A Guided Multi-Agent Tutor for Progressive Help in Programming Practice

Sergio Arjona-Giner✉[0009−0003−9182−9518], Faraón Llorens-Largo[0000−0002−2117−0784], Javier García-Sigüenza[0000−0002−4456−4348], and Rosana Satorre-Cuerda[0000−0002−3380−9776]

Smart Learning Research Group, Department of Computer Science and Artificial Intelligence, University of Alicante, Alicante, Spain
{sergio.arjona,Faraon.Llorens,javierg.siguenza,rosana.satorre}@ua.es

Abstract. Large Language Models (LLMs) are increasingly used as tutoring assistants in programming education, but their pedagogical value depends not only on response accuracy but also on how much learning help they provide. In particular, LLM-based tutors may over-help students by revealing solution steps too early or by failing to adapt their support to the learner's current needs. This paper presents a guided multi-agent tutor for progressive help in programming practice. The proposed architecture separates student-state analysis, response generation, and response validation. It also uses an explicit activity characterization to distinguish between explainable concepts and protected solution steps, enabling finer control over solution disclosure. To evaluate the approach, we compare three configurations: a monolithic Socratic baseline, a guided multi-agent tutor without validation, and a full guided multi-agent tutor with validation. Nine computer science teachers evaluated responses generated for introductory Python programming scenarios using three dimensions: pedagogical quality, gradual support, and over-helping. Results show that both guided configurations were perceived as more gradual and less prone to over-helping than the monolithic baseline, while the full architecture achieved the most favorable mean values overall.

Keywords: Large language models · Programming education · Multi-agent systems · Intelligent tutoring systems

1 Introduction

Large Language Models (LLMs) are being rapidly integrated into computer science education [5, 6], particularly as interactive tutors capable of complementing human teachers. Given their natural language processing and reasoning capabilities, they can explain concepts, provide immediate feedback, and generate personalized responses. In the context of programming education, LLMs are especially attractive because they can identify programming errors, explain problem statements, and offer step-by-step assistance as students develop their skills [3]. However, their role in educational programming practice should be distinguished from their role as productivity tools for professional developers. While

professional coding assistants are usually expected to accelerate implementation and provide direct solutions, tutoring systems for novice programmers should support learning, preserve productive effort, and avoid replacing the learner's own reasoning process.

This distinction is important because the usefulness of educational assistants is not determined solely by the accuracy of their responses; rather, the quality of tutoring depends on whether the system's response is pedagogically appropriate, avoiding one of the major problems in LLM-based tutoring: over-helping [12, 11]. This issue is particularly important in guided learning scenarios, where a tutor should provide progressive support while also encouraging the student to reflect on the topic at hand. In practice, however, LLMs often provide the full reasoning process, correct errors too early, and frequently fail to adapt the level of support to the student's current needs [4]. These issues point to the need for pedagogical alignment, where model behavior not only with correctness, but also with effective teaching strategies such as scaffolding, gradual assistance, and student-centered guidance. Therefore, a tutoring system should focus on regulating how much help is given at each moment.

In order to address this need for explicit pedagogical control, we propose a guided multi-agent tutor for programming practice. Rather than treating tutoring as a single response-generation step, the proposed system decomposes each turn into three pedagogical functions through a Multi-Agent System (MAS): inferring the student's current help need, generating a response consistent with that need, and validating whether the response remains within the intended level of support.

To study the pedagogical value of this design, we compare three response-generation settings: a monolithic Socratic baseline based only on prompting, a guided multi-agent tutor without validation agent, and a full guided multi-agent tutor with validation. The comparison is based on a human evaluation with computer science teachers, who rated the generated responses in terms of pedagogical quality, gradual support, and over-helping. The main contributions of this paper are:

- We propose a guided multi-agent architecture for programming tutoring that separates student-state analysis, response generation, and pedagogical validation.
- We introduce two explicit control mechanisms for progressive help: a turn-level pedagogical state that captures the student's inferred help need and allowed help strength, and an activity characterization that distinguishes between explainable concepts and protected solution steps.
- We provide an initial human evaluation showing that guided multi-agent configurations are perceived as more gradual and less prone to over-helping than a monolithic Socratic baseline.

The remainder of the paper is structured as follows. Section 2 reviews related work; section 3 presents the proposed architecture; section 4 describes the evaluation setup; section 5 reports the results; and finally, section 6 concludes the paper and outlines future work.

