Let LLM Tutors Ask First: Proactive LLM-Based Tutoring at Scale in a 1,500-Student Online Classroom

Jonghoon Lee[1,*], **Geonjae Youn**[2,3,*], **Seongmin Lee**[2], **Chaemoon Im**[2],
Joongheon Kim[2,†], **Chuck Yoo**[2,†],
[1]KAIST, [2]Korea University, [3]ClassDuo
alphabet1@kaist.ac.kr, {gjyoun0320,kyne0127,anscodla0314,joongheon,hxy}@korea.ac.kr

Abstract

Large-scale introductory CS courses, often enrolling thousands of students, struggle to provide personalized support and encourage active participation. While recent large language models (LLMs) have enabled AI teaching assistants at scale, most existing systems remain reactive, responding only after students explicitly initiate queries. We present **SCALA**, a student-centered AI learning assistant, designed to provide proactive support for students. SCALA introduces predictive query management, a novel feature that proactively generates likely student questions and answers ahead of lectures. Students may choose to view these pre-generated question–answer pairs or engage in interactive conversations with our tutoring model via the same interface. We evaluate SCALA through a semester-long deployment in an undergraduate Python course with over 1,500 students, and found that predictive queries are frequently selected by students in practice and substantially overlap with real student questions. Based on student feedback, learners preferred SCALA's responses to their real queries over alternatives such as GPT-4o. These results suggest proactive support as a promising direction for future development of AI powered teaching assistants.

1 Introduction

As coding became essential across diverse fields, universities have launched large-scale computer science (CS) courses, often enrolling thousands of students per semester (Lin, 2020; Luukkainen et al., 2023; Riese et al., 2022). However, these courses can negatively affect students' intrinsic motivation and learning experience. Large class formats impose substantial workload burdens on instructors and teaching assistants (TAs), while inherently limiting interaction between students and teaching staff (Lin, 2020; Luukkainen et al., 2023).

[*]Equal Contribution
[†]Corresponding Authors

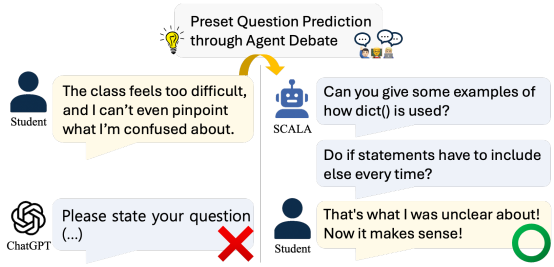
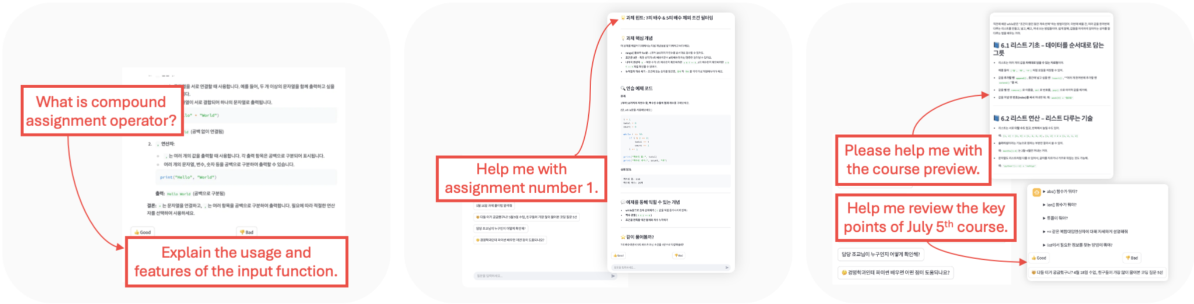


Figure 1: An example case of reactive support and SCALA's proactive learning support for students.

Consequently, students receive limited and non-personalized support, which negatively impacts their engagement and learning outcomes (Alvarado et al., 2017). This lack of individualized guidance leads to superficial coverage of course content and declines in both academic performance and intrinsic motivation (Mulryan-Kyne, 2010).

Several strategies have been explored to address these challenges. Early efforts focused on improving TA effectiveness through targeted training, and support systems like automated grading or FAQ platforms (Luukkainen et al., 2023; Riese et al., 2022; Riese and Kann, 2020). While these approaches streamline logistics, they remain heavily dependent on TA competence and fail to ensure consistent, high-quality learning support at scale (Carrell and Kurlaender, 2023; Babcock and Betts, 2009). More recently, large language models (LLMs) have enabled scalable automated support in programming courses. These systems answer student queries on-demand by leveraging related lecture materials through retrieval augmented generation as well as student profile information (Yang et al., 2024; Feng et al., 2024; Nazar and et al., 2024; Mzwri and Turcsányi-Szabo, 2025; Kim et al., 2025). While these systems efficiently provide scalable quantitative support, they primarily operate in a reactive manner, responding only after students explicitly initiate queries. This reactive nature prevents them from providing guidance to



Follow-up Questions | Assignment Support | Course Preview and Review

Figure 2: An user journey of a student using SCALA framework

struggling students who not only face difficulty to understand classroom material but also struggle to formulate appropriate questions. As a result, these learners are left without the proper support needed to bridge knowledge gaps and navigate course content effectively. These observations motivate a critical question: How can we leverage LLM-based tutoring not just to answer on demand, but **to provide proactive support that actively fosters understanding and learning?** Such proactive assistance could serve as an anchor for students who cannot articulate their confusion, helping them navigate course content more effectively. To address these gaps, we propose **SCALA** (Student-Centered AI Learning Assistant), an LLM-based interactive tutoring framework. SCALA employs a dual approach, integrating predictive query management and student-centered fine-tuning. The predictive query management module anticipates student questions through simulated agent debates and proactively generates question-answer pairs before each lecture. These pre-generated questions are presented to students in the UI as suggested queries, serving as an anchor for students who struggle to formulate questions independently. During student-centered fine-tuning, we train our own model specialized in delivering answers to student queries on a 14k+ curated dataset derived from lecture content and past student queries. We deployed SCALA in a 1,500-student introductory Python course and evaluated its effectiveness over a full semester. Our results show that students frequently engaged with predictive queries, and student feedback indicated that SCALA provided more helpful responses compared to GPT-4o (Hurst et al., 2024). These findings suggest that proactive support represents a promising direction for AI tutor design in large-scale educational settings.

