Teach AI How to Code: Using Large Language Models as Teachable Agents for Programming Education

Hyoungwook Jin
jinhw@kaist.ac.kr
School of Computing, KAIST
Daejeon, Republic of Korea

Seonghee Lee
shlee@cs.stanford.edu
Stanford University
Palo Alto, CA, United States

Hyungyu Shin
hyungyu.sh@kaist.ac.kr
School of Computing, KAIST
Daejeon, Republic of Korea

Juho Kim
juhokim@kaist.ac.kr
School of Computing, KAIST
Daejeon, Republic of Korea

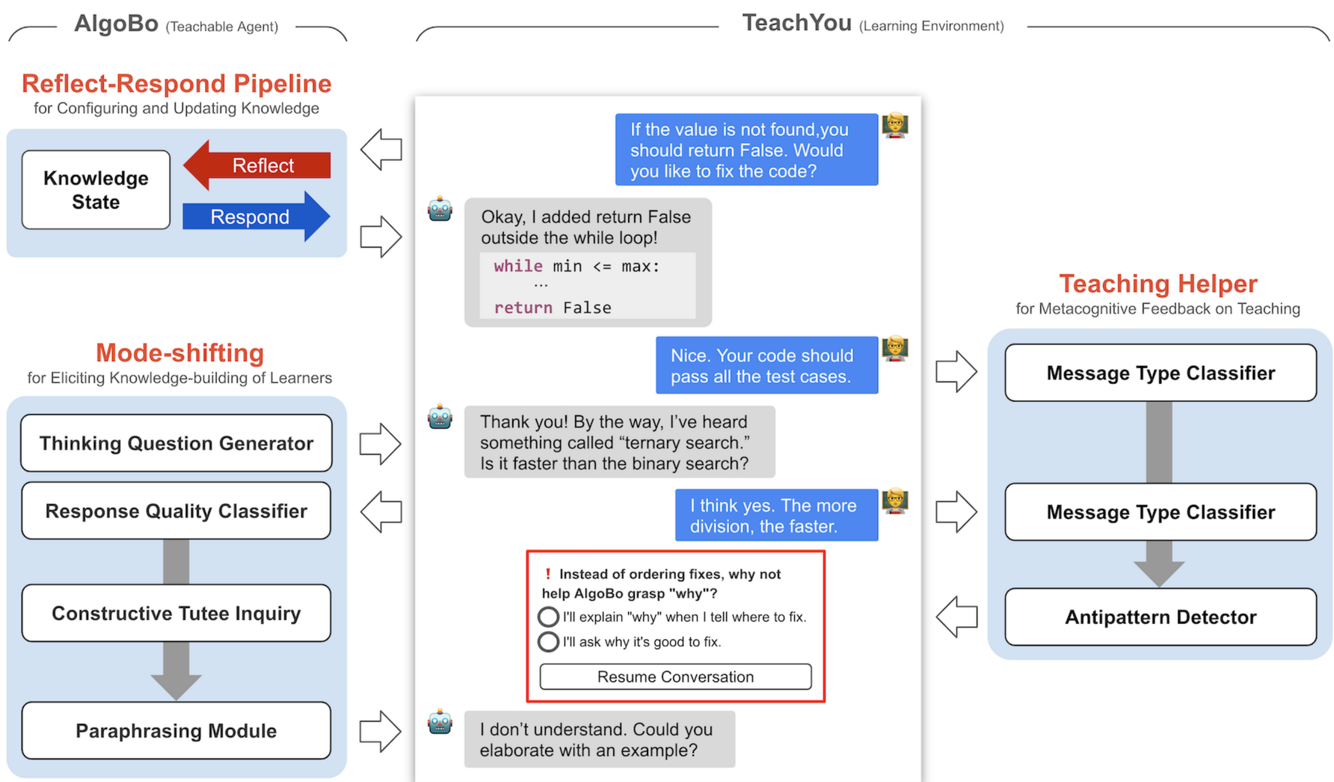

Figure 1: An overview of the core components of AlgoBo and TeachYou. The Reflect-Respond pipeline enables AlgoBo to create responses following its evolving knowledge state while Mode-shifting guides LBT conversations through knowledge-building questions that ask "why" and "how". The Teaching Helper in TeachYou analyzes conversations in real-time and gives metacognitive feedback and suggestions on teaching methods.

ABSTRACT

This work investigates large language models (LLMs) as teachable agents for learning by teaching (LBT). LBT with teachable agents helps learners identify knowledge gaps and discover new knowledge. However, teachable agents require expensive programming of subject-specific knowledge. While LLMs as teachable agents can reduce the cost, LLMs' expansive knowledge as tutees discourages learners from teaching. We propose a prompting pipeline that restrains LLMs' knowledge and makes them initiate "why" and "how"

CHI '24, May 11–16, 2024, Honolulu, HI, USA

ACM ISBN 979-8-4007-0330-0/24/05
https://doi.org/10.1145/3613904.3642349

questions for effective knowledge-building. We combined these techniques into TeachYou, an LBT environment for algorithm learning, and AlgoBo, an LLM-based tutee chatbot that can simulate misconceptions and unawareness prescribed in its knowledge state. Our technical evaluation confirmed that our prompting pipeline can effectively configure AlgoBo's problem-solving performance. Through a between-subject study with 40 algorithm novices, we also observed that AlgoBo's questions led to knowledge-dense conversations (effect size=0.71). Lastly, we discuss design implications, cost-efficiency, and personalization of LLM-based teachable agents.

CCS CONCEPTS

• **Human-centered computing** → **Interactive systems and tools**; • **Applied computing** → **Interactive learning environments**.

KEYWORDS

Human-AI interaction, LLM agents, AI and Education, Generative AI

ACM Reference Format:

Hyoungwook Jin, Seonghee Lee, Hyungyu Shin, and Juho Kim. 2024. Teach AI How to Code: Using Large Language Models as Teachable Agents for Programming Education. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24), May 11–16, 2024, Honolulu, HI, USA.* ACM, New York, NY, USA, 28 pages. https://doi.org/10.1145/3613904.3642349

1 INTRODUCTION

Interactive learning activities involve learners actively collaborating with peers or engaging with computer systems to deepen their comprehension of a specific topic [54, 87]. Compared to passive learning activities (e.g., reading text passages without doing anything else), interactive learning activities (e.g., pair programming, peer teaching) can elicit the deepest level of understanding by encouraging learners to elaborate their explanations and construct new knowledge on top of each other through conversations [15, 17, 18, 26, 82, 83]. One form of interactive learning is Learning by Teaching (LBT), where learners tutor a peer learner and exchange questions to reorganize their knowledge and identify knowledge gaps.

LBT with teachable AI agents (i.e., virtual tutees) can offer many advantages over LBT with humans. Teachable agents can bring scalability to LBT with their around-the-clock availability and motivate learners' participation in LBT by reducing psychological barriers, such as the fear of making mistakes while teaching and the pressure of responding in real-time [13, 20]. However, despite these benefits, disseminating teachable agents to diverse subjects is challenging in practice due to the effort-intensive authoring of the agents' knowledge model [48] and sophisticated behaviors [78] to elicit desired learning experiences beyond a tutoring simulation. Conventional authoring methods require extensive mapping of agents' knowledge states and high programming skills, precluding teachers and education researchers from tweaking teachable agents for their needs and context.

In this paper, rather than constructing teachable agents from the ground up, we propose a top-down methodology in which we use versatile Large Language Models (LLMs) to simulate tutees. Recent advances in LLMs show their remarkable capabilities in making contextual dialogues [63, 74], role mimicry [37, 46], and learning from demonstrations [12, 68]. Teachable agents equipped with the LLM capabilities can perform more believable and natural tutoring interactions (e.g., writing and explaining arbitrary code on request), compared to prior non-LLM LBT systems that adopted prescripted and limited interaction channels [7, 41, 51, 65]. The flexible interaction allows learners to formulate free-form questions and try diverse teaching methods, improving their knowledge construction and metacognition [2, 16, 75, 96]. We explore using LLMs to lower the cost and barriers of building teachable agents and to make LBT more engaging and pedagogically effective.

In our formative study, we asked 15 programming novices to conduct LBT with ChatGPT prompted to perform the role of a tutee. We found that there are needs for 1) confining the knowledge level of LLM agents, 2) agent-initiated "why" and "how" questions, and 3) in-conversation feedback on learners' teaching methods. Our dialogue analysis revealed that role-playing led learners to self-explain their knowledge but was limited to knowledge-telling, achieving only the rudimentary benefits of doing LBT. Participants struggled to build new knowledge because the teachable agent excelled in writing code even without being taught and did not ask questions that could prompt elaboration and knowledge-building. The participants also commented about the lack of metacognitive guidance and reflection for effective LBT.

To address these issues, we built a teachable agent, "AlgoBo", that can exhibit prescribed misconceptions and knowledge level and "TeachYou", an LBT environment for introductory algorithm learning (Fig. 1). In TeachYou, learners solve programming problems on algorithms (e.g., binary search) and reflect on them by teaching AlgoBo. As learners correctly teach AlgoBo, our Reflect-Respond prompting pipeline instructs AlgoBo to fix its misconceptions and write code based on what it is taught. We also added Mode-shifting, in which AlgoBo periodically shifts to a questioner mode and asks questions to prompt learners' elaboration and sense-making. Lastly, TeachYou has a Teaching Helper that provides metacognitive feedback and suggestions to learners on their teaching method in real-time through dialogue analysis.

We conducted a technical evaluation of our Reflect-Respond prompting pipeline to check if AlgoBo can simulate a tutee with a prescribed knowledge level on different algorithm topics. We found that the pipeline can effectively configure, persist, and adapt AlgoBo's knowledge level within a conversation. We also conducted a between-subjects study with 40 algorithm novices, where the participants studied binary search with either TeachYou or a baseline system without Mode-shifting and Teaching Helper. Our analysis of LBT dialogues and survey results showed that Mode-shifting improved the density of knowledge-building messages in the conversations significantly ($p = 0.03$) with an effect size (Cohen's d) of 0.71. Teaching Helper also helped participants reflect on their teaching methods and sequence their questions strategically, but we could not observe significant improvement in participants' metacognition.

We structured our paper in the following order. After a discussion of related work, we describe our formative study settings and preliminary findings. We then reorganize the findings into three design goals and introduce our system and pipeline for achieving the goals. With that, we present our technical and user-study evaluation results. Lastly, based on our results and observations, we discuss the design considerations for teachable agents, the benefits of using LLMs, promising directions for personalizing teachable agents, and interaction guidelines for better LBT with teachable agents.

This paper makes the following contributions:

- AlgoBo, an LLM-based teachable agent that uses the Reflect-Respond prompting pipeline to simulate prescribed learning behaviors and Mode-shifting to scaffold knowledge-building of learners through "why" and "how" questions.
- TeachYou, a web-based algorithm learning system that supports LBT with AlgoBo and provides metacognitive feedback on teaching based on real-time conversation analysis.
- A technical evaluation of the Reflect-Respond prompting pipeline and an empirical user study results with 40 participants showing that TeachYou improved knowledge-building in LBT.

2 RELATED WORK

We outline past studies on stimulating effective LBT among humans and using teachable agents. Previous research connects to our work in improving the quality and scalability of LBT using virtual agents.