2 Related Work

Programming education has emphasized that more than error correction is required to support effective student learning. In programming practice, students benefit from timely feedback, hints, and explanations adapted to their current level of understanding, especially novice learners, whose difficulties may range from syntax issues to program decomposition and conceptual misunderstanding [10, 9]. From this perspective, a useful tutor should not only identify whether a solution is correct or incorrect, but also decide what type of intervention or response is appropriate at each moment: a small hint, a conceptual clarification, or feedback on an intermediate step. This approach is closely related to scaffolding-based tutoring, in which assistance is most effective when it helps the learner progress gradually through the learning process.

In LLM-based programming tutoring, a central concern is that conversational fluency and apparent helpfulness do not necessarily translate into effective learning support. Although such systems may improve performance or reduce frustration in educational settings, these outcomes do not necessarily imply learning [3]. This is especially relevant for novice programmers, who may benefit from assistance but also need opportunities to reason, make mistakes, and progressively construct solutions. Thus, the same flexibility that makes LLMs useful in programming tasks can also create a pedagogical risk when models behave as answer-producing assistants rather than tutors.

This challenge has recently been framed as *pedagogical alignment*. Puech et al. [11] argue that current LLMs tend to reveal solutions too quickly and fail to sustain richer multi-turn tutoring strategies, proposing prompt-optimization methods to steer models toward predefined pedagogical plans rather than unconstrained helpfulness. In programming education, Ross et al. [12] similarly show that general-purpose LLMs often over-assist learners and present GuideLM, a fine-tuned model aimed at producing more pedagogically sound responses with stronger Socratic guidance and greater economy of words. In parallel, the broader LLM literature has increasingly moved from monolithic prompting toward *agentic* and multi-agent architectures, where different components specialize in tasks such as planning, state tracking, generation, verification, or critique. Recent surveys argue that such architectures are especially useful when systems require greater controllability, role specialization, and explicit coordination among intermediate decisions [7, 13, 5]. For example, MAIC proposes an LLM-driven MAS designed to balance scalability and adaptivity in online education [15]. Taken together, tutoring systems need explicit mechanisms to regulate disclosure, preserve productive struggle, and adapt the level of help to the learner's situation.

A related issue is how tutoring systems represent the student's current situation in order to guide those pedagogical decisions. Many conversational LLM tutors remain largely reactive at the turn level, without making explicit what kind of support is currently needed or how much help is pedagogically appropriate. Recent educational systems have started to introduce intermediate structured representations that connect learner-related information with downstream tutoring decisions, often for personalization, coordination, or interpretability [14, 8].

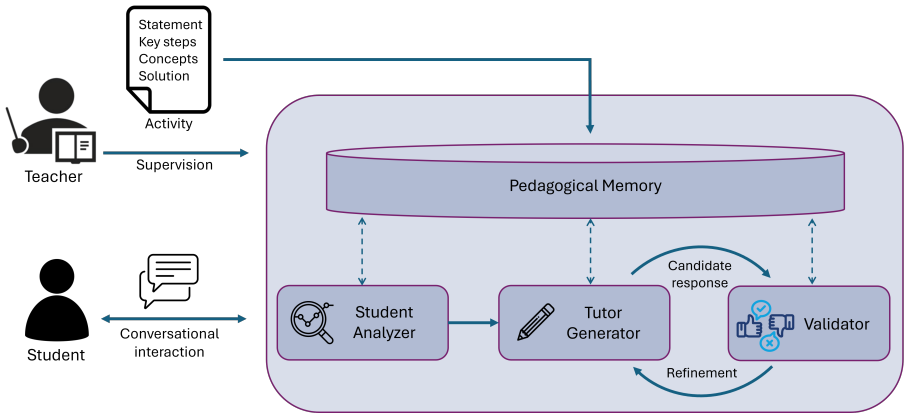


Fig. 1. Architecture of the guided multi-agent tutor, combining a shared pedagogical memory with three specialized agents for student-state analysis, response generation, and validation.

However, much of this work focuses on broad personalization or general orchestration, while comparatively less attention has been paid to turn-level pedagogical control in programming tutoring, where systems must infer a pedagogical state from the interaction and use it to regulate both the type and explicitness of the help provided.