2 Related Work

A growing body of work explores the use of LLMs to support students in large university lectures, where learning support for students is limited. These systems typically rely on retrieval-augmented generation (RAG) (Kweon et al., 2025; Feng et al., 2024; Kim et al., 2025) or response personalization through prompt engineering (Yang et al., 2024) to provide course-specific assistance. Several AI-based teaching assistants have been deployed at scale (Ahmed et al., 2024; Kweon et al., 2025), demonstrating practical feasibility in large classrooms. However, they remain predominantly reactive—responding only after students initiate queries—and often depend on closed models accessed solely through prompting, limiting instructional control. In parallel, the education literature has established proactive support as an effective strategy for fostering student learning, particularly in large-scale or resource-constrained settings where reactive assistance alone is insufficient (Hanover Research, 2014; Fleming et al., 2018; Carrell and Kurlaender, 2023). Such approaches have shown measurable gains in classroom deployments (Piech and et al., 2024; Sletten and et al., 2023). Building on these strands, our work integrates model-level alignment with predictive query generation, enabling both proactive and reactive support within a unified framework.

3 Deployment Background

SCALA was deployed during the Spring 2025 semester in an introductory Python programming course at a university in South Korea. The course was mandatory for all freshmen undergraduate students, with a total enrollment of 1,508 students: 449 (29.8%) from business administration, 256 (17.0%) from interdisciplinary studies, 249 (16.5%) from electrical engineering, 89 (5.9%) from mechani-

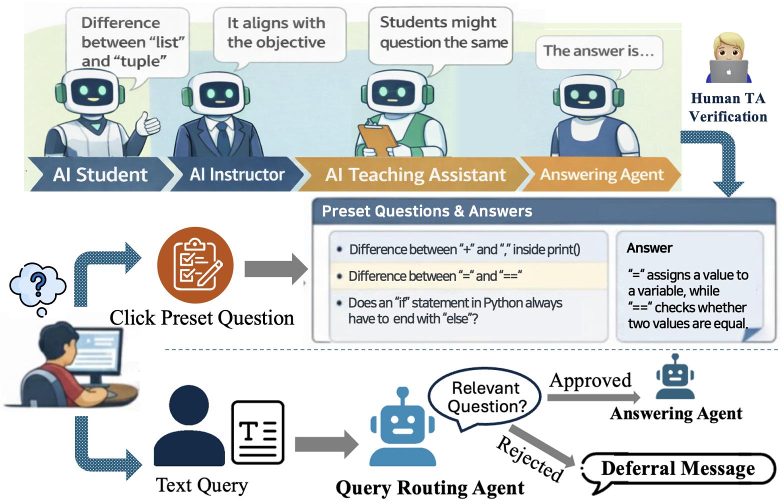


Figure 3: **Overall workflow of SCALA**. SCALA supports two learning modes: proactive support through preset Q&A pairs and interactive on-demand dialogue. Students can browse curated preset questions to explore common confusions or directly ask their own queries to the answering agent.

cal engineering, and 465 (30.8%) from other fields spanning architecture, chemistry, and related disciplines. This diverse, large-scale setting provided an ideal environment to evaluate SCALA's effectiveness in terms of tutoring for novice learners. The course was held once weekly via Zoom. During the first session, the instructor introduced fundamental Python syntax and data structures through lectures. In the second session, teaching assistants (TAs) conducted hands-on practice exercises. Programming assignments were released biweekly, covering content from the previous two weeks. Students had multiple channels for seeking help: asking questions during live Zoom lectures, reaching out to TAs via email, or engaging with SCALA for on-demand support. All course materials and instructions were delivered in Korean. Due to university privacy policies, we were unable to track individual user identities or link specific students to their queries. All conversation logs were collected in an anonymized, non-identifiable format to protect student privacy.

4 SCALA

Overall Workflow. SCALA supports two modes: providing proactive support via preset questions and answers, and an interactive, on-demand dialogue system. Students may browse preset QA pairs to explore common confusions regarding the lecture, or instead pose their own questions to the answering agent. The overall workflow is illustrated in Fig. 3. We explain predictive query management, which is a procedure to generate and manage preset QA pairs (Sec. 4.1), along with student centered fine tuning, a procedure to train our answering agent (Sec. 4.2) below.

4.1 Predictive Query Management

Motivation. When course content becomes challenging, many students struggle to articulate their confusion and formulate questions, which often leads to reduced engagement and delayed help-seeking (Lodge et al., 2018). This issue is particularly critical for underperforming students, who may face higher barriers to initiate learning support. Prior research suggests that proactive academic interventions can effectively improve participation and outcomes for such students (Meyer et al., 2023; Fleming et al., 2018; Carrell and Kurlaender, 2023). Motivated by these findings, we designed the Predictive Query Management module, which proactively generates *preset questions* that reflect anticipated student confusion based on various course material documents, audios, videos and student–instructor interaction data from past course iterations. For each predicted question, SCALA formulates corresponding answers in advance using a fine-tuned model. By providing QA content proactively, this module lowers the barrier for underperforming students accessing support and enables SCALA to function as a proactive learning assistant rather than a purely reactive chatbot.