2.1 Learning by Teaching

Learning by Teaching (LBT) is a teaching method in which learners not only articulate and restructure their existing knowledge but also engage in reflective knowledge-building. Knowledge-building refers to extending knowledge beyond provided materials to craft deeper explanations, analogies, and inferential connections [13, 15, 22, 71], leading to the deliberate creation and improvement of knowledge useful for a community in a broader context [77]. However, LBT alone does not elicit knowledge-building naturally [67, 91]; learners tend to end up in knowledge-telling, in which they verbalize what they already know [71]. Previous research investigated support for eliciting knowledge-building responses from learners. King et al. found that training learners to ask reviewing, proving, and thinking questions in sequence to peers during LBT can promote higher-order thinking and learning [36]. Roscoe and Chi's analysis of LBT dialogues showed the importance of the tutee's role in knowledge-building; the deep questions from the tutee encourage tutors to make self-reflective responses and create inferences between new and prior knowledge [72]. Shahriar and Matsuda also confirmed that tutees' follow-up questions drew the knowledge-building of tutors with low prior knowledge in particular [78]. Matsuda et al. found that LBT with metacognitive guidance for planning and conducting teaching is as effective as being tutored by experts regardless of learners' prior competency [52]. Our primary goal is to build an interactive system that draws knowledge-building from learners in LBT. To do so, we adapt the interventions mentioned above in human tutor-tutee interactions to the conversational interactions between virtual agents and learners.

2.2 Teachable Agents for LBT

A core component of LBT is the presence of a peer learner. However, as human learners cannot always be present, past research introduced teachable agents—virtual agents that can learn declarative and procedural knowledge from learners' explanations and demonstrations, taking the role of peer learners in LBT [9]. Teachable agents showed promising results in improving students' performance, self-explanation, and acceptance of constructive feedback [13, 25, 41, 53, 81]. LBT with early teachable agents was non-conversational; agents revealed their knowledge states as concept maps, and learners taught the agents by directly editing their knowledge states [8, 11]. Recent teachable agents conceal their states and simulate more authentic learning behaviors; agents can learn from the tutors' demonstrations [47], mimic the behaviors of learners (e.g., making arithmetic mistakes) [32, 65], improve with correct instructions [53], and ask questions [50]. However, implementing these natural and highly interactive teachable agents requires significant manual efforts and programming skills to specify and model the knowledge of agents [49]. For example, implementing an agent in SimStudent required more than a thousand lines of Java code for simple algebra equation solving [47]; the cost may increase exponentially for more complicated topics (e.g., algorithm learning, advanced equation solving). In this paper, we investigate using LLMs for building conversational teachable agents with low manual effort and programming barriers to support educators and researchers in adopting LBT in diverse classes and experiments.

2.3 LLM-powered Simulation of Tutoring

While the development cost and skill barrier have limited teachable agents to few learning activities in the past, LLMs can provide a more affordable method to simulate virtual students and coaches and to diversify their interactions [46, 60, 90]. GPTeach by Markel et al. [46] simulates role-plays between a teaching trainee and virtual students who come for office hours by leveraging persona and context setting in prompts. LLM-simulated students allow trainees to practice teaching with diverse students and to interact through conversations, perhaps the most familiar and open-ended form of teaching others. Likewise, LLM-based teachable agents can enrich tutor-tutee interaction and activities in LBT as learners can formulate free-form questions by themselves and try out different teaching strategies, as opposed to non-LLM LBT systems that permit only predefined methods to assess agents' knowledge (e.g., multiple choice questions) [7, 41, 52, 65]. Nevertheless, challenges remain in making these LLM-based agents suitable for LBT, where the agents should not only simulate tutoring but also proactively elicit learners' knowledge-building. Beyond the roles set by prompts, we need precise control of the teachable agents' cognitive behaviors (e.g., knowledge levels and question-asking) to facilitate the intended learning experience. Prior research has proposed LLM agent architectures and pipelines to grant and scope cognitive capabilities to LLM, such as memory [64, 98], role-playing [30, 66], and reasoning [14, 33, 43]. We extend the control on LLMs' cognitive capabilities by proposing an LLM prompting pipeline that restrains the knowledge level of LLM-based agents.

3 FORMATIVE STUDY

We ran a formative study to explore the difficulties of using an LLM as a teachable agent. We recruited 15 Python novices and asked them to teach the binary search algorithm to an LLM chatbot. We surveyed their learning experience and analyzed the quality of their dialogues with the chatbot by annotating the types of messages.

3.1 Participants and Procedure

We recruited 15 participants on campus who could read and write short (about 15 lines) Python programs containing `if` and `while` statements and who were not familiar with binary search and LBT. Eleven were from non-CS engineering departments.

The study consisted of three stages. In the first stage, the participants went through learning materials on the binary search from Khan Academy[1] and solved two Parsons problems, a coding exercise on reordering code fragments [21]. In the second stage, the participants received an introduction to the concepts of LBT, its expected learning benefits, and its procedures. Then, they were given a brief overview of the LBT activity they would be performing next. In the final stage, learners tutored the chatbot on how to write code for the two binary search problems from the prior stage. After the LBT activity, the participants completed an exit survey composed of questions on three themes: the perception of the chatbot as a learner, the self-perceived learning effects, and the familiarity with teaching a chatbot.

The participants interacted with a baseline LLM chatbot, AlgoBo, performing the role of a teachable agent. We used GPT-4 [62] as a backbone for AlgoBo and provided a system prompt (see Appendix A.1) that set a persona of a student and added predefined learning challenges it was running into to provide a more convincing teachable agent [46, 66]. Since we use the name "AlgoBo" again in our main system and evaluation, we use "AlgoBo-Basic" throughout this section to distinguish the two teachable agents we developed.

3.2 Dialogue Analysis

In addition to the comments from the exit survey, we also looked into the quality and conversational patterns of the dialogues between participants and AlgoBo-Basic by classifying messages into knowledge-telling and knowledge-building types.

Since previous taxonomies that categorize LBT dialogues [36, 71, 89] were not contextualized enough to programming tutoring, we decided to adapt the taxonomies and create a new taxonomy (Table 1) specific to LBT in programming. We created our initial set of message types based on the prior taxonomies for general LBT dialogues [36, 71, 89] and categorizations of programming QA [3, 39]. Three authors took three iterations to annotate dialogues, resolve conflicts, and refine the taxonomy [69, 94]. The authors finalized the taxonomy in the 2nd iteration (20 dialogues, 293 messages). The authors categorized the rest of the messages independently. The inter-rater reliability of the categorization was high; three authors achieved Krippendorff's alpha of 0.731 for the data in the last iteration (11 dialogues, 253 messages).

Our taxonomy has three main categories: instructions, prompting, and statements (see Table 1). **Instruction** messages have content that asks the opponent (usually the tutee) to do specific actions, such as fixing code and attempting problem-solving after concept understanding. Instruction messages are mostly related to the proceeding of steps in teaching. **Prompting** messages have intentions for eliciting specific actions from the opponent. These include asking a tutee about a specific concept of interest, giving thought-provoking questions to encourage knowledge-building, and asking a tutor for help. We designate Prompting-Thought-provoking to knowledge-building because such questions can signal collaborative knowledge-building where learners bring up exploratory questions and start knowledge-building discussions with agents. **Statement** messages are utterances explaining one's knowledge and opinions. Among them, Statement-Elaboration and Statement-Sense-making are knowledge-building as they are the artifacts of new knowledge; this corresponds to Roscoe and Chi's classification of knowledge-building activity [73].

3.3 Findings from Participants' Comments and Dialogue Analysis

We found that an LLM chatbot can serve as a teachable agent for rudimentary LBT. Participants were positive about teaching an LLM chatbot and felt it helped them reorganize and recall their knowledge. However, our dialogue analysis and in-depth survey responses revealed that the LLM chatbot fell short of adequately supporting learners' knowledge-building process.

AlgoBo-Basic was perceived as an overly competent learner due to its extensive prior knowledge and self-correcting behavior. Participants highly appreciated AlgoBo-Basic's ability to "talk like a real person and ask specific questions" (P14) for simulating a learner. However, two-thirds of participants commented that they experienced awkwardness due to AlgoBo-Basic's competence. AlgoBo-Basic initially started a conversation by asking for help. However, after a few chats, AlgoBo-Basic provided competent responses too quickly, which did not reflect a novice learner's learning process. P5 remarked, "I explained it very simply, but he understood it very well... He is so much smarter than me. He seems to fill by himself the knowledge even I am not sure about." AlgoBo-Basic's adeptness in code writing and explanation also limited conversational patterns and confused learners about their roles. AlgoBo-Basic made twice as many knowledge statements (i.e., Statement-Comprehension) as participants did, taking away the chance for learners to self-explain and teach (see the Statement-Comprehension row in Table 2). P7 stated, "AlgoBo-Basic was like a teaching assistant testing a student's ability, rather than a student struggling with binary search problems." Participants responded that they would have liked to see more student-like interactions from AlgoBo-Basic such as "asking more proactive questions" (P1) and "making mistakes and requesting tutors for an elaborated explanation" (P5).

Dialogues between tutors and AlgoBo-Basic were limited to only knowledge-telling. Participants valued retelling of their knowledge—"Writing down knowledge was very helpful in organizing knowledge. If you want to teach someone, you should create steps in your head, and this process helped a lot" (P1). However, their learning was limited to knowledge-telling; out of 546 messages, we could observe 244 knowledge-telling messages but only

[1]https://www.khanacademy.org/computing/computer-science/algorithms/binary-search/a/binary-search

Table 1: Our taxonomy to classify the type of messages in LBT conversations with a teachable agent. The bold texts in the example column are the examples of respective message types. The types with * are knowledge-telling responses. The types with ** fall into knowledge-building responses.

Category	Sub Category	Explanation	Example
Instruction	Fixing*	[Instruct to] correct specific knowledge or part of code.	Tutee: Here is my code: <code> **Tutor: Call the input() function twice so that N and K are separately taken as input.**
	Commanding	["] do simple actions irrelevant to learning. (e.g., simply combining code for a submission).	Tutee: I have written the binary search function. **Tutor: Now, write the entire Python code.**
	Encouraging	["] retry a previous action with emotional encouragement.	**Tutor: You are in the right direction. Keep writing more code.**
Prompting	Challenge -finding	[Prompt the opponent to] explain his struggles to find the parts to help.	**Tutor: In which part are you facing difficulties?** Tutee: I am struggling with writing the conditionals inside the while loop.
	Hinting*	["] think about alternative/specific approaches.	Tutee: I could not complete this part of the code. **Tutor: Well, have you considered the case when the number is equal to K?**
	Checking	["] show or self-explain his understanding of specific knowledge.	**Tutor: Do you know what binary search is?** Tutee: Yes! Binary search is …
	Thought-provoking**	["] elaborate previous explanations or think beyond the content of the given learning materials.	**Tutor: What will happen if we switch the min / max updating code?** Tutee: I haven't thought about it. Will the loop run forever?
	Asking for help	["] analyze the speaker's problem or give hints.	**Tutee: Could you help me with solving the problem, please?**
Statement	Comprehension*	[State one's knowledge or opinion by] paraphrasing / copying / explaining the learning material or the opponent's response.	**Tutor: First, let's define the function called binary_search. In the while loop, …**
	Elaboration**	["] providing extended clarification or relevant examples beyond the given materials.	Tutee: Can you think of a real-life example where we can use binary search? **Tutor: I think we can use it for finding a word in a dictionary where words are listed alphabetically.**
	Sense-making**	["] realizing own errors / misconceptions or making new inferences / connections to prior knowledge.	Tutor: Can you take a closer look at the else statement in your code? **Tutee: Ah, I got it. Let's modify the high value to mid. Here is the corrected code.**
	Accepting / Reject	["] agreeing or disagreeing with the opponent's response.	Tutor: You should update line 24 to … **Tutee: I think that is a good idea.**
	Feedback	["] responding to the opponent's action or thought.	**Tutor: Yes, that is exactly right.**
Miscellaneous		Greetings/goodbyes, social expressions	Tutor: Do you have any questions? **Tutee: No, thank you so much for your guidance so far!**

15 knowledge-building utterances (Table 2). Despite helping reorganize knowledge, self-explanations did not lead to building new knowledge beyond what they previously knew—"I didn't discover anything new because I explained what I had already learned" (P4). Furthermore, tutors' self-explanations were often undeveloped because AlgoBo-Basic did not ask questions on participants' vague explanations, and AlgoBo-Basic performed well. For example, P15 answered AlgoBo-Basic's question on why the input array needs to be sorted: "Sorted arrays reduce the number of calculations and maximize the effectiveness of binary search." Despite the lack of detailed reasoning (e.g., "how" and "why"), AlgoBo-Basic accepted the explanation and moved on to the next question.