Beyond pedagogical effectiveness, these educational systems inherently have other institutional considerations. Recent work in higher education has focused on the need for AI systems to be trustworthy, transparent, and compatible with institutional oversight. This includes concerns about privacy, explainability, governance, and the need for universities to retain meaningful control over how AI is integrated into teaching and learning processes [1, 2]. Although our work does not address digital sovereignty directly, it is aligned with this broader perspective in that pedagogical decisions are made explicit, the response-generation process is structured into inspectable components, and tutoring outputs are subject to an explicit revision step before delivery.

3 Architecture of the Guided Multi-Agent Tutor

This section presents the architecture of our guided multi-agent tutor for progressive help in programming practice. Instead of relying on a single prompt to interpret the student's need and generate the final response, the architecture separates three functions: student-state analysis, response generation, and response validation. This decomposition is intended to improve controllability and reduce over-helping while preserving local usefulness for the learner.

At each tutoring turn t, the system processes the current activity, the recent interaction history, and the latest student message through three specialized

agents: a *Student Analyzer*, a *Tutor Generator*, and a *Validator*. These components interact through a shared pedagogical memory that stores the context required for tutoring. Fig. 1 shows the overall architecture and the information flow among its components.

3.1 Activity Characterization and Pedagogical Memory

Each programming activity is represented through a structured characterization that provides both explanatory context and protected solution information. Formally, we define an activity as

$$a = (q, C, K, s^\star) \tag{1}$$

where q is the activity statement, C is the set of explanation concepts that the tutor may discuss, K is the set of key solution steps, and $s^\star$ is the reference solution.

This distinction is pedagogically important. The set C contains concepts that may be explained in order to clarify the task or support understanding, whereas K contains solution steps that should not be disclosed prematurely. In this way, the activity characterization provides an explicit boundary between *explainable content* and *protected content*, which is central to controlling over-helping in programming tutoring.

The system also maintains a shared pedagogical memory that stores the structured context required for tutoring at turn t. We represent this memory as

$$M_t = (a, \tilde{h}_t, \tilde{s}_t) \tag{2}$$

where a is the activity characterization, and $\tilde{h}_t$ and $\tilde{s}_t$ are, respectively, bounded windows of the recent interaction history available at turn t and of recent pedagogical states.

3.2 Pedagogical State Inference, Generation, and Validation

The Student Analyzer is responsible for inferring a pedagogical state from the current tutoring context. This state is not intended to be a comprehensive learner model; rather, it is a turn-level representation used to regulate the next tutoring move. We define the active pedagogical state as

$$s_t = (n_t, \alpha_t) \tag{3}$$

where n_t denotes the inferred help need and α_t denotes the allowed help strength.

The Student Analyzer uses the activity characterization, the recent interaction history, and the latest student message to determine which type of support is pedagogically appropriate next. The recent history is particularly useful for detecting repeated confusion or unsuccessful prior hints. In the current system, the inferred help need belongs to a small set of tutoring modes (see Table 1)

Table 1. Tutoring modes inferred by Student Analyzer agent.

Help mode	**Description**
Hint	A brief clue or local nudge intended to help the student make progress.
Conceptual explanation	A clarification of an underlying concept, principle, or misconception.
Procedural explanation	Guidance on how to carry out one immediate local phase of the task, without applying the procedure to the exercise.
Step validation	An evaluation of the student's current step, together with feedback.

designed to capture the main forms of progressive support in programming tutoring, while the help strength is restricted to `low` and `medium`. A higher level of explicitness is not considered, since it would make premature disclosure of key solution steps more likely. In particular, `medium` support is reserved for situations in which the student appears stuck on the same obstacle across turns.

Considering the pedagogical state, the Tutor Generator produces a proposed response that aligns with the inferred tutoring need, respecting the help mode and its strength. A key design principle is the distinction between conceptual explanation and solution disclosure. The tutor may explain concepts from C when appropriate, and under `medium` support it may reveal a limited protected step when this is necessary to help the student move forward. However, it should still avoid disclosing the overall solution structure or making the remaining task trivial. This is particularly important in programming tutoring, where revealing a decisive implementation step can collapse the remaining learning task.

The Validator evaluates whether a candidate response satisfies the pedagogical constraints of the system. In particular, it checks whether the response adheres to the selected help mode and its strength, and avoids solution leakage. If the response fails any of these checks, the validator returns revision issues that are fed back to the Tutor Generator in order to produce a refined version.