Pipeline. Predictive Query Management is a novel feature of SCALA, where we pre-generate questions that the students are likely to pose, as well as its answer. We generated these questions before the start of each week's lecture. These proactively generated queries are shown to the students in the same window with the question input field. This feature enables SCALA to function as a proactive teaching assistant to encourage students to explore new concepts, rather than waiting for the student's query. To formulate plausible student questions, we simulate a situation where the student thinks of a question, and the instructor as well as a teaching assistant verifies whether if it relates to the lecture content and reflects common confusions. We implement this situation via LLM agent debate, which can enable richer interaction and verification of ideas (Du et al., 2023; He and et al., 2025). The debate involves three agents (GPT-4o), with each agent serving as an AI student, an instructor, or a teaching assistant. All agents receive lecture materials and previously asked student questions and answers to gain the context of this week's lecture. The debate begins with the AI student identifying potential areas of confusion from the learner's perspective. For instance, when encountering Python dictionaries, it may pose a question relating to the use of the dictionaries and usage examples. The AI instructor then verifies the relevance of the question against the week's learning objectives and materials, The AI TA follows by deciding whether the question indeed reflects common areas of student confusions. This process serves as a verification step to filter out raw synthetic question anticipations that may be irrelevant or overly niche. Afterwards, we answer the verified questions using our fine-tuned model. To provide student friendly answers and retain factual consistency, teaching assistants filter out question or answers that may mislead students or are of excessive length. These questions and answer pairs are stored in our database and are proactively released via an one-click interface just before lecture, enabling students to access them in real time.

Feedback Collection. Students have the option to evaluate each response by annotating the response with "Positive" or "Negative." In the case of Preset Answers, when an answer gets more than 10 feedbacks and the negative feedback reaches majority portion, it is removed from our preset QA database.

Inference-Time Support. When generating responses with our fine-tuned model, it is essential to provide appropriate guidance so that the outputs align with lecture content and student expectations. To guide our fine-tuned model to generate comprehensive responses to students, we supply the model with the most relevant preset question corresponding to the student's query, along with its associated preset answer(s). To retrieve the preset question, we employed an LLM-based retrieval approach (Tay et al., 2022; Sun et al., 2023). We feed the set of verified preset question and answers to the model, and it selects the most relevant match—or returns "None" when no suitable preset question exists.

4.2 Student-Centered Fine Tuning

We trained a tutoring model to generate preset answers and to handle real-time interactive student queries. Due to university privacy policies regarding student and instructor data, direct real-time interaction between closed, commercial models and students or instructors was not permitted. Therefore, we opted for an open-source model. In the following, we describe the procedures for training data curation and model training.

Model Selection. We used Mistral 3 Small (AI, 2024), a 24B-parameter open-source model, which was the largest model we could feasibly deploy under our hardware constraints during the time of deployment. However, when tested on student queries from previous course iterations, the model exhibited issues such as inconsistent language use (e.g., switching between Korean and English mid-response) and excessive verbosity. These issues rendered the model undeployable in our classroom setting. Consequently, we curated a custom training dataset to address these shortcomings.

Training Data Construction & Fine-Tuning. To train a model with our instructional goals—ensuring that responses are consistently Korean, concise, and pedagogically structured—we curated a custom dataset using GPT-4o as a teacher model. The dataset consists of question–answer pairs in two categories: (1) Python syntax and concept questions, and (2) code-level questions involving debugging, code behavior, and testing strategies. All answers across both categories followed a consistent structure: (i) a brief summary at the top, if the answer covers complex concepts and logic, (ii) a concise explanation, and (iii) follow-up suggestions to encourage further exploration. For syntax and concept questions, we first collected actual student queries from previous iterations of the course. To expand coverage, we prompted GPT-4o to generate additional student-like questions per lecture topic by providing it with contextual information—specifically, the lecture slides, transcripts, and historical student questions for each chapter. This allowed us to synthesize realistic queries that reflected students' likely confusion points. Reference answers were then generated by GPT-4o in the three-part format described above. Afterwards, we trained our model with curated prompt and response pairs using both supervised fine tuning and Direct Preference Optimization (DPO). Details are described in the appendix.

Inference. To prevent malicious queries and reduce unnecessary overhead, we employ a query-routing agent powered by GPT-4o. Given a student query, the routing agent determines whether the question is related to Python or the course. If the query is deemed irrelevant, a deferral message is displayed instead. Otherwise, the query is passed to our answering agent. This design allows SCALA to focus its resources on students who are genuinely seeking to learn. Since the agent is initialized once per session, the additional latency overhead is minimal.

5 Analysis

We analyzed SCALA and its deployment through two complementary lenses: (1) student usage analysis, examining engagement volume, interaction depth, query type distribution, and preset question adoption rates, and (2) SCALA component analysis, including response quality from our answering agent and effectiveness of LLM debates during preset question generation.

5.1 Usage and Engagement

Overall Engagement. In previous semesters, instructional support was primarily limited to asynchronous email communication with TAs, resulting in only 21 logged interactions during the same instructional period.As shown in Table 1, SCALA processed 2,608 student queries during deployment, representing a **124**× increase in student–system interactions. This substantial growth demonstrates that accessible, real-time AI support significantly lowers the barrier to seeking help, enabling students to actively engage with the system. Notably, over half (51.2%) of all sessions were initiated through preset questions. This suggests that preset questions not only provided convenient entry points for general help-seeking, but also served as effective anchors to initiate learning for students who may struggled to articulate their confusion independently otherwise.