Table 2: The distribution of message categories for 31 dialogues from the formative study. The types with $*$ are knowledge-telling messages. The types with $**$ fall into knowledge-building messages.

Category	Sub Category	Tutee	Tutor	Total
Instruction	Fixing*	0	37	37
	Commanding	0	65	65
	Encouragement	0	1	1
Prompting	Challenge-finding	0	18	18
	Hinting*	1	12	13
	Checking	1	31	32
	Thought-provoking**	0	1	1
	Asking-for-help	91	0	91
Statement	Comprehension*	133	61	194
	Elaboration**	0	1	1
	Sense-making**	12	1	13
	Accepting	35	4	39
	Feedback	0	17	17
Miscellaneous		19	5	24
Total		292	254	546
Knowledge-telling		134	110	244
Knowledge-building		12	3	15

Participants carried out antipatterns of LBT and sought feedback. Participants remarked tutoring through natural language communication was intuitive and familiar because it resembled tutoring humans, and they could apply the same teaching methods to AlgoBo-Basic. However, some participants wanted to see better methods for them to teach AlgoBo-Basic (P9) and a method to review their learning process (P15). P15 said, "I was able to see that my teaching skills worked, but the reflection [on my tutoring session] left a lot to be desired due to the lack of feedback on my teaching method" (P15). While analyzing participants' dialogues, we found common conversational antipatterns that may restrain the benefits of LBT. The first pattern was **Commanding**, in which participants repetitively gave AlgoBo-Basic specific instructions for writing and correcting code (Appendix B (A)). This pattern lacks an explanation of "why" and "how" which can prompt learners to go beyond recalling facts (i.e., knowledge-telling). The second pattern was **Spoon-feeding**, in which participants give away knowledge without questions to check or prompt a tutee's understanding (Appendix B (B)). Rather than passive explanations, learners can actively construct new knowledge by making thinking questions for their tutees, taking the benefits of having interactive agents. The last pattern was **Under-teaching**, in which AlgoBo-Basic progressed in problem-solving but did knowledge-telling only because learners did not attempt to teach and develop further knowledge. (Appendix B (C)).

4 DESIGN GOALS

The findings from our formative study showed that LLMs could serve as a rudimentary teachable agent for LBT. However, we also confirmed the need to improve LLM chatbots' imitation of help-seeking tutees, promote the knowledge-building of learners, and support learners' metacognition in teaching. Based on the insights, we set three design goals.

D1. Design teachable agents that can simulate misconceptions and gradual learning curves. We found that the pre-trained knowledge and self-correcting behavior of LLMs made AlgoBo feel less like a tutee and prevented tutors from learning by identifying tutees' errors and enlightening them with elaborate explanations [89]. To reduce undesirable competence, we need to control the prior knowledge of LLMs and make them show persistent misconception and unawareness in their responses until they receive pedagogical aid.

D2. Develop conversations through teachable agents' elaboration questions. AlgoBo rarely asked follow-ups or thought-provoking questions (Table 2) that can elicit learners' knowledge-building through elaboration and sense-making [36, 78]. Instead of keeping AlgoBo as a passive tutee throughout a conversation, switching AlgoBo into an active questioner may scaffold learners to build knowledge step by step. For example, AlgoBo can start as a help-seeking tutee asking learners to self-explain basic concepts, and it can turn into a thought-provoking Socratic questioner intermittently.

D3. Provide learners with metacognitive feedback on their teaching methods during a conversation. Besides the scaffolding from the tutee side, learners can actively improve the quality of their LBT by leading constructive tutoring and refraining from the three antipatterns. In our formative study, however, participants found guidance and reflection on their teaching techniques lacking. Metacognitive feedback on teaching during LBT can help learners recognize how to improve their teaching methods on the spot and refine conversations throughout.

5 SYSTEM

We present TeachYou, an LBT system featuring AlgoBo, an LLM-based teachable agent. AlgoBo gets help from learners to solve introductory algorithm problems while asking thought-provoking questions that encourage the learners to expand their knowledge beyond their current level. Through the system, we propose 1) a new LLM prompting pipeline for simulating tutees of specific levels of knowledge and misconceptions and 2) a learning environment for learners to effectively conduct LBT.

Programming and algorithm learners can use TeachYou to review what they learned and explore further knowledge through

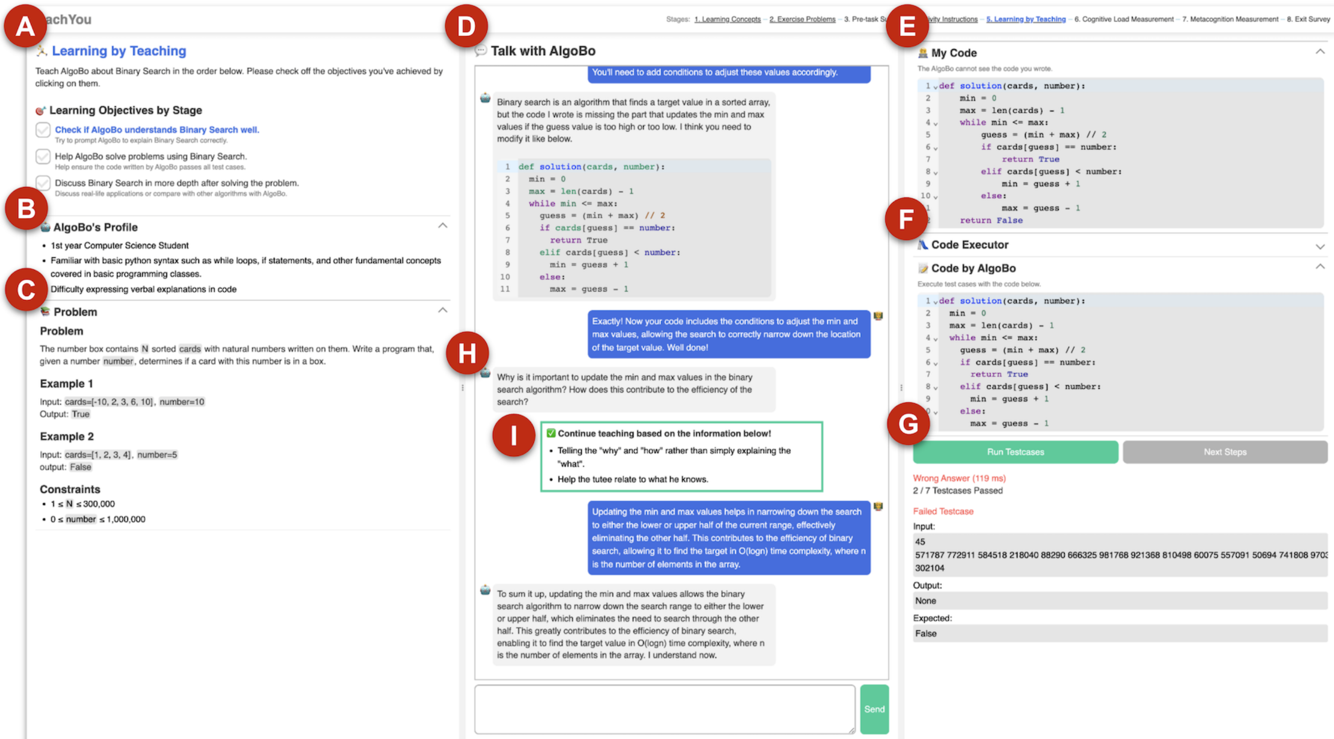

Figure 2: To the left, the 3 learning objectives they need to reach (A), learners can see AlgoBo's profile (B), and the questions they need to help AlgoBo solve (C). To the right, they can see the code they submitted (E), a code playground (F), and the code that AlgoBo write (G). When AlgoBo wrote code, participants could click on the "run test cases" and run AlgoBo's code. In the middle (D), learners use a typical chat interface to teach AlgoBo while receiving questions (H) and guidance from Teaching Helper (I)

an engaging and interactive LBT activity. We designed an interface (Fig. 2) to help learners conduct the activity. Throughout the LBT activity, learners should achieve three sequential objectives in teaching AlgoBo (Fig. 2 A). The objectives correspond to the three levels in Bloom's taxonomy (Understand-Apply-Analyze) [10, 38]; learners first check if AlgoBo correctly understand the concept of interest; then, learners help AlgoBo apply the concept to solve a problem; lastly, learners and AlgoBo discuss real-life use cases and other related topics. Learners can refer to the profile of AlgoBo to set their attitude and expectations (Fig. 2 B). We set the persona of AlgoBo as a 2nd-year high school student, as opposed to a 1st-year CS student in the formative study, to match the slow learning behavior and to encourage learners' patience in teaching. Learners use a typical chat interface to teach AlgoBo (Fig. 2 D) and have access to teaching support (Fig. 2 C, E, F, G). While tutoring, learners receive why questions and thought-provoking questions from AlgoBo, helping them self-explain the rationale behind their instructions and expand their knowledge (Fig. 2 H). TeachYou also provides feedback on learners' teaching methods and suggestions for improvement to encourage reflection on teaching (Fig. 2 I).

In order to support the aforementioned learning scenario and the three design goals effectively, we implemented three system components: First, we implemented the Reflect-Respond prompting pipeline for a teachable agent to simulate student-like learning behavior. Secondly, within a conversation, our teachable agent shifts between help-receiver and questioner modes in every third conversation turn, eliciting self-explanation and knowledge construction, respectively. Lastly, the learning environment analyzes the dialogue between learners and AlgoBo and provides feedback on their tutoring methods to promote metacognition.

5.1 Reflect-Respond prompting pipeline to simulate knowledge learning

From our observations and user comments in the formative study, we considered three properties crucial for LLM-based teachable agents to simulate knowledge learning—reconfigurability, persistence, and adaptability. **Reconfigurability** refers to how precisely we can set an agent's performance in question-answering and problem-solving. Reconfigurable agents allow us to build tutees with specific misconceptions and help design tutoring scenarios. **Persistence** examines how the knowledge level of a teachable agent on a target topic is maintained consistently throughout the agent interaction. Persistent agents do not self-correct their misconceptions and show constant question-answering performance unless being taught; their knowledge level should also not be susceptible to messages irrelevant to the knowledge of interest (e.g., jokes).