Formally, a candidate response $y_t^{(k)}$ is accepted only if it satisfies all pedagogical constraints:

$$\text{Valid}(y_t^{(k)}) = m_t^{(k)} \wedge \neg \ell_t^{(k)} \wedge \left(|I_t^{(k)}| = 0 \right) \tag{4}$$

where $m_t^{(k)}$ indicates adherence to the selected help mode and strength, $\ell_t^{(k)}$ solution leakage, and $I_t^{(k)}$ the set of revision issues identified by the validator agent.

This revision loop acts as an explicit pedagogical safeguard. Instead of trusting a single response-generation pass, the system checks whether the candidate response is appropriately restrained, sufficiently useful, and aligned with the student's current situation before it is shown to the learner.

Table 2. Scenario types used in the evaluation.

Alias	Scenario type	Description
BlockStart	Blocked start	The student does not know how to begin the exercise.
ConcConf	Conceptual confusion	The student does not understand a relevant concept.
StepVal	Step validation	The student asks whether an intermediate idea or step is correct.
EdgeConf	Edge-case confusion	The student is unsure why a special or boundary case must be considered.
BugCode	Buggy code	The student provides code that is partially correct but contains a local error.
RepBlock	Repeated blockage	The student remains blocked after multiple interactions.
PerConcept	Persistent conceptual difficulty	The student still does not understand a concept after prior explanation.

4 Experimental Setup

To evaluate the pedagogical value of the proposed architecture, we conducted a human evaluation comparing three assistants: (i) *Monolithic Tutor*, a monolithic baseline with a Socratic prompt; (ii) *Guided Tutor*, the guided multi-agent tutor without validator agent; and (iii) *Full Tutor*, the guided multi-agent tutor with validator. The goal of the evaluation was not to assess the functional correctness of the programming solutions but to compare the pedagogical quality of the tutoring responses produced by each configuration.

The evaluation was built around two introductory Python programming exercises: *Run-Length Encoding* and *Gradebook Summary*, a data-processing task in which students must traverse a list of `(student_name, grade)` pairs and compute summary values such as the average, the highest grade, and the number of passing students. For each exercise, we defined seven tutoring scenarios reflecting common student situations in programming practice (see Table 2). Since each scenario type was instantiated in both exercises and answered by the three tutoring configurations, the final evaluation set contained 14 exercise–scenario instances and 42 responses in total. All responses were generated using a quantized version of Qwen3.5-9B model.

The responses were evaluated by 9 computer science teachers through a questionnaire where the identity of the assistant that generated each response was hidden from them. For each item, evaluators were shown the exercise statement, the student message, the available conversation history when applicable, and one candidate response. They then rated that response on a 5-point Likert scale according—where 1 indicated strong disagreement and 5 indicated strong agreement—to three statements: *(i) The response has good pedagogical quality*, *(ii) The response helps the student advance gradually*, and *(iii) The response*

Table 3. Questionnaire scores by tutoring configuration. Values are reported as mean ± standard deviation. Lower scores are better for *Over-helping*. Bold indicates the most favorable mean value in each dimension.

Configuration	Pedagogical quality	Gradual support	Over-helping
Monolithic Tutor	3.35 ± 1.11	2.11 ± 1.06	4.44 ± 0.98
Guided Tutor	3.83 ± 0.90	4.06 ± 0.94	1.90 ± 1.12
Full Tutor	**4.12 ± 0.98**	**4.31 ± 0.89**	**1.60 ± 1.04**

reveals too much information. The first two items capture the perceived educational value of the response, whereas the third captures the risk of over-helping.

5 Results

Table 3 summarizes the questionnaire scores for the three tutoring systems. In general, both MAS outperformed the monolithic baseline in all three evaluation dimensions. Descriptively, the monolithic configuration obtained the lowest mean scores in pedagogical quality and gradual support, and the highest mean score in over-helping. By contrast, the *Full Tutor* configuration achieved the most favorable mean values overall, with the highest ratings in pedagogical quality and gradual support, and the lowest score in over-helping; this suggests that the Validator agent fulfills its role of improving the response, thereby achieving better pedagogical alignment.

In this regard, Fig. 2 provides a distributional view of these differences through mean teacher ratings. For *Pedagogical quality*, the guided configurations tend to receive higher ratings than the monolithic tutor, but the distributions still overlap substantially. In contrast, the separation is much clearer for *Gradual support* and *Over-helping*: the monolithic baseline is concentrated at low values in gradual support and high values in over-helping, whereas both guided configurations show the opposite pattern. In both dimensions, the full guided architecture with validator tends to show the most favorable distribution, although its advantage over the guided configuration without validator is smaller than the gap between either guided configuration and the monolithic baseline.