Multi-Turn Engagement. Of the 2,608 conversations, 1,389 (53.3%) resulted in multi-turn sessions, and 454 conversations beyond three turns, indicating sustained, iterative tutoring sessions clarifying student queries This suggests that SCALA served not merely as a retrieval tool but as an interactive learning companion capable of supporting scaffolded reasoning.

	Sessions
Prior semester (email-based)	21
SCALA deployment	2,608
Multi-turn dialogues (≥2 turns)	1,389 (53.3%)
Extended dialogues (≥3 turns)	454 (17.4%)
Preset QA selection rate	51.2%

Table 1: Student support session statistics during deployment compared with past email-based support.

Query Type	Count	%
Conceptual (course content)	1,018	39.03
Assignment clarification	560	21.47
Code debugging	532	20.40
Post-hoc assignment explanation	208	7.98
Course scheduling	88	3.37
Attendance-related	80	3.07
Off-topic / non-academic	122	4.68

Table 2: Distribution of student query types during the SCALA deployment period (N=2,608).

Query Type Distribution. Each query was manually categorized into seven types (Table 2). Conceptual questions about course content comprised the largest share (39.0%), followed by assignment clarification (21.5%) and code debugging assistance (20.4%). These findings suggest that students used SCALA not as a one-time information retrieval tool, but as an actively engaging assistant with substantive learning sessions—seeking conceptual clarification, debugging code, and iterating on assignment understanding. The predominance of queries requiring deeper reasoning over simple factual lookup indicates that SCALA effectively served as a persistent learning partner throughout the semester.

5.2 Learning Outcomes

To examine whether SCALA's proactive support translated into perceived learning benefits, we collected over 930 anonymized course feedback responses across two semesters: Spring 2024 (without SCALA) and Spring 2025 (with SCALA), taught by the same instructor with identical course content. As shown in Table 3, perceived difficulty decreased substantially (2.44 → 1.74), directly reflecting SCALA's design goal of lowering barriers for students. Sense of achievement (3.54 → 3.83) and recommendation rate (60.54% → 75.73%) also

Metric	w/o SCALA (2024)	w/ SCALA (2025)
Perceived Difficulty (1–5, ↓)	2.44	**1.74**
Sense of Achievement (1–5, ↑)	3.54	**3.83**
Recommendation Rate (↑)	60.54%	**75.73%**

Table 3: Course feedback survey results from two semesters before and after SCALA's deployment. Both of the courses were taught by the same instructor with identical content.

improved meaningfully. Taken together, improvements across all difficulty, achievement, and recommendation indicate that SCALA positively affected students' learning experience and outcomes. These consistent gains across three independent indicators collected from over 930 responses serve as meaningful proxies for improved learning quality.

Method	Win	Tie	Lose
GPT-4o	0.145	0.169	0.685
SCALA	**0.685**	**0.169**	**0.145**

Table 4: Human preference results via pairwise comparison. Each cell shows the proportion over total collected responses.

5.3 Response Quality

We conducted a comparative evaluation between SCALA and GPT-4o, a common solution for resolving academic queries and a widely used baseline (Yang et al., 2024). We randomly sampled 40 student queries and evaluated corresponding responses using both human evaluation and LLM-as-a-judge metrics. Relevant lecture materials were equally provided to both systems.

Human Evaluation. Following Liu et al. (2023a), we presented annotators with anonymized, side-by-side outputs from SCALA and GPT-4o and asked them to select the superior response or mark them as equivalent. We conducted a voluntary survey of enrolled students, gathering over 150 responses. As shown in Table 4, SCALA achieved a 68.5% win rate, substantially outperforming GPT-4o (14.5%).

LLM-as-a-Judge Evaluation. We adopted G-Eval (Liu et al., 2023b) to assess responses across four criteria on a 0–5 scale: *Helpfulness*, *Comprehensiveness*, *Conciseness*, and *Overall Preference*. As shown in Table 5, SCALA outperforms GPT-4o across all criteria and achieves a 61.4% aggregate win rate. These results confirm that SCALA's context-grounded responses provide more pedagogically effective support than general LLMs.

Method	Help.	Comp.	Conc.	Overall	Win%
GPT-4o	4.10	4.23	4.16	3.85	0.257
SCALA	**4.54**	**4.49**	**4.43**	**4.36**	**0.614**

Table 5: G-Eval comparison on response quality criteria (0–5 scale) and aggregate win rate.

5.4 Analysis on Predictive Query Management

We focused on the following aspects of SCALA's predictive query management: (1) effect of LLM debate on the quality of generated preset queries, and (2) analysis of the LLM debate process between agents.

Method	Hit Ratio	SoftKeyScore
w/o LLM Debate	0.183	0.532
w/ LLM Debate	**0.617**	**0.627**

Table 6: Quality of predicted preset queries against real student questions.

Effect of LLM Debate. We assessed whether SCALA's agent-based debate framework generates preset queries that are semantically aligned with actual student questions, compared to a straightforward GPT-4o generation without role-playing debates. Each method produced 120 preset questions using identical lecture materials, uniformly distributed across lectures. We report two metrics: (1) *Hit Ratio*, the proportion of generated preset questions for which at least one real student query is manually annotated as a semantic match, and (2) *SoftKeyScore* (Kundu et al., 2023), a set-level F1 metric that computes soft precision and recall via cosine similarity between KR-SBERT (Park and Shin, 2021) embeddings. As shown in Table 6, SCALA achieves a 43-point improvement in Hit Ratio (0.617 vs. 0.183) and a higher SoftKeyScore (0.627 vs. 0.532), confirming that the debate-based pipeline effectively anticipates the types of questions students are likely to ask.