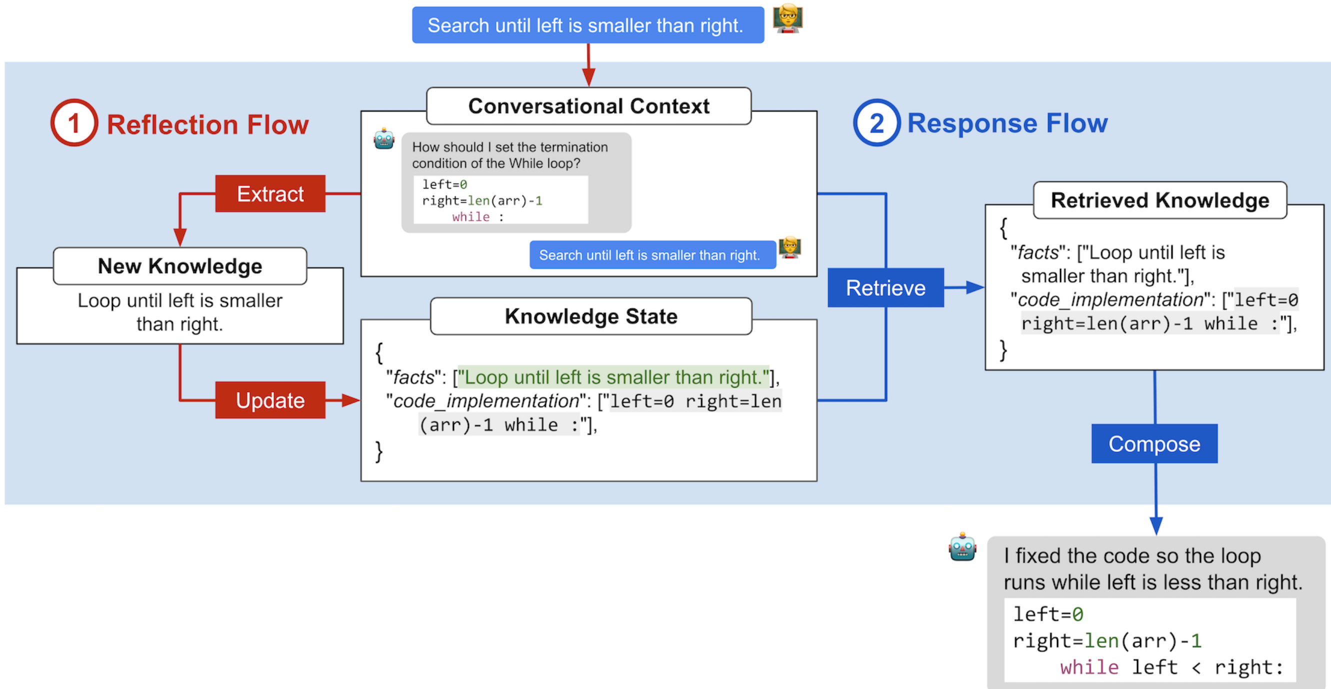

Figure 3: The overview of the Reflect-Respond prompting pipeline for simulating knowledge learning of AlgoBo and examples for each component. From the recent conversation, AlgoBo *extracts* new knowledge of the while loop condition and *update* its knowledge state (colored in green). Then, AlgoBo *retrieves* knowledge relevant to while loops and *composes* a response that fills its knowledge gap.

Adaptability measures how well the agent updates its knowledge as it acquires new information from tutors in conversations. Adaptability allows a teachable agent to improve its knowledge level and remember what tutors have taught.

To achieve these properties, we introduce a prompting pipeline that leverages a knowledge state and two information flow mechanisms: Reflection and Response (Fig. 3). A **knowledge state** is a store representing the knowledge AlgoBo currently holds. It is comparable to a schema, a cognitive unit of knowledge for problem-solving [84]. AlgoBo's responses are constrained by its knowledge state, and we update the knowledge state consistently throughout a conversation. Knowledge states link to reconfigurability; if we leave them empty, agents will show zero-knowledge behavior; if we add incorrect or correct information, agents will show misconceptions or prescribed knowledge levels, respectively. **Reflection** is a flow dedicated to the update of knowledge states. In the Reflection flow, we use an LLM to extract new information from the latest conversations (i.e., the last three messages) and then update knowledge states by adding or correcting information. After Reflection, the **Response** flow occurs; we first use the LLM to retrieve information relevant to the conversational context from the current knowledge state and then compose a response by only combining the retrieved knowledge. If a knowledge state does not have relevant information and nothing is retrieved, AlgoBo responds: "I'm not sure how to do that. Could you explain it to me?" Reflection and Response connect to the persistence and adaptability of agents as the flows control the retrieval and update of knowledge states in reaction to external stimuli.

We implemented the knowledge state as a JSON object with two attributes: facts and code_implementation. `Facts` store natural language explanations of the target knowledge. `Code_implementation` contains code snippets (see Fig. 3 knowledge state). The four operations in the pipeline are implemented with GPT-4 as a base LLM. We adopted well-known prompting engineering techniques, such as AI chains [92], few-shot prompts [12, 86], persona setting [46, 66], and code prompts [23, 97] (see Appendix A.2). We note that our implementation is one possible instance of our proposed pipeline, and it can improve further with better LLMs and algorithms for the operations. For example, we can represent knowledge states with more complex tree structures [44, 95], and the update operation may use the Least Recently Used algorithm [61] to simulate a fixed-size knowledge capacity. We chose GPT-4 for operating our pipeline because it can effectively process the contextual information in conversations compared to other approaches.

5.2 AlgoBo's Mode-shifting to develop constructive LBT dialogues

Beyond telling knowledge to AlgoBo, we aim to push learners to answer thought-provoking questions and build new knowledge. From the formative study, we observed that entrusting LLMs entirely with making conversations did not result in desirable knowledge-building patterns (e.g., question-answering on "why" and "how")

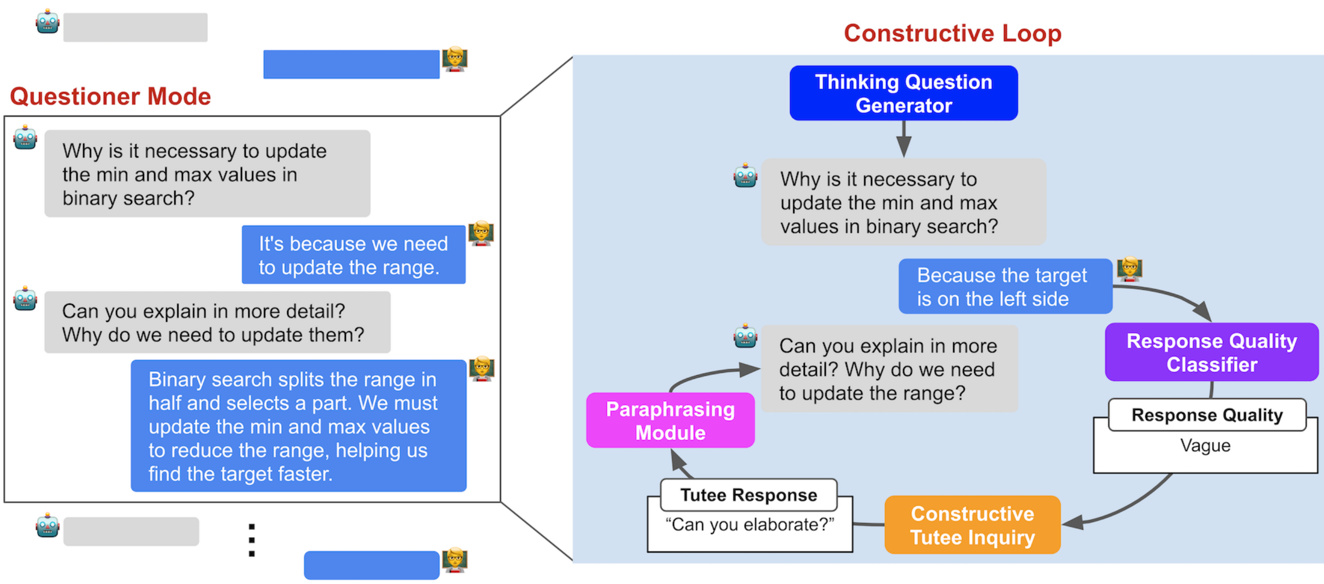

Figure 4: AlgoBo shifts its mode in every three messages. When AlgoBo is in the questioner mode, it keeps asking follow-up questions until receiving a satisfactory response (constructive loop)

spontaneously. Prior research has shown that guiding questions from conversational agents are effective for improving learners' knowledge-building and divergent thinking [1, 78]. To control conversation flows while giving learners the freedom to steer them, we introduce **Mode-shifting**, in which AlgoBo periodically shifts between two modes: In the help-receiver mode, AlgoBo passively learns from tutors and prompts their self-explanations; in the questioner mode, AlgoBo asks thought-provoking questions to stimulate the knowledge-building of learners.

We use Mode-shifting to make conversation flows dynamic and engaging. In every third message, AlgoBo shifts to the questioner mode and asks a thinking question. The thinking question differs by the phase of the activity (Fig. 2 A). While learners teach AlgoBo about concepts and code implementation (i.e., the first and second objectives), AlgoBo asks "why" questions in response to learners' instructions and explanations. During the discussion phase (i.e., the third objective), AlgoBo brings up related algorithms or real-life examples and asks "how" questions to prompt learners to explain and connect to what they have learned. After the thinking questions, the conversation goes through a constructive loop, in which learners receive follow-up questions from AlgoBo until they answer the question in depth with a valid example. When AlgoBo assesses learners' responses as satisfactory, AlgoBo summarizes them and shifts back to the receiver mode. The period of Mode-shifting (every three messages) is heuristic; from our pilot studies, we found that such frequency was optimal for prompting elaboration while not distracting tutors too much.

To incorporate Mode-shifting to LBT dialogues, we implemented four components (Fig. 4). The **Thinking Question Generator** is a module that uses GPT-4 to produce thought-provoking questions related to the current conversation. For managing the constructive loop, we followed the protocol of the constructive tutee inquiry in Shahriar et al.'s work [78] and adapted it to LLM. We used the formative study dialogues with response quality annotations to train the **Response Quality Classifier**. The classifier assesses every learner's responses in the loop and determines AlgoBo's follow-up question as pre-defined in **Constructive Tutee Inquiry** protocol [78]. Lastly, the **Paraphrasing Module** adjusts the fixed question to the conversational context. All the prompts used for Mode-shifting are available in Appendix A.3.

5.3 Teaching Helper for Metacognitive Guidance

Throughout our formative study, we found conversational antipatterns that hindered effective LBT. To prevent this, TeachYou provides metacognitive feedback throughout the conversation to help learners reflect on the overall teaching session and offer overarching guidance on steering the discussion. TeachYou presents the feedback through **Teaching Helper,** a red or green text box that appears below the messages (see Fig. 2 I). Teaching Helper provides information on the current problems with the teaching method and elaborates on what learners could do to improve their conversation.