To assess whether the observed differences were statistically reliable, we first averaged ratings for each teacher, dimension, and tutoring configuration across all scenarios and activities. We then applied Friedman tests separately to each evaluation dimension, using tutoring configuration as the within-subject factor. The results are shown in Table 4, and they do not allow us to conclude that there is a statistically significant effect on pedagogical quality. In contrast, statistically significant differences were found for both gradual support and over-helping.

Therefore, these results suggest that the main improvement comes from making pedagogical control explicit through the guided architecture. Both guided configurations were perceived as substantially more gradual and considerably less prone to over-helping than the monolithic Socratic baseline. The validator contributed an additional descriptive improvement in all three dimensions, es-

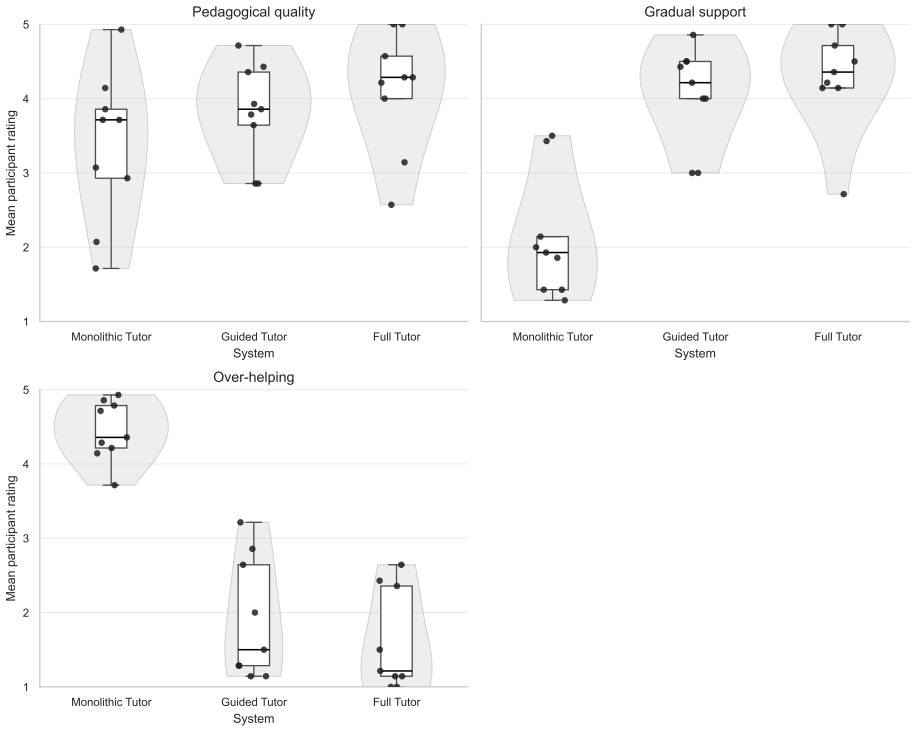


Fig. 2. Distribution of mean teacher ratings by tutoring configuration for the three evaluation dimensions. Each point represents the mean score assigned by one evaluator to a given configuration. Higher scores are better for pedagogical quality and gradual support, whereas lower scores are better for over-helping.

pecially in reducing over-helping, but this extra gain was not large enough to reach statistical significance with the present sample size.

In addition, Table 5 shows that the same overall pattern appears across scenario types. The only exception is *PerConcept*, where the *Guided Tutor* slightly exceeds the full architecture in pedagogical quality. This exception does not alter the general trend, but suggests that the benefit of the validator agent is more consistent in gradualness and restraint than in perceived pedagogical quality. The scenario-based results are especially informative for understanding where the guided architecture is most beneficial. The largest differences with respect to the monolithic baseline appear in scenarios involving blocked starts, conceptual confusion, repeated blockage, and persistent conceptual difficulty, that is, situations in which the regulation of help strength and the risk of over-helping are particularly relevant. In these cases, the full guided architecture consistently provides more gradual support while keeping disclosure at a lower level. This reinforces the interpretation that explicit pedagogical control is especially use-

Table 4. Friedman test results for each evaluation dimension.

