:2304.05128 [cs.CL] https://arxiv.org/abs/2304.05128

[2] S. Choi and H. Kim. 2024. The impact of a large language model-based programming learning environment on students' motivation and programming ability. *Education and Information Technologies* (2024). https://doi.org/10.1007/s10639-024-13107-x

[3] DeepSeek-AI, Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y. Wu, Yukun Li, Huazuo Gao, Shirong Ma, Wangding Zeng, Xiao Bi, Zihui Gu, Hanwei Xu, Damai Dai, Kai Dong, Liyue Zhang, Yishi Piao, Zhibin Gou, Zhenda Xie, Zhewen Hao, Bingxuan Wang, Junxiao Song, Deli Chen, Xin Xie, Kang Guan, Yuxiang You, Aixin Liu, Qiushi Du, Wenjun Gao, Xuan Lu, Qinyu Chen, Yaohui Wang, Chengqi Deng, Jiashi Li, Chenggang Zhao, Chong Ruan, Fuli Luo, and Wenfeng Liang. 2024. DeepSeek-Coder-V2: Breaking the Barrier of Closed-Source Models in Code Intelligence. arXiv:2406.11931 [cs.SE] https://arxiv.org/abs/2406.11931

[4] Tuan Dinh, Jinman Zhao, Samson Tan, Renato Negrinho, Leonard Lausen, Sheng Zha, and George Karypis. 2023. Large language models of code fail at completing code with potential bugs. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 1794, 27 pages.

[5] Lingyue Fu, Huacan Chai, Shuang Luo, Kounianhua Du, Weiming Zhang, Longteng Fan, Jiayi Lei, Renting Rui, Jianghao Lin, Yuchen Fang, Yifan Liu, Jingkuan Wang, Siyuan Qi, Kangning Zhang, Weinan Zhang, and Yong Yu. 2024. CodeApex: A Bilingual Programming Evaluation Benchmark for Large Language Models. arXiv:2309.01940 [cs.CL] https://arxiv.org/abs/2309.01940

[6] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming – The Rise of Code Intelligence. arXiv:2401.14196 [cs.SE] https://arxiv.org/abs/2401.14196

[7] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. 2024. Qwen2.5-Coder Technical Report. arXiv:2409.12186 [cs.CL] https://arxiv.org/abs/2409.12186

[8] Amirabbas Majd, Mojtaba Vahidi-Asl, Alireza Khalilian, Ahmad Baraani-Dastjerdi, and Bahman Zamani. 2019. Code4Bench: A multidimensional benchmark of Codeforces data for different program analysis techniques. *Journal of Computer Languages* 53 (2019), 38–52. https://doi.org/10.1016/j.cola.2019.03.006

[9] OpenAI. 2024. GPT-4o-mini. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/. Accessed: 2024-07-18.

[10] M. Yousef, K. Mohamed, W. Medhat, et al. 2025. BeGrading: large language models for enhanced feedback in programming education. *Neural Computing and Applications* 37 (Jan. 2025), 1027–1040. https://doi.org/10.1007/s00521-024-10449-y

[11] Ruochen Zhang, Samuel Cahyawijaya, Jan Christian Blaise Cruz, Genta Indra Winata, and Alham Fikri Aji. 2023. Multilingual Large Language Models Are Not (Yet) Code-Switchers. arXiv:2305.14235 [cs.CL] https://arxiv.org/abs/2305.14235

[12] Wenbo Zhang, Aditya Majumdar, and Amulya Yadav. 2024. Code-mixed LLM: Improve Large Language Models' Capability to Handle Code-Mixing through Reinforcement Learning from AI Feedback. arXiv:2411.09073 [cs.CL] https://arxiv.org/abs/2411.09073