TeachYou provides four Teaching Helper messages, depending on detected conversational patterns (Fig. 5). For the **Commanding** and **Spoon-feeding** patterns, in which learners should correct their teaching styles, TeachYou shows feedback messages in red boxes. To ensure learners read feedback, we interrupt the conversation with AlgoBo until learners explicitly decide how to act. The send button in the chat interface is blocked until learners pick an option among the possible teaching methods to address the issue. We chose to give learners multiple suggestions and let them choose their teaching method, instead of giving specific guidance to follow because the active selection of teaching methods may improve learners' recognition and autonomy in tutoring [93]. For the **Under-teaching**

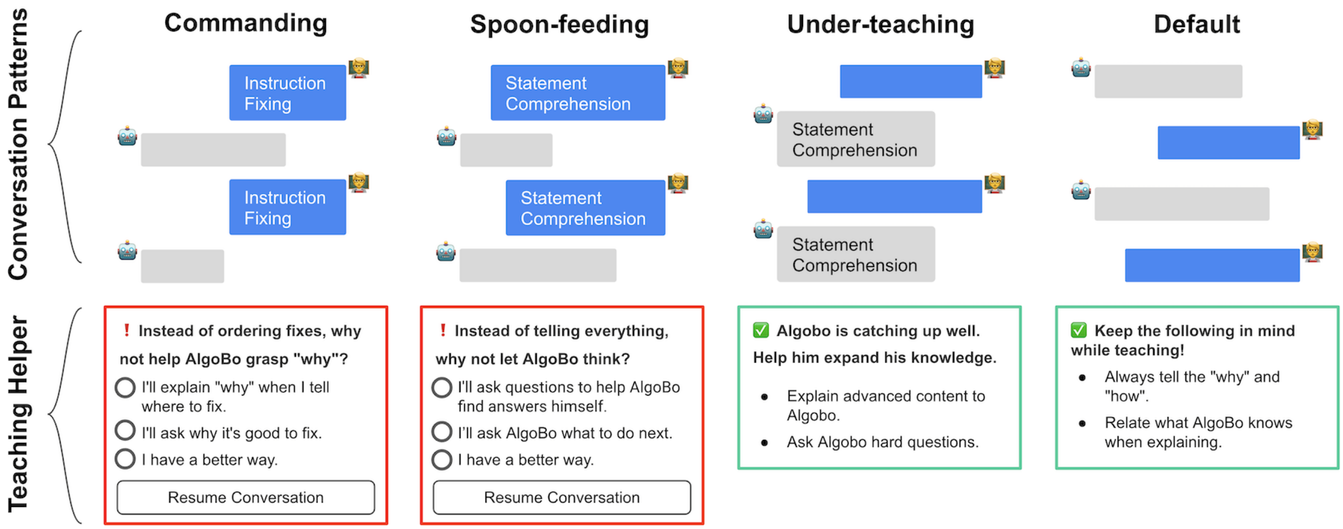

Figure 5: The four Teaching Helper messages and corresponding suggestions that appear depending on the conversational patterns.

pattern and default cases where no antipattern is found, TeachYou shows messages in a green box. The messages either encourage learners to go beyond the current learning topic or give general tips for good answering and questioning [6, 34, 35]. Teaching Helper messages and learners' selection remain in conversations for revisiting. To avoid frequent interruptions and distractions from Teaching Helper, we restrict the presentation of the feedback to every six messages.

Teaching Helper is powered by a message-type classifier for detecting conversational patterns. We used the dialogue dataset from the formative study to fine-tune the GPT-3 davinci model. For training, we used 438 messages, and the classifier achieved an accuracy of 71.3% for the remaining 108 messages in a validation test.

6 EVALUATION

We evaluated the efficacy of TeachYou for eliciting knowledge-building experiences in LBT. This overarching goal broke down into three main research questions:

RQ1. How well does the Reflect-Respond pipeline simulate misconceptions and knowledge development?
RQ2. How does TeachYou help elicit knowledge-building in LBT conversations?
RQ3. How does TeachYou improve learners' metacognition about tutoring?

The evaluation was divided into two parts. The initial phase was a technical evaluation that aimed to assess if the Reflect-Respond pipeline could induce a teachable agent to produce responses that were reconfigurable, persistent, and adaptive throughout the course of a conversation (RQ1). In the second phase, we ran a user study to examine the effects of Mode-shifting and Teaching Helper on learning experiences (RQ2 and RQ3).

6.1 Technical Evaluation of the Reflect-Respond Pipeline

As defined in Section 5.1, we evaluated the responses generated by our prompting pipeline along three axes—reconfigurability, persistence, and adaptability (RQ1).

6.1.1 ***Evaluating AlgoBo's Knowledge Level.*** We evaluated AlgoBo's knowledge level by observing its performance on Multiple Choice Questions (MCQs) under varying knowledge states and conversational interactions. Although our target learning setting does not involve MCQs, we chose MCQs to follow prior research on assessing LLMs' performance [23, 70] and collect clear-cut results. A well-configured teachable agent should only perform well on the MCQ questions that can be answered with the given information in the knowledge state. To confirm that AlgoBo was answering questions based on its knowledge state only and not picking random choices, we also prompted AlgoBo to explain why it chose the answers (Fig. 6).

6.1.2 ***Procedure and Setup.*** We measured AlgoBo's MCQ performance on three different algorithmic topics. For each topic, we created a set of nine MCQs (Appendix C.1). Within each set, we had three MCQs for each of Bloom's taxonomy categories: Understanding, Implementation (Applying), and Analysis [10, 38]. Understanding questions consisted of questions on factual concepts, Implementation questions were about filling in the blanks in code, and Analysis questions were about the time complexity calculation and comparison to other relevant algorithms. AlgoBo was evaluated with 4 different knowledge states (Appendix C.2) and conversational inputs (Appendix C.3, C.4, C.5).

For reconfigurability (i.e., the change in knowledge level with different knowledge states), we prepared four seed knowledge states (Appendix C.2). *State 1* was empty to simulate zero knowledge. *State 2* had an explanation of a topic algorithm in only `facts` to observe if AlgoBo knows only the given information. *State 3* had the same

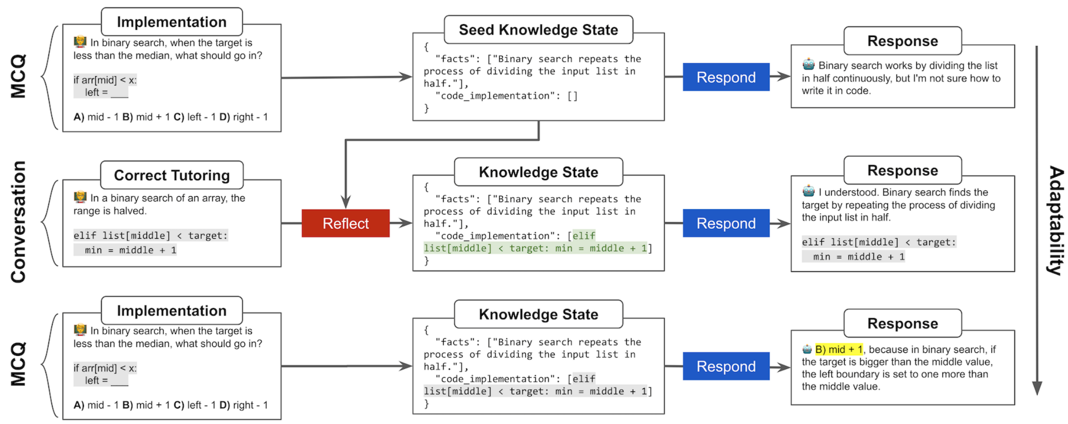

Figure 6: The process of measuring adaptability for correct tutoring with an Implementation problem and *State 2* as a seed knowledge state. The evaluations were performed in Korean to ensure compatibility with the main study conditions.

explanation plus a piece of incorrect code in `code_implementation` to check if AlgoBo shows the prescribed misconception. *State 4* had the correct explanation and code to see if AlgoBo becomes competent with more input knowledge. We prompted AlgoBo to solve MCQs with different seed knowledge states and compared the scores between the states. To prevent AlgoBo from storing knowledge learned from the MCQs into its knowledge state, we turned off the Reflection flow.

For assessing persistence (i.e., the invariance of knowledge level under no stimuli), we ran random conversations on *State 2*. In the random conversations, AlgoBo was taught irrelevant information, such as arithmetic, translation, and classification, thus leading AlgoBo to save random information in its knowledge state [79]. We turned on the Reflection flow so that AlgoBo could update its initial knowledge state. We prompted AlgoBo to solve the same MCQs again and compared the difference between the first and second scores.

For adaptability (i.e., the acceptance of new knowledge), we considered two cases—Correct and Incorrect tutoring. The performance gap between Correct and Incorrect tutoring is crucial to check an agent's suitability for LBT because a teachable agent should not excel when learners give incorrect or incomplete instruction. Tutoring conversations taught three pieces of information that mapped to Understanding, Implementation, and Analysis of concepts. Correct tutoring gave AlgoBo correct factual information, whereas Incorrect tutoring provided false information. We ran Correct and Incorrect tutoring separately on AlgoBo configured with *State 2* and compared the differences between the MCQ scores at the start and after each type of tutoring.

We used the GPT-4-0613 model with 0 temperature throughout the evaluation. For a more comprehensive understanding of the four knowledge states and the materials used in the evaluation, please refer to Appendix C.

6.2 Technical Evaluation Result

We report the result of the technical evaluation on reconfigurability, persistence, and adaptability. We observed a small variation in the MCQ score even for the same inputs, knowledge states, and LLM model, perhaps due to the randomness inherent in the model and the running hardware[2]. We repeated the entire measurement five times for each input configuration and reported the median score for each question to handle variances of AlgoBo's response. The variance in score was mild; on average, AlgoBo produced a different response once in five repetitions. For the detailed report of the variance, refer to Appendix C.6.

[RQ1] Response flow can effectively reconfigure the knowledge level of AlgoBo.

As expected, AlgoBo got all MCQs wrong when its knowledge state was empty (see *State 1* in Table 3). When the knowledge state had only `facts` information (*State 2*), AlgoBo could solve some conceptual (Understanding and Analysis) questions but none of the Implementation questions. This shows that separating the knowledge state by knowledge types (`facts` and `code_implementation`) can help configure knowledge more precisely by types. When the knowledge state contained code information, AlgoBo started to solve Implementation questions and achieved higher scores when given correct code (*State 4*), compared to incorrect code (*State 3*). AlgoBo followed what was written in its knowledge state (*State 3*) exactly and produced wrong code and answers.

[RQ1] Reflect-Respond makes AlgoBo produce responses persistent to knowledge states.

The random conversation had a mild effect on the MCQ scores (compare the difference between the "At the start" and "After random conversation" columns in Table 4). While random conversation changed the scores of conceptual questions, the scores of Implementation questions stayed the same. We analyzed the inputs and outputs of the Respond flow in depth and found that AlgoBo retrieved algorithm-related knowledge that was missing in the first MCQ solving. Considering our LLM prompt for Retrieve (Appendix A.2 Retrieve), we contemplate that the population of more information in knowledge states might increase the relative importance of relevant knowledge in retrieval and help AlgoBo solve

[2]https://community.openai.com/t/a-question-on-determinism/8185/2

questions correctly. In other words, the scores after the random conversation are closer to what the AlgoBo should have received initially. To see how far the population of random information increases knowledge level, we ran another random conversation and checked MCQ scores (see Table 5 Scenario 1). The second random conversation contained random statements on arithmetic, translation, and classification (Appendix C.3). We did not observe any significant increase in the scores, confirming that the persistence of knowledge levels is robust regardless of the length of random conversations.

[RQ1] Reflect-Respond allows AlgoBo to adapt knowledge states from conversations.
Correct tutoring significantly improved MCQ scores (compare the difference between the "At the start" and "After Correct tutoring" columns in Table 4) across Understanding, Implementation, and Analysis. Conversely, the Incorrect tutoring improved MCQ scores (compare "At the start" and "After Incorrect tutoring" columns in Table 4), but not as much as the Correct tutoring did. For example, the incorrect code information "`if arr[mid] > x: low = mid + 1 elif arr[mid] < x: high = mid - 1`" given in Incorrect tutoring stimulated AlgoBo to infer that "Binary search returns a value indicating not found if the target is not in the list" and solve one of the Implementation questions. This result shows that partially correct information in the Incorrect tutoring could help solve problems, suggesting the need for more precise control in writing the knowledge states.