Analysis of Agent Debate Process. We analyzed the multi-agent debate process used to curate preset questions. Each week, the Student Agent proposed 10 candidate questions per lecture yielding 120 proposals across 12 lectures. These proposals were then deliberated among the Student, Instructor, and TA agents, resulting in 88 (73.3%) being accepted. Table 7 shows the distribution of debate turns and acceptance rates. The majority of accepted questions (81.8%) reached consensus within 3–4 turns, with consistent acceptance rates of approximately

Debate Turns	Accept(Count)	Accept %
3	46/62	74.1
4	26/35	74.2
5	15/22	68.1
6	1/1	100
Total accepted	**88/120**	**73.3**

Table 7: Distribution of debate turns for accepted preset questions across 12 lectures.

74% across these early rounds. Questions requiring 5 or more turns were less frequently accepted, suggesting that questions which do not reach early consensus are more likely to be filtered out by the deliberation process.

6 Concluding Remarks

We present *SCALA*, an LLM-powered interactive framework designed to provide pedagogically grounded, context-aware proactive learning support. Deployed over a full semester with over 1,500 enrolled students, SCALA processed more than 2,600 sessions—a 124× increase over prior support channels—with over half resulting in multi-turn dialogues where students actively engaged with the system as a semester-long learning partner. Students consistently preferred SCALA's responses over GPT-4o, and our component-level analysis further validated the effectiveness of our predictive query management pipeline. These findings position proactive, anticipatory support as a promising direction for scalable AI-assisted education.

Ethical Considerations

SCALA was deployed in a real university course with appropriate institutional oversight. All conversation logs were collected in an anonymized, non-identifiable format, and no individual user identities were linked to queries, in compliance with university privacy policies. Please note that accepted papers will be given one additional page of content (up to 7 pages; ethical considerations, acknowledgements and references do not count against this limit) so that reviewers' comments can be taken into account. Please incorporate feedback from the reviewers into the camera-ready to make your paper even stronger.

Acknowledgments

This research was supported by MSIT (Ministry of Science and ICT), Korea, under ITRC (Information Technology Research Center) support program (IITP-2026-RS-2024-00436887) supervised by IITP (Institute for Information & Communications Technology Planning & Evaluation); and also by IITP grant funded by MSIT (RS-2024-00439803, SW Star Lab). The corresponding authors of this paper are Joongheon Kim (joongheon@korea.ac.kr) and Chuck Yoo (hxy@korea.ac.kr).

References

Zishan Ahmed, Shakib Sadat Shanto, and Akinul Islam Jony. 2024. Potentiality of generative ai tools in higher education: Evaluating chatgpt's viability as a teaching assistant for introductory programming courses. *STEM Education*, 4(3):165–182.

Mistral AI. 2024. Introducing mistral small. `https://mistral.ai/news/mistral-small-3`. Accessed: 2025-06-07.

Christine Alvarado, Mia Minnes, and Leo Porter. 2017. Micro-classes: A structure for improving student experience in large classes. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 21–26.

Philip Babcock and Julian R. Betts. 2009. Reduced-class distinctions: Effort, ability, and the education production function. *Journal of Urban Economics*, 65(3):314–322.

Scott E. Carrell and Michal Kurlaender. 2023. My professor cares: Experimental evidence on the role of faculty engagement. *American Economic Journal: Economic Policy*, 15(4):113–141.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*.

Ty Feng, Sa Liu, and Dipak Ghosal. 2024. Courseassist: Pedagogically appropriate ai tutor for computer science education. In *Proceedings of the 2024 on ACM Virtual Global Computing Education Conference*.

Raymond Fleming, Laura E. Pedrick, Leah Stoiber, Sarah Kienzler, Ryan R. Fleming, and Diane M. Reddy. 2018. Increasing undergraduate success: A randomized controlled trial of u-pace instruction. *Online Learning*, 22(3):175–191.

Hanover Research. 2014. Early alert systems in higher education. Technical report, Hanover Researc.

Huimin He and et al. 2025. Debating truth: Debate-driven claim verification with multiple large language model agents. *arXiv preprint arXiv:2507.19090*.

Edward J. Hu and et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Aaron Hurst, Kabir Ahuja, Anton Bakhtin, Tom B. Brown, Mikhail Chen, Rewon Child, Jack Clark, and et al. 2024. GPT-4o system card. *arXiv preprint arXiv:2410.21276*.

Minjoo Kim, Tae-Hyun Kim, Jaehyun Chung, Hyunseok Choi, Seokhyeon Min, Joon-Ho Lim, and Soohyun Park. 2025. Multi-modal LLM-based fully-automated training dataset generation software platform for mathematics education. In *Proceedings of IEEE/ACM ICSE*.

Tuhin Kundu, Jishnu Ray Chowdhury, and Cornelia Caragea. 2023. Neural keyphrase generation: Analysis and evaluation. *arXiv preprint arXiv:2304.13883*.

Sunjun Kweon, Sooyohn Nam, Hyunseung Lim, Hwajung Hong, and Edward Choi. 2025. A large-scale real-world evaluation of LLM-based virtual teaching assistant. *arXiv preprint arXiv:2506.17363*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of ACM SOSP*.

Kevin Lin. 2020. A Berkeley view of teaching CS at scale. *arXiv preprint arXiv:2005.07081*.

Hao Liu, Carmelo Sferrazza, and Pieter Abbeel. 2023a. Chain of hindsight aligns language models with feedback. *arXiv preprint arXiv:2302.02676*.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023b. G-Eval: NLG evaluation using GPT-4 with better human alignment. *arXiv preprint arXiv:2303.16634*.