Dimension	χ^2	p
Pedagogical quality	4.67	0.097
Gradual support	14.80	< 0.001
Over-helping	14.80	< 0.001

Table 5. Mean questionnaire scores by scenario type and tutoring configuration. Higher scores are better for pedagogical quality and gradual support, whereas lower scores are better for over-helping. Bold indicates the most favorable value in each scenario and dimension.

Scenario	**Configuration**	**Pedagogical quality**	**Gradual support**	**Over-helping**
BlockStart	Monolithic Tutor	3.33	1.89	4.83
	Guided Tutor	3.22	3.83	1.44
	Full Tutor	**4.00**	**4.33**	**1.28**
ConcConf	Monolithic Tutor	3.50	2.17	4.78
	Guided Tutor	4.00	4.06	1.94
	Full Tutor	**4.22**	**4.22**	**1.39**
StepVal	Monolithic Tutor	3.56	2.44	3.67
	Guided Tutor	3.67	4.39	1.50
	Full Tutor	**4.11**	**4.50**	**1.33**
EdgeConf	Monolithic Tutor	3.33	1.89	4.33
	Guided Tutor	3.83	3.94	1.94
	Full Tutor	**4.22**	**4.22**	**1.78**
BugCode	Monolithic Tutor	3.11	2.33	4.22
	Guided Tutor	4.00	4.22	2.11
	Full Tutor	**4.28**	**4.50**	**1.83**
RepBlock	Monolithic Tutor	3.22	1.94	4.72
	Guided Tutor	3.94	3.78	2.33
	Full Tutor	**4.06**	**4.06**	**2.17**
PerConcept	Monolithic Tutor	3.39	2.11	4.56
	Guided Tutor	**4.11**	4.17	2.00
	Full Tutor	3.94	**4.33**	**1.44**

ful when the student's need is underspecified, sustained across turns, or more vulnerable to over-assistance.

6 Conclusions and Future Work

This paper presented a guided multi-agent tutor for progressive help in programming practice. The proposed architecture separates student-state analysis, response generation, and response validation in order to make tutoring decisions more explicit and controllable. For this purpose, the system infers a turn-level pedagogical state that captures both the student's help need and the strength of assistance; and incorporates an explicit characterization of the activity that distinguishes between explainable concepts and protected solution structure, which helps regulate disclosure and reduce over-helping. This design also highlights an

active pedagogical role for the teacher, since the characterization of the activity becomes a key element for defining how the system should support the learner.

The results suggest that making pedagogical control explicit improves tutoring responses. In particular, both guided configurations were perceived as more gradual and less prone to over-helping than the monolithic baseline. Although the differences in pedagogical quality were not statistically significant, the descriptive results also favored the guided architectures. In addition, the full architecture with validator obtained the most favorable mean values overall, which is consistent with the intended role of the validator as an additional mechanism for pedagogical alignment.

This work has several limitations, including the limited number of evaluators, and the fact that the study measured teacher judgments rather than direct learning outcomes. Future work should therefore extend the evaluation to a larger and more diverse set of exercises and validate the system in authentic classroom settings with students. It would also be valuable to further study how teachers can participate in the definition and supervision of activity characterizations, as part of a more human-centered and pedagogically grounded use of learning assistants based on LLMs.

Acknowledgments. This work was funded by the University Teaching Research Networks Programme of the Institute of Education Sciences at the University of Alicante (Redes ICE), call 2025, reference 6368.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Alier Forment, M., García Peñalvo, F., Casañ Guerrero, M.J., Pereira, J.A., Llorens Largo, F.: Safe ai in education manifesto (October 2024), https://manifesto.safeaieducation.org, version 0.4.0
2. Arjona-Giner, S., Llorens-Largo, F., Real-Fernández, A., Molina-Carmona, R.: Trustworthy and open educational practices with ai: Small language models for digital sovereignty in higher education. In: García-Peñalvo, F.J., Sein-Echaluce, M.L., Fidalgo-Blanco, Á. (eds.) Open Educational Practices in the Artificial Intelligence Era, pp. 75–84. Springer Nature Singapore (2026)
3. Bassner, P., Lenk-Ostendorf, B., Beinstingel, R., Wasner, T., Krusche, S.: Less stress, better scores, same learning: The dissociation of performance and learning in ai-supported programming education. Computers and Education: Artificial Intelligence **10**, 100537 (2026). https://doi.org/https://doi.org/10.1016/j.caeai.2025.100537, https://www.sciencedirect.com/science/article/pii/S2666920X25001778
4. Borchers, C., Shou, T.: Can large language models match tutoring system adaptivity? a benchmarking study (2025), https://arxiv.org/abs/2504.05570
5. Chu, Z., Wang, S., Xie, J., Zhu, T., Yan, Y., Ye, J., Zhong, A., Hu, X., Liang, J., Yu, P.S., et al.: Llm agents for education: Advances and applications. arXiv preprint arXiv:2503.11733 **2** (2025)