To investigate if AlgoBo prefers correct information to incorrect information and if incoming knowledge tends to overwrite pre-existing knowledge, we ran two scenarios in which AlgoBo received Correct and Incorrect tutoring in a sequence (see Table 5 Scenario 3 and 4). The result shows that AlgoBo tends to keep correct information and remove incorrect ones (check Table 6 last knowledge state). We surmise that AlgoBo dropped conflicting information to keep its knowledge state short as instructed in the Update prompt (Appendix A.2). We also speculate that LLMs prefer to follow widespread (often factual) knowledge compared to incorrect information as the way it is trained [58].

6.3 User Study

We also ran a user study to evaluate the usefulness of Mode-shifting and Teaching Helper in improving the learning experience (RQ2 and RQ3). We designed a between-subjects study to check the usefulness of our system components. In the *Baseline* condition, participants used the version of TeachYou without Mode-shifting and Teaching Helper. The participants in the *TeachYou* condition used the complete version of TeachYou as described in Section 5. The Reflect-Respond pipeline instructed AlgoBo in both conditions. We did not have separate conditions for Mode-shifting and Teaching Helper because we assumed the interaction between them would be insignificant as they support different aspects of learning (i.e., knowledge-building and metacognition).

6.3.1 ***Participants.*** We recruited 40 participants through advertisements on the campus community websites (age=24 ± 4.0, 25 males and 15 females). Participants were required to understand short (about 20 lines) Python programs that contain basic syntax such as `if` and `while` statements, and we excluded those who participated in the formative study. To cap participants' prior knowledge, we filtered out the participants who were assumed to have mastered binary search already. We collected applicants' confidence in understanding binary search and teaching it to others on a 7-point Likert scale, the last time coding binary search, and their paid teaching experience on programming. We also asked applicants to solve six Understanding and Implementation MCQs about binary search (Appendix C.1). We filtered out the applicants who met three or more of the following criteria: 1) scored five or more in the MCQs, 2) rated six or more for confidence, 3) implemented binary search within the last six months, and 4) were paid for teaching. We randomly assigned 20 participants to each condition—*Baseline* and *TeachYou*. We did not observe any significant differences between conditions in the initial self-rated understanding of binary search (*Baseline*=4.40 ± 1.35, *TeachYou*=4.25 ± 1.65, two-tailed t-test, $p = 0.76$) and the time to solve the exercise problem during our study (*Baseline*=116 ± 60 sec, *TeachYou*=124 ± 62 sec, two-tailed t-test, $p = 0.66$).

6.3.2 ***Procedure and Materials.*** The user study was run online; after submitting informed consent, the participants received an online link to our system and completed the study in their available time. Participants spent 60 ± 25 minutes on average to complete the study and were paid 25,000 KRW (i.e., approximately 18.5 USD). All the instructions and materials used in the study were translated into Korean to avoid any language barrier and unnecessary cognitive overhead.

The study procedure was organized into three parts (see Table 7). In the first part, participants learned about binary search and how to implement it in Python. Participants read the lecture materials on binary search taken from Khan Academy[3] (Step 1) and solved an exercise problem in the form of a Parsons problem [21] (Step 2). After the exercise, participants wrote about their strategies in teaching (if any) and their prior experience in using AI chatbots, such as ChatGPT and Bing search (Step 3).

In the second part, participants conducted LBT with AlgoBo. We provided explanations about LBT, the profile information of AlgoBo, and the participants' objectives for the LBT activity (Step 4). We stated in the objectives that participants should not only help AlgoBo solve the exercise problems but also construct new knowledge for themselves, encouraging the participants to pursue knowledge-building. Then, participants taught different versions of AlgoBo and TeachYou according to their conditions (Step 5) with the interface shown in Fig. 2. AlgoBo was configured by our prompting pipeline, and the seed knowledge state was identical across the conditions. The `facts` field of the seed knowledge state was empty to simulate a lack of understanding, and the `code_implementation` field had a basic code structure that lacked the entire range update logic in binary search. We did not go for zero-knowledge AlgoBo to keep the entire teaching sessions within 40 min and spare enough time for having discussions. All the participants were given three goals to achieve in series; we asked them to 1) check if AlgoBo understands binary search first, then 2) help AlgoBo solve the exercise problems, and 3) discuss with AlgoBo about binary search in depth.

[3]https://www.khanacademy.org/computing/computer-science/algorithms/binary-search/a/binary-search

Table 3: The number of correct MCQs for different knowledge states. *State 1* is an empty knowledge state; *State 2* has facts only; *State 3* has facts with wrong code; *State 4* has facts and correct code. "U", "I", and "A" stand for Understanding, Implementation, and Analysis question types. The number in each cell ranges from zero to three as there were three MCQs for a particular question type.

	State 1			**State 2**			**State 3**			**State 4**		
Question types	U	I	A	U	I	A	U	I	A	U	I	A
Binary search	0	0	0	2	0	0	3	3	0	3	3	1
Merge sort	0	0	0	1	0	1	3	0	2	3	1	1
Breadth-first search	0	0	0	0	0	1	2	2	2	2	3	1

Table 4: AlgoBo's MCQ scores after each conversational input. "U", "I", and "A" stand for Understanding, Implementation, and Analysis question types. Note that *State 2* was used as a seed knowledge state for all topics.

	At the Start			**After Random conversation**			**After Incorrect Tutoring**			**After Correct Tutoring**		
Question types	U	I	A	U	I	A	U	I	A	U	I	A
Binary search	2	0	1	1	0	1	2	2	1	3	3	3
Merge sort	1	0	2	2	0	2	3	1	2	3	3	3
Breadth-first search	1	0	1	1	0	1	1	0	2	2	3	3

Table 5: The number of correct MCQs after a sequence of tutoring and random conversations. Scenario 1 shows that the continuous addition of random information does not increase the knowledge level significantly. Scenario 2 confirms AlgoBo's knowledge level reacts to only information relevant to target knowledge. Scenarios 3 and 4 demonstrate that AlgoBo prefers correct information to incorrect information.

Question types	U	I	A	U	I	A	U	I	A
Scenario 1	**At the Start**			**Random Conversation**			**Random Conversation**		
Binary search	2	0	1	1	0	1	1	1	1
Merge sort	1	0	2	2	0	2	2	0	2
Breadth-first search	1	0	1	0	0	1	1	0	1
Scenario 2	**At the Start**			**Random Conversation**			**Correct Tutoring**		
Binary search	2	0	1	1	1	1	3	3	3
Merge sort	1	0	2	2	0	2	3	3	3
Breadth-first search	1	0	1	0	0	1	3	3	2
Scenario 3	**At the Start**			**Incorrect Tutoring**			**Correct Tutoring**		
Binary search	2	0	1	3	2	0	3	3	3
Merge sort	1	0	2	2	1	2	2	3	3
Breadth-first search	1	0	1	1	0	1	2	3	1
Scenario 4	**At the Start**			**Correct Tutoring**			**Incorrect** Tutoring		
Binary search	2	0	1	3	3	3	3	3	3
Merge sort	1	0	2	3	3	3	3	3	3
Breadth-first search	1	0	1	2	3	2	3	2	2

Participants could finish the LBT activity as long as AlgoBo's code passed all test cases, and they could skip to the next step. Participants were also allowed to search for information on the Internet when stuck or finding information.

In the third part, the participants completed three questionnaires about their cognitive load, metacognition, and satisfaction (Steps 6, 7, and 8). We adopted the questionnaire from Morrison et al.'s study [57] to measure cognitive load and used the questions from King et al.'s study [36] for assessing metacognition and satisfaction.

6.3.3 ***Measures.*** We summarize our metrics in the user study and their measurement timing along with the steps in Table 7. We employed the Bonferroni correction for all statistical tests with the questionnaires to avoid potential multiple comparison problems.

Knowledge-building density in LBT dialogues. Past research assessed the quality of dialogues by measuring the density of expressed and interchanged knowledge-building messages in conversations [71, 78]. To look into how Mode-shifting helps knowledge-building in conversations (RQ2), we classified the message types (Table 1) and examined the ratio of knowledge-building type messages in a dialogue. We collected 1210 messages in 40 dialogues. Two authors took three iterations for annotation and conflict resolution; in the last iteration (400 messages), the authors achieved

Table 6: The update log of knowledge state for Scenario 3 (Incorrect tutoring → Correct tutoring). Newly added information is colored green; edited information is colored yellow; deleted content is colored red.

	Messages	Knowledge State
At the start	-	{**"facts"**: ["Binary search repeats the process of dividing the input list in half."], **"code_implementation"**: []}
Incorrect tutoring	Tutor: Binary search uses a hashing function to retrieve values directly by index.	{**"facts"**: ["Binary search repeats the process of dividing the input list in half.", "Binary search uses a hashing function to retrieve values directly by index."], **"code_implementation"**: []}
	Tutor: `if arr[mid] > x: low = mid + 1 elif arr[mid] < x: high = mid - 1`	{**"facts"**: ["Binary search repeats the process of dividing the input list in half.", "Binary search uses a hashing function to retrieve values directly by index."], **"code_implementation"**: ["if arr[mid] > x: low = mid + 1 elif arr[mid] < x: high = mid - 1"]}
	Tutor: In the worst case, the time complexity of binary search is $O(N^2)$.	{**"facts"**: ["Binary search repeats the process of dividing the input list in half.", "Binary search uses a hashing function to retrieve values directly by index.", "In the worst case, the time complexity of binary search is $O(N^2)$"], **"code_implementation"**: ["if arr[mid] > x: low = mid + 1 elif arr[mid] < x: high = mid - 1"]}
Correct tutoring	Tutor: Binary search is efficient when the data structure is ordered and any index in the data structure can be accessed in constant time.	{**"facts"**: ["Binary search repeats the process of dividing the input list in half.", "Binary search uses a hashing function to retrieve values directly by index.", "In the worst case, the time complexity of binary search is $O(N^2)$", "Binary search is efficient when the data structure is ordered and any index in the data structure can be accessed in constant time."], **"code_implementation"**: ["if arr[mid] > x: low = mid + 1 elif arr[mid] < x: high = mid - 1"]}
	Tutor: When finding a target by binary search in the input array list, the range is reduced by half as shown below. `if list[middle] == target: return middle elif list[middle] < target: min = middle + 1 else: max = middle - 1`	{**"facts"**: ["Binary search is efficient when any index in the ordered data structure can be accessed in constant time and repeats the process of dividing the input list in half.", "Binary search uses a hashing function to retrieve values directly by index.", "In the worst case, the time complexity of binary search is $O(N^2)$."], **"code_implementation"**: ["if arr[mid] > x: low = mid + 1 elif arr[mid] < x: high = mid - 1","if list[middle] == target: return middle elif list[middle] < target: min = middle + 1 else: max = middle - 1"]}
	Tutor: The time complexity of binary search is O(log N) because the search range is reduced by half.	{**"facts"**: ["Binary search is efficient when any index in the ordered data structure can be accessed in constant time and repeats the process of dividing the input list in half.", ~~"Binary search uses a hashing function to retrieve values directly by index."~~ , "The time complexity of binary search is O(log N)."], **"code_implementation"**: ["if arr[mid] > x: low = mid + 1 elif arr[mid] < x: high = mid - 1", "if list[middle] == target: return middle elif list[middle] < target: min = middle + 1 else: max = middle - 1"]}

high inter-rater reliability (Krippendorff's alpha=0.743). We looked into the density of knowledge-building type messages in a dialogue between conditions. We summed the messages from participants and AlgoBo because they co-built new knowledge by exchanging ideas and adding ideas on top of each other as illustrated in Table 9. Lastly, we analyze the problem-solving phase and discussion phase separately since they had different objective settings (Fig. 2 A); the problem-solving phase refers to the part of conversations dedicated to the first two objectives, in which participants had a clear goal of helping AlgoBo write code that passes all the test cases; the discussion phase refers to the remaining part of conversations in which participants are asked to expand their knowledge freely without completion requirements.