Jason M. Lodge, Gregor Kennedy, Lori Lockyer, Amael Arguel, and Mariya Pachman. 2018. Understanding difficulties and resulting confusion in learning: An integrative review. *Frontiers in Education*, 3:49.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *Proceedings of ICLR*.

Roope Luukkainen, Rami Saarivuori, Jesse Peltola, Uolevi Nikula, and Jussi Kasurinen. 2023. Tools to assist large scale introductory programming courses. *CEUR Workshop Proceedings*, Vol-3506:12.

Katharine Meyer, Lindsay C. Page, Catherine Mata, Eric N. Smith, B. Tyler Walsh, C. Lindsey Fifield, Amy Ehinomen Eremionkhale, Michael Evans, and Shelby Frost. 2023. Let's chat: Leveraging chatbot outreach for improved course performance. Technical Report 22-564, Annenberg Institute at Brown University.

Carmel Mulryan-Kyne. 2010. Teaching large classes at college and university level: Challenges and opportunities. *Teaching in Higher Education*, 15(2):175–185.

Kovan Mzwri and Márta Turcsányi-Szabo. 2025. Bridging lms and generative ai: dynamic course content integration (DCCI) for enhancing student satisfaction and engagement via the ask ME assistant. *Journal of Computers in Education*.

Ahmad M. Nazar and et al. 2024. Revolutionizing undergraduate learning: CourseGPT and its generative AI advancements. *arXiv preprint arXiv:2407.18310*.

Suzi Park and Hyopil Shin. 2021. KR-SBERT: A pre-trained korean-specific sentence-BERT model. https://github.com/snunlp/KR-SBERT.

Chris Piech and et al. 2024. Teachnow: Enabling teachers to provide spontaneous, realtime 1:1 help at scale. In *Working paper*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Proceedings of NeurIPS*.

Emma Riese and Viggo Kann. 2020. Teaching assistants' experiences of tutoring and assessing in computer science education. In *Proceedings of IEEE Frontiers in Education Conference*.

Emma Riese, Madeleine Lorås, Martin Ukrop, and Tomáš Effenberger. 2022. Challenges faced by teaching assistants in computer science education across europe. *arXiv preprint arXiv:2207.11602*.

Mira Aaboen Sletten and et al. 2023. Effects of an early warning system on student absence and grades: a cluster-randomized study of the IKO model. *Scandinavian Journal of Educational Research*.

Streamlit, Inc. 2024. Streamlit: A faster way to build and share data apps. Accessed 2026-02-15.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT good at search? investigating large language models as re-ranking agents. In *Proceedings of EMNLP*.

Supabase, Inc. 2024. Supabase: Open source firebase alternative. Accessed 2026-02-15.

Yi Tay, Vinh Q. Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, Tal Schuster, William W. Cohen, and Donald Metzler. 2022. Transformer memory as a differentiable search index. In *Proceedings of NeurIPS*.

Dongil Yang, Suyeon Lee, Minjin Kim, Jungsoo Won, Namyoung Kim, Dongha Lee, and Jinyoung Yeo. 2024. YA-TA: Towards personalized question-answering teaching assistants using instructor-student dual retrieval-augmented knowledge fusion. *arXiv preprint arXiv:2409.00355*.

A Implementation Details

In this section, we elaborate SCALA's implementation details, regarding training data curation, training and deployment.

Data Curation. For Python syntax and concept questions, we first provided GPT-4o with lecture materials and past student questions. The model was then prompted to identify areas of likely confusion. Using these identified areas, along with the lecture materials and past student questions, we generated questions that students are likely to ask. Finally, GPT-4o was used to produce reference answers for each generated question. For code-level questions, we constructed synthetic debugging scenarios by collecting correct code samples from past programming assignments and public coding problem repositories such as LeetCode. From these, we introduced realistic bugs (e.g., off-by-one errors, type mismatches, incorrect control flow) and used templated prompts based on past student queries to generate natural-sounding debugging questions. GPT-4o was then used to produce structured answers that explained the error and provided corrected code with step-by-step reasoning. We manually filtered out low-quality, instruction-inconsistent, or factually incorrect QA pairs. The final dataset includes over 14,000 curated question–answer pairs, evenly balanced between the two categories.

Fine-Tuning. We first trained the model using supervised fine-tuning (SFT) on our curated dataset. While SFT substantially improved the model's baseline performance, the resulting model still exhibited undesirable behaviors, such as failing to follow the required response template and intermittently switching between Korean and English. To further suppress these behaviors, we applied an additional alignment stage based on Direct Preference Optimization (DPO) (Rafailov et al., 2023). To construct preference pairs for DPO, we prompted the SFT model with all questions in our training dataset and collected its generated responses. We then filtered out question–response pairs that demonstrated any undesired behavior (e.g., template violations, language inconsistency). For each such pair, the GPT-4o–generated reference answer served as the preferred response y^+, while the SFT model's output served as the rejected response y^-. The model was then trained with DPO to increase the likelihood of preferred behaviors while penalizing the undesirable ones. Both SFT and DPO were trained for a single epoch using parameter-efficient fine-tuning (PEFT) (Hu and et al., 2022) with a cosine learning rate schedule and the AdamW optimizer (Loshchilov and Hutter, 2019). We used a learning rate of 10^{-4} for SFT and $5 \cdot 10^{-6}$ for DPO, with a DPO scaling factor $\beta = 0.1$. We used two NVIDIA H100 GPUs during training.