6. González-Santamarta, M.Á., Rodríguez-Lera, F.J., Conde-González, M.Á., Rodríguez-Sedano, F., Fernández-Llamas, C.: Exploring the use of llms for teaching ai and robotics concepts at a master's degree. In: Alonso-Betanzos, A., Guijarro-Berdiñas, B., Bolón-Canedo, V., Hernández-Pereira, E., Fontenla-Romero, O., Camacho, D., Rabuñal, J.R., Ojeda-Aciego, M., Medina, J., Riquelme, J.C., Troncoso, A. (eds.) Advances in Artificial Intelligence. pp. 254–263. Springer Nature Switzerland, Cham (2024)
7. Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N.V., Wiest, O., Zhang, X.: Large language model based multi-agents: A survey of progress and challenges (2024), https://arxiv.org/abs/2402.01680
8. Huang, S., Sun, Y., Yu, X.: Sp-teachllm: An llm-driven framework for personalized and adaptive programming education. Information **16**(12) (2025). https://doi.org/10.3390/info16121045, https://www.mdpi.com/2078-2489/16/12/1045
9. Jeuring, J., Keuning, H., Marwan, S., Bouvier, D., Izu, C., Kiesler, N., Lehtinen, T., Lohr, D., Peterson, A., Sarsa, S.: Towards giving timely formative feedback and hints to novice programmers. In: Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education. p. 95–115. ITiCSE-WGR '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3571785.3574124, https://doi.org/10.1145/3571785.3574124
10. Pitts, G., Hridi, A.P., Lekshmi Narayanan, A.B.: A survey of LLM-based applications in programming education: Balancing automation and human oversight. In: Blodgett, S.L., Curry, A.C., Dev, S., Li, S., Madaio, M., Wang, J., Wu, S.T., Xiao, Z., Yang, D. (eds.) Proceedings of the Fourth Workshop on Bridging Human-Computer Interaction and Natural Language Processing (HCI+NLP). pp. 255–262. Association for Computational Linguistics, Suzhou, China (Nov 2025). https://doi.org/10.18653/v1/2025.hcinlp-1.21, https://aclanthology.org/2025.hcinlp-1.21/
11. Puech, R., Macina, J., Chatain, J., Sachan, M., Kapur, M.: Towards the pedagogical steering of large language models for tutoring: A case study with modeling productive failure (2025), https://arxiv.org/abs/2410.03781
12. Ross, E., Kansal, Y., Renzella, J., Vassar, A., Taylor, A.: Supervised fine-tuning llms to behave as pedagogical agents in programming education (2025), https://arxiv.org/abs/2502.20527
13. Tran, K.T., Dao, D., Nguyen, M.D., Pham, Q.V., O'Sullivan, B., Nguyen, H.D.: Multi-agent collaboration mechanisms: A survey of llms (2025), https://arxiv.org/abs/2501.06322
14. Wang, T., Zhan, Y., Lian, J., Hu, Z., Yuan, N.J., Zhang, Q., Xie, X., Xiong, H.: Llm-powered multi-agent framework for goal-oriented learning in intelligent tutoring system (2025), https://arxiv.org/abs/2501.15749
15. Yu, J., Zhang, Z., Zhang-li, D., Tu, S., Hao, Z., Li, R.M., Li, H., Wang, Y., Li, H., Gong, L., Cao, J., Lin, J., Zhou, J., Qin, F., Wang, H., Jiang, J., Deng, L., Zhan, Y., Xiao, C., Dai, X., Yan, X., Lin, N., Zhang, N., Ni, R., Dang, Y., Hou, L., Zhang, Y., Han, X., Li, M., Li, J., Liu, Z., Liu, H., Sun, M.: From mooc to maic: Reshaping online teaching and learning through llm-driven agents (2024), https://arxiv.org/abs/2409.03512