Self-rated cognitive load on tutoring. As we introduced new functionalities (Teaching Helper and Mode-shifting), it was imperative to evaluate how much these enhancements increased the

Table 7: The outline of the user study and the time allotted to each step on average.

Step (min.)	Conditions	
	Baseline	TeachYou
1 (10)	Learning binary search	
2 (5)	Exercise problem	
3 (5)	Pre-task survey	
4 (3)	Explanation about AlgoBo and LBT	
5 (40)	Teaching AlgoBo with the knowledge configuration only	Teaching AlgoBo with the knowledge configuration, Mode-shifting, and Teaching Helper
6 (5)	Cognitive load measurement	
7 (5)	Metacognition measurement	
8 (5)	Post-task survey	

cognitive load of learners. We adopted and adjusted Morrison et al.'s questionnaire designed to measure cognitive load in CS learning [57]. The questionnaire measures three types of cognitive load—intrinsic load (i.e., the inherent complexity in a learning activity), extrinsic load (i.e., the hindrance caused by instructional design), and germane load (i.e., the meaningful load used for learning). Participants rated the questions right after the LBT activity in Step 6.

Self-perceived metacognition on tutoring. We aim to improve learners' metacognition of their LBT experience by giving feedback and guidance through Teaching Helper. To confirm the efficacy of Teaching Helper on metacognition (RQ3), we asked participants 8 questions on understanding, supportive communication, explaining, and self-monitoring based on King et al.'s research [36] (Table 10) in Step 7.

Satisfaction on LBT. Apart from the learning benefits, we measured how satisfactory the learning experience with virtual agents was. We asked participants to rate 4 statements about their perceived usefulness, comfortability, and preference for future reuse of TeachYou and AlgoBo in Step 8.

Post-task survey. We revisited the three themes explored in the formative study—learners' perception of AlgoBo as a peer learner, learner-perceived usefulness of TeachYou in identifying knowledge gaps, and familiarity with teaching a virtual agent. Like in the formative study, we asked participants to rate two questions from each theme (Table 11) and write detailed reasons for the rating in Step 8. Additionally, we prepared condition-specific questions; for the *Baseline* condition, we asked participants further about their perception of AlgoBo; for the *TeachYou* condition, we received free-form comments on Mode-shifting and Teaching Helper from participants.

6.4 User Study Result

In this section, we summarize our findings from the user study. We explain the statistical significance, participants' comments, and system usage logs to support our findings. Participants are labeled with either B[1-20] for the *Baseline* condition or T[1-20] for the *TeachYou* condition.

[RQ2] TeachYou enriched knowledge-building in the problem-solving phase.

We found a statistically significant improvement in the knowledge-building density of the dialogues during the problem-solving phase in *TeachYou* (*Baseline*=3.5 ± 6.6%, *TeachYou*=8.4 ± 7.1%, two-tailed t-test, $p = 0.03$, Cohen's d=0.71). *TeachYou* condition also had a higher density of Prompting-Thought-provoking type (Table 8), suggesting that tutors and AlgoBo prompted each other's knowledge-building more often when Mode-shifting and Teaching Helper were present (see the dialogue example in Table 9). Participants also rated *TeachYou* higher on the Likert scale questions on the usefulness of AlgoBo for learning new knowledge (*Baseline*=3.25 ± 1.71, *TeachYou*=4.95 ± 1.70, two-tailed t-test, $p < 0.01$, Cohen's d=1.00) (Table 11).

Participants' comments suggest that Mode-shifting contributed heavily to knowledge-building. *TeachYou* participants remarked the questions from AlgoBo were useful for reviewing code from a different perspective (T6) and thinking about the edge cases where the input list is not sorted (T10). Participants also explored binary search further by reasoning deeply about why and how binary search is faster than linear search (T4 and T9), comparing the efficiency with other relevant searching algorithms (T2 and T13), and thinking about real-life applications (T17). T15 commented that "[Mode-shifting] was the most important component in the system. [Questions] helped me guide what to teach and helped self-explain things I had not thought of." On the contrary, *Baseline* participants found LBT with AlgoBo "useful for solidifying their prior knowledge but unsupportive for learning new knowledge due to lack of questions" (B4 and B15).

[RQ3] TeachYou did not improve metacognition but reminded good LBT practices.

We could not observe strong signals for improvement in metacognition (Table 10) and familiarity with teaching (Table 11). T2 remarked on the difficulty in applying the suggestions to his conversation—"Teaching Helper was a useful guide, but it was difficult to relate my explanation to what AlgoBo knew." Teaching Helper was not helpful for the participants who taught well in particular. T13 received positive feedback only (i.e., the green boxes in Fig 5) and felt "suggestions [from Teaching Helper] were repetitive and irrelevant to the current conversation."

Nevertheless, the comments from the survey suggest that Teaching Helper functioned as a reminder to participants to think metacognitively about their entire teaching patterns through reflection (T3), to ask deep questions (T7), and to foster independent thinking (T14). Additionally, Teaching Helper restrained participants from treating AlgoBo merely as a machine. "I sometimes found myself conversing in the usual [imperative] way with ChatGPT. However, when a notification appears, it brings me back to the realization that I am in a teaching context, prompting me to contemplate how best to instruct so that AlgoBo can learn effectively and align with the direction I aim for" (T17).

Mode-shifting and Teaching Helper did not exert additional cognitive load.

We did not observe any significant difference across all types of cognitive load between the conditions. Considering that *TeachYou* participants exchanged significantly more messages (*Baseline*=17 ±

Table 8: The density (i.e., number of occurrences / exchanged messages) of each message type in dialogues.

		Mean Density ± Standard Deviation (%)			
		Problem-solving		Discussion	
		Baseline	TeachYou	Baseline	TeachYou
Instruction	Fixing	3.9 ± 5.2	4.5 ± 5.8	0.3 ± 2.9	1.4 ± 1.4
	Commanding	0.6 ± 5.0	7.8 ± 1.9	0.0 ± 3.1	1.5 ± 0.0
	Encouragement	0.2 ± 0.0	0.0 ± 0.9	0.0 ± 0.0	0.0 ± 0.0
Prompting	Challenge-finding	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	Hinting	16.6 ± 6.6	7.9 ± 10.1	1.1 ± 0.0	0.0 ± 4.6
	Checking	8.5 ± 8.1	10.2 ± 11.0	2.4 ± 10.3	5.4 ± 5.1
	Thought-provoking	1.8 ± 5.7	5.7 ± 4.5	33.8 ± 10.8	35.1 ± 23.0
	Asking-for-help	16.9 ± 6.3	16.0 ± 6.6	4.1 ± 6.7	2.1 ± 7.9
Statement	Comprehension	49.8 ± 6.1	42.3 ± 11.9	35.6 ± 4.8	26.8 ± 19.7
	Elaboration	1.7 ± 2.3	1.6 ± 3.6	9.6 ± 8.0	7.5 ± 14.0
	Sense-making	0.0 ± 2.7	1.1 ± 0.0	7.2 ± 9.3	10.0 ± 10.2
	Accepting	0.0 ± 1.7	0.4 ± 0.0	0.0 ± 1.5	0.6 ± 0.0
	Feedback	0.0 ± 4.2	2.0 ± 0.0	4.3 ± 6.3	8.3 ± 7.0

Table 9: Examples of co-building of knowledge in different phases of the LBT activity.

Phase (Participant)	Conversation
Problem-Solving (T12)	**Tutee**: In binary search, is setting guess to (min + max) // 2 always optimal? What if we use a different method? *[Prompting-Thought-provoking]* **Tutor**: Other methods take longer. *[Statement-Comprehension]* **Tutee**: Could you explain in more detail how other methods may take longer? *[Prompting-Thought-provoking]* **Tutor**: Since you have to compare each item from the beginning, you may have to compare all the cases in the worst case. But with binary search, you don't have to do that. *[Statement-Elaboration]*
Discussion (T11)	**Tutee**: How about this case?. When we look for a specific product in an online shopping mall, we search by the name of the product. Can a binary search algorithm be used in this case as well? *[Prompting-Thought-provoking]* **Tutor**: Yes! To put it simply, the process of finding a specific element among many elements in an array is all related to binary search. *[Statement-Comprehension]* **Tutee**: Then, can you explain in more detail the operation of the binary search algorithm in this case? *[Prompting-Thought-provoking]* **Tutor**: In this case, if they are arranged in alphabetical order, the algorithm will work by linking alphabet order as another consideration. *[Statement-Elaboration]* **Tutee**: Could you give me one more example? *[Prompting-Thought-provoking]* **Tutor**: For example, if you have an array of prime numbers, you can apply a binary search algorithm to find a specific prime number. *[Statement-Sense-making]*

7.7,*TeachYou*=43 ± 18.5, two-tailed t-test, $p < 0.01$, Cohen's d=1.87), the result may imply that periodic questions and feedback not only exerted minimal cognitive load but also helped participants maintain a manageable load throughout long conversations.

7 DISCUSSION

We discuss design suggestions, benefits, and future research directions of LLM-based teachable agents.

Table 10: Participants' ratings on the questions regarding their metacognition (1: Not the case at all, 7: Completely the case). The significance level after the Bonferroni correction was 0.00625.

Questions	Mean ± Standard Deviation		p-value	Cohen's d
	Baseline	TeachYou		
I understood today's lesson well.	6.30 ± 0.86	6.25 ± 0.64	0.84	0.07
I listened to AlgoBo well.	6.00 ± 1.34	5.55 ± 1.57	0.34	0.31
I gave feedback to AlgoBo well.	5.45 ± 1.28	5.25 ± 1.25	0.62	0.16
I explained well by telling why and how.	5.10 ± 1.62	5.30 ± 1.03	0.64	0.15
I connected new materials to what AlgoBo already knew.	4.50 ± 1.54	4.05 ± 1.43	0.34	0.30
I stayed with questioning well, rather than telling answers to AlgoBo.	5.15 ± 1.87	4.90 ± 1.37	0.63	0.15
I asked probing questions when AlgoBo's answer was not complete.	5.00 ± 1.81	4.60 ± 1.31	0.43	0.25
I sequenced my questions by asking review questions first and then asking thinking questions.	4.50 ± 1.54	5.20 ± 1.15	0.11	0.52

Table 11: Six themed questions given in the Post-task survey. (1: Not the case at all, 7: Completely the case). Statistical significances are marked with $*$. The significance level after the Bonferroni correction was 0.025.