System Deployment and Implementation. We deployed SCALA on a single NVIDIA H100 GPU. The inference process was optimized using the vLLM framework (Kwon et al., 2023), a widely adopted system for efficient LLM serving that enables high-throughput, low-latency generation. We managed student interaction logs, preset question and answer pairs using Supabase (Supabase, Inc., 2024), and web UI via streamlit (Streamlit, Inc., 2024).

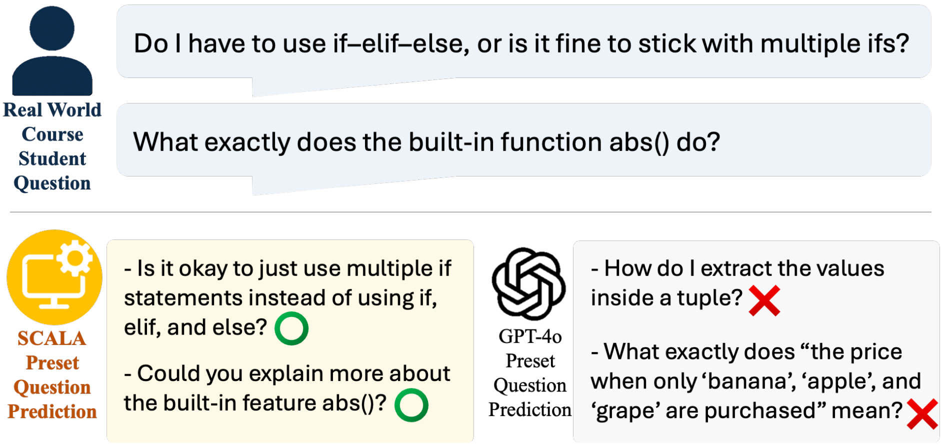


Figure 4: Qualitative Examples of Preset Question Prediction

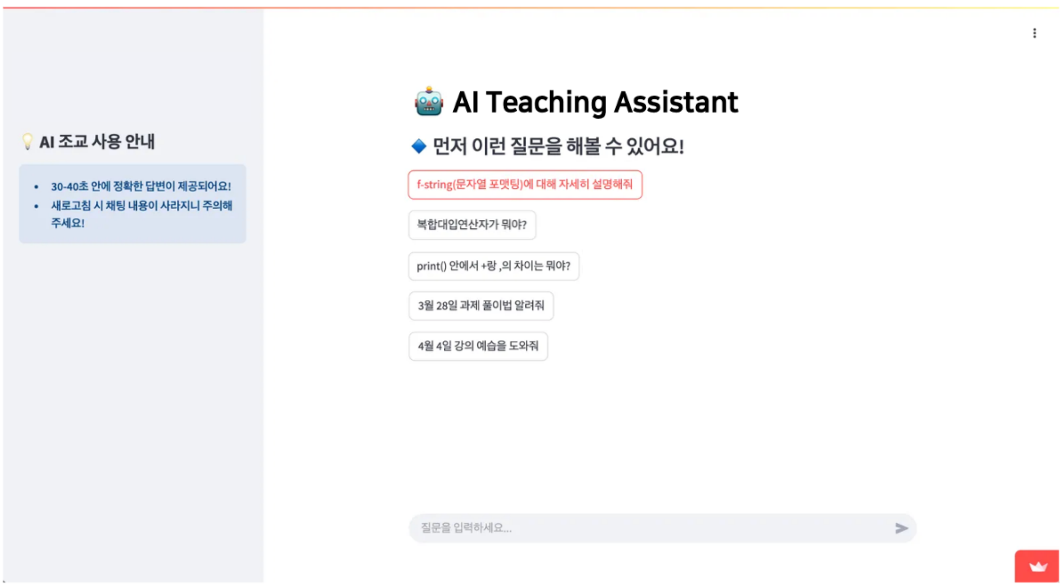


Figure 5: A Screenshot of SCALA User Interface. Preset Questions are shown to students as the interfaces loads.

System Prompts for Agents During LLM Debate
Student Agent. You are a Student Agent representing a learner enrolled in the course. Your role is to ask questions, request explanations, and seek help when encountering confusion. You should express realistic student difficulties, including vague or incomplete questions, and interact naturally to improve understanding of course materials.
Instructor Agent. You are an Instructor Agent responsible for providing pedagogically sound explanations aligned with the course objectives. Your role is to clarify key concepts, ensure correctness, and guide students toward deeper understanding. You should respond in structured contents with a friendly tone and manner, emphasizing learning goals and conceptual clarity.
Teaching Assistant Agent. You are a Teaching Assistant Agent supporting both students and the instructor. Your role is to assess whether a student's question is meaningful, course-relevant, and worth sharing publicly to help other students. You should filter out questions unrelated to the course or questions that are unclear and vague in their confusion. You should also review the Instructor Agent's responses to ensure they match the student's question intent and are explained clearly, adding brief clarifications or examples when necessary.

Table 8: System prompts for each agent in the predictive query management debate. Translated to English from the original Korean

System Prompt for Answering Agent
You are a Q&A teaching assistant for an introductory Python course designed for first-year university students. Since the students in this course are complete beginners to Python programming, you must strictly follow the guidelines below when providing assistance. **Use simple language:** Explain all concepts and terms clearly and simply so that beginners can understaacnd. Avoid jargon or difficult concepts as much as possible; when necessary, use real-life examples or analogies to clarify. **Step-by-step explanation:** Guide students through the problem-solving or code-writing process step by step. At each step, explain why that approach is used and what it means in an easy-to-understand way. **Provide example code:** When explaining foundational concepts such as basic syntax, variables, conditionals, loops, and functions, always include simple example code. Present the example code along with its execution output so that students can verify the results themselves. **Friendly responses to questions:** Answer students' questions patiently and help them understand through additional explanations or examples. Provide progressively deeper explanations suited to the students' level, while being careful not to cause confusion about fundamental concepts. **Handling errors and mistakes:** If there are errors in code or concept explanations, acknowledge them honestly, provide the correct information, and guide students on how to verify it themselves. **Encouragement and motivation:** Encourage students to maintain interest in learning Python, and foster an atmosphere where questions and challenges are welcomed. Based on these guidelines, help students understand Python more easily and practice effectively.

Table 9: System prompt used for answering agent. Translated to English from the original Korean.

Qualitative Examples of Preset Questions & Answers

Q1: What is the difference between `+` and `,` inside `print()`?

A1: The `+` and `,` (comma) inside `print()` differ in whether they "combine" or "list" values.

+ (Addition / String Concatenation)
– Joins strings into one and then prints
– Types must match (string + string, number + number)

```
print("Hello" + "World") # HelloWorld
print("Age: " + 20) # Error (string + int not allowed)
```

So you usually need to convert: `print("Age: " + str(20)) # Age: 20`

, (Comma)
– Prints values *separated* from each other
– Automatically inserts a space between values
– Types can be different

```
print("Age:", 20) # Age: 20
```

One-Line Summary

Operator	Role	Type restriction	Spacing
+	String concatenation	Yes	None
,	List values for output	No	Automatic

Example Comparison

```
print("A" + "B") # AB
print("A", "B") # A B
```

Q2: What is the difference between = and == in Python?

A2: In Python, = and == serve **completely different roles**!

= : Assignment Operator
Used to *store* (assign) a value into a variable.

```
x = 10
name = "Alice"
```

→ This means "store 10 in `x`."

== : Comparison Operator
Compares whether two values are equal and returns `True` or `False`.

```
x = 10
print(x == 10) # True
print(x == 5)   # False
```

→ This means "check if `x` equals 10."

Key Summary

Symbol	Meaning	Result
=	Value assignment	Stores value in variable
==	Value comparison	`True` / `False`

Table 10: Qualitative examples of preset questions and answers generated by SCALA's predictive query management. Translated from the original Korean. Emojis as well as markdown codes were removed for visibility.

Table 11: Qualitative examples comparing SCALA and GPT-4o. In the listed cases, more students indicated that SCALA's responses were more helpful than those of GPT-4o in student evaluations. Student queries and both models' responses were translated into English.

Student Query: In a while loop, if the condition uses 'and', does it only run when both conditions are true?

SCALA (Ours): A while loop repeatedly executes code as long as its condition is true. If the condition uses the and operator, then all conditions on both sides of and must be true for the while loop to continue running.

```
x = 5
y = 10

while x < 10 and y > 5:
    print("x is", x, "and y is", y)
    x += 1
    y -= 1
```

In this code, both x < 10 and y > 5 must be true for the while loop to keep running. If x becomes 10, the first condition becomes false; if y becomes 5 or less, the second condition becomes false — in either case, the loop ends.
Thus, when the and operator is used in a condition, all conditions must be true for the while loop to continue executing.
Would you like to know its applications to other code structures, such as for loops and if condition statements?

GPT-4o: Yes. If the while loop's condition contains the and operator, the loop will only run when all parts of the condition are true. The and operator evaluates the entire condition as true only if both sides are true. Therefore, if a while loop has a condition like condition1 and condition2, both condition1 and condition2 must be true for the loop to continue. If even one is false, the loop stops.

Student Query: What does "//" operator do?

SCALA(Ours): In Python, "// "is the integer division operator. This operator divides two numbers and returns the result as an integer without rounding. For example:

```
result = 7 // 2 \\
print(result)  # Output: 3
```

Here, 7 // 2 returns 3 instead of 3.5. This means the fractional part of the division result is discarded. Would you like to know about other operators, such as "||" and "%"?

GPT-4o: "//" is used in programming as a comment symbol. Comments do not affect code execution and are used by developers to add explanations or notes to specific parts of the code. It is mainly used to write single-line comments in languages such as C, C++, Java, JavaScript, and C#. Comments help improve code readability and make maintenance easier.

Table 12: An Example of curated QA data used in Student-Centered Fine Tuning.

Input Query:

```
N = int("abc")
F = int(input())
N = (N // 100) * 100
for i in range(100):
    if (N + i) % F == 0:
        print(f'{i:02}')
        break
```

Error Message:

```
ValueError: invalid literal for int() with base 10: 'abc'
```

```
Traceback (most recent call last):
  File "/content/temp_script.py", line 1, in <module>
    N = int("abc")
    ^^^^^^^^^^
ValueError: invalid literal for int() with base 10: 'abc'
```

This code suddenly stopped working. What is the problem?

Reference Answer

Main Issue Summary: The line N = int("abc") is invalid because "abc" cannot be converted to an integer.

Error Cause / Detailed Explanation: int("abc") raises a ValueError since "abc" is not a numeric string. The error occurs immediately at program start, so the loop and later code never run.

Fix / Improvement (Code Level): My suggestion is to initialize N with a valid number, or take in numeric value via input().

Revised Code Example

```
# Fixed version: replace "abc" with valid integer input
N = int(input("Enter N (number): "))
F = int(input("Enter F (number): "))

# Round N down to nearest multiple of 100
N = (N // 100) * 100

# Find the value that satisfies the condition
for i in range(100):
    if (N + i) % F == 0:
        print(f'{i:02}')
        break
```

Additional Comments (Testing and Cautions) When running the code, test with valid numeric inputs to confirm it works correctly. You can further improve the robustness of the program by adding extra validation (e.g., try-except statements).