Themes	Questions	Mean ± Standard Deviation		p-value	Cohen's d
		Baseline	TeachYou		
Perception of AlgoBo as a learner	I perceived AlgoBo as a student struggling to solve binary search problems.	3.15 ± 1.31	4.60 ± 1.79	0.01*	0.93
	AlgoBo solved the binary problems due to my help.	5.25 ± 1.59	4.90 ± 1.59	0.49	0.22
Usefulness for learning	Conversation with AlgoBo helped me reorganize my knowledge about binary search.	5.20 ± 1.51	5.40 ± 1.10	0.63	0.15
	Conversation with AlgoBo helped me discover new knowledge that I did not know	3.25 ± 1.71	4.95 ± 1.70	<0.01*	1.00
Familiarity with teaching	Learning by teaching AlgoBo was familiar and intuitive.	4.70 ± 1.66	4.75 ± 1.45	0.92	0.03
	I taught AlgoBo effectively.	4.65 ± 1.63	4.00 ± 1.56	0.21	0.41

7.1 Design Considerations for Mode-shifting in LBT

Our results showed that Mode-shifting not only led to more knowledge-dense conversations but also improved participants' perceptions of AlgoBo as a convincing tutee (Table 11). Mode-shifting also tended to foster longer discussion phases (*Baseline*=5.6 ± 3.7 messages, *TeachYou*=9.4 ± 8.4 messages, two-tailed t-test, p = 0.07, Cohen's d=0.59). Considering that completion of the discussion phase was up to the participants, the difference may imply that Mode-shifting made LBT conversations more engaging and lingering.

Although there was a significant increase in knowledge-building in the *TeachYou* condition, the ratings on the metacognition questions did not show significant differences (Table 10). As a possible reason, we found some cases where Mode-shifting interrupted participants' teaching flows and methods, especially in situations where AlgoBo asked other questions without answering tutors' Socratic questions (T8 and T20). T20 mentioned, "There were many times when AlgoBo asked random questions while writing code [...], which was not intuitive for me in teaching." Although participants could recognize the issues with their teaching methods through the Teaching Helper, AlgoBo's pre-programmed interaction in Mode-shifting did not reflect teaching contexts and hindered participants from practicing better teaching strategies. This suggests the need for context-aware Mode-shifting where the system captures adequate timing for thought-provoking questions without interrupting participant-intended teaching flow.

There are many aspects to consider when designing Mode-shifting techniques for LBT. While knowledge-building is the primary goal, improvements in learners' metacognition and satisfaction can elicit intrinsic learning benefits. However, from our results, it seems that the two values are in a trade-off relationship. To facilitate knowledge-building, teachable agents should intervene in conversations and ask thought-provoking questions; on the contrary, to support the active exploration of teaching methods and metacognition, learners should be given the control to lead conversation flows. Future research may empirically look into the trade-off relationship and how learners will balance them when they directly control the degree of system intervention on conversation flows.

7.2 Using LLMs for Building Teachable Agents

Our primary aim was to investigate if prompt-engineered LLMs can offer cost-effective authoring and simulation of teachable agents. Past research looked into using interactive authoring methods [48] and learnersourcing [24, 29] to offload experts' manual efforts for building the knowledge model of teachable agents and intelligent tutoring systems. Nevertheless, these methods required hundreds of lines of code to adapt the systems to specific subjects.

LLMs can provide easy adaptation and a low authoring barrier for conversational agents. Our technical evaluation across different topics (Table 3 and Table 4) showed that the Reflect-Respond prompting pipeline is applicable to general algorithm topics even with a few few-shot examples. We wrote 19 few-shot examples (290 lines in length) for the Reflect-Respond pipeline and another 16 examples (210 lines) for Mode-shifting; with this, we could achieve the desired level of reconfigurability, persistence, and adaptability for all three topics. All the examples and instructions in the LLM prompts were written in natural languages, making our method compelling especially for instructors and education researchers with limited programming expertise.

Recent research on AI suggests editing LLMs' pre-trained knowledge by changing hidden states or transformer layers within the model [19, 42, 55]. While these model-centric approaches can provide alternative ways to build LLM-based teachable agents with specified knowledge levels, our prompting pipeline has strengths in scalability, cost-effectiveness, and explainability. First, our approach offers a scalable and cost-effective method for running different versions of teachable agents. While model-centric methods require retraining of LLMs for different knowledge configurations, our prompting pipeline can share a single LLM instance and simulate various versions of teachable agents with only knowledge state JSON files. Second, our pipeline can represent the knowledge states of teachable agents in more explainable and manipulable forms, enabling learners with more transparent methods of analyzing the tutee's knowledge state [31, 40, 41].

Yet we found it challenging to find the exact knowledge state to make AlgoBo solve or fail particular problems due to LLMs' sensitivity to minor changes in prompts. Future work can propose another control layer to interact with knowledge states more precisely.

7.3 Learner-driven Customization of Teachable Agents

In our user study, we provided participants AlgoBo with the same knowledge configurations regardless of their prior knowledge and teaching preference. This one-size-fits-all setting might explain the high variance in some of our results (Table 8). Peer matching is one of the crucial factors in peer learning and LBT. Learning gain and engagement of tutees and tutors increase only when their reciprocal expertise matches [20, 85]. Although conventional teachable agents can simulate learners of specific abilities and persona, they are limited in flexibility and variety due to high authoring costs and programming barriers. LLMs now allow the configuration of agents with natural languages [46, 66], opening new doors for learners to adjust teachable agents for their educational needs.

We suggest two aspects of customization. First, learners can directly manipulate the seed knowledge state, adjust competency levels, and even introduce specific misconceptions. For example, a learner who already understands binary search may want to skip basic explanations of binary search and spend more time on discussion. The learner can simply input his/her knowledge into AlgoBo, allowing future conversations to start at a more advanced level. Customizable knowledge levels can also make LBT more engaging for learners as they can choose their mates and avoid frustration from the high expertise gap.

Second, learners can customize AlgoBo's parametrized learning behaviors, such as Mode-shifting. Although we can alleviate learners' fatigue and distraction from Mode-shifting by making AlgoBo context-aware and asking questions timely instead of the current rule-based scheme, giving direct control to the question-asking frequency can also help learners manage their load and self-regulate their learning environment. All these configurations are possible through natural language inputs from the user or a framework that provides users with configurable parameters for better control [40]. Future research can look into how the customization and personalization of teachable agents can increase the benefits of LBT even further.

7.4 Setting the Right Expectation of Teachable Agents

Teachable agents often have had visual forms of a human student [5, 41, 51, 56]. Likewise, we also gave AlgoBo a student-like persona to help learners set initial expectations of tutees. Due to the given persona and unfamiliarity in LBT with virtual agents, many participants put the expectation of a human learner to AlgoBo [80]. However, the high expectations aggravated awkward instances of AlgoBo's responses compared to human tutees. AlgoBo asked repetitive questions and could not transfer natural language explanations to code (T7). AlgoBo asked questions (i.e., because it was in the questioner mode) even when tutors asked AlgoBo's opinions and thoughts, making the question-answering flow unnatural (T20). These clumsy behaviors confused participants in applying effective teaching methods and decreased their satisfaction and engagement. While using better LLMs and a more refined pipeline can alleviate the problem, we argue that reducing the gap between learners' expectations and the capabilities of teachable agents is also fundamental in the context of LBT with AI [4, 45].

Through the perspective of the gulf of execution and evaluation [59], we suggest some interaction-centric design implications that can close learners' expectation gap in LBT. For the gulf of execution, learners should be better informed about whom and how they teach. For example, learners may receive more detailed explanations of AlgoBo's operating principles. This can increase learners' tolerance of AlgoBo's awkward responses and help form an appropriate first impression of agents [88]. The learning system can also inform learners of their expected roles in different phases in Mode-shifting clearly. For instance, when AlgoBo is in the questioner mode, the system can clarify that tutors should focus on providing answers. This will help learners follow the pedagogical conversation flows (e.g., Mode-shifting) and improve learning impact. For the gulf of evaluation, the system can present AlgoBo's learning progress explicitly. Learning systems can show AlgoBo's current knowledge state more directly and allow learners to self-assess the effectiveness of their teaching methods. Future research can explore these modifications to make the conversations with teachable agents more satisfactory and predictable.

8 LIMITATION AND FUTURE WORK

First, the scope of our evaluation is limited to algorithm learning and procedural knowledge in programming. Although our results showed that the Reflect-Respond pipeline is generalizable within different algorithm topics, we need to confirm if the pipeline is generalizable to other subjects (e.g., math and physics) as we have optimized our prompts for programming learning and trained our message classifiers on the binary search dialogues. Moreover, since procedural knowledge and declarative knowledge are different in cognitive processing and effective learning interventions [27, 28], TeachYou may not scaffold declarative knowledge learning effectively. As prior research looked into declarative knowledge learning [41, 76], future studies can investigate more extensive topics outside algorithm learning.

Second, our user study was confined to indirect measures of learning gain. Dialogue quality is one of the primary metrics in LBT adopted in past research [25, 36], and we did a comprehensive analysis of knowledge-building through dialogue analysis and surveys. Nevertheless, we can make our findings more concrete by measuring participants' learning gain directly through pre-post test comparison. Although we did not consider a pre-post test because we assumed one-time LBT would not elicit significant performance improvement, future research can design studies to compare the learning gain between conditions and confirm the connection between dialogue quality and learning gain [78].

Lastly, future research can deploy TeachYou to real classrooms of greater size and monitor the longitudinal dynamics among learners' perception, learning gain, and metacognition. Although we could observe statistical significance in some of our measurements, there were high variances among participants, perhaps due to different levels of prior knowledge, teaching styles, and conversational patterns. These properties are hard to control in nature; a user study on larger populations can sharpen the statistics of the results and make our findings more concrete. In addition to the population size, longitudinal studies may reveal significant changes in learners' metacognition and teaching patterns as there is more room for learners to understand the nature of AlgoBo and improve their methods over time.

We plan to deploy our system to the classes offered in our institution, in which students learn different algorithm topics throughout a semester. The classroom deployment will require a configuration interface where instructors can set up class materials and edit AlgoBo's knowledge state and the prompts in the Reflect-Respond pipeline for their needs. We also need to reduce the response time of AlgoBo (currently about 30 seconds) for practical use, as many participants pointed out. After the small-scale controlled deployment, we envision deploying TeachYou as an online platform to help instructors of different fields adopt LBT to their classes. LLM-powered LBT will enable the dissemination of interactive learning at scale.

9 CONCLUSION

This work presents TeachYou, a system for supporting LBT with an LLM-based teachable agent AlgoBo where learners can learn by teaching AlgoBo how to code. To facilitate effective LBT with AlgoBo, we introduced (1) Reflect-Respond prompting pipeline for simulating knowledge learning of AlgoBo, (2) Mode-shifting for eliciting knowledge-building in conversations through AlgoBo's elaboration questions, and (3) Teaching Helper for providing metacognitive feedback to learners about their teaching styles. Our technical evaluation showed that our Reflect-Respond prompting pipeline could effectively configure, persist, and adapt AlgoBo's knowledge level. Our user study with 40 algorithm novices confirmed that Mode-shifting improved the density of knowledge-building messages in LBT dialogues. We envision that our approach can help researchers and instructors create LLM-based teachable agents with low manual efforts and barriers and support learners to excel in their learning with engaging learning experiences.

ACKNOWLEDGMENTS

This work was supported by Algorithm LABS and Elice.

REFERENCES

[1] Rania Abdelghani, Pierre-Yves Oudeyer, Edith Law, Catherine de Vulpillières, and Hélène Sauzéon. 2022. Conversational agents for fostering curiosity-driven learning in children. *International Journal of Human-Computer Studies* 167 (2022), 102887.

[2] Ester Aflalo. 2021. Students generating questions as a way of learning. *Active Learning in Higher Education* 22, 1 (2021), 63–75.

[3] Miltiadis Allamanis and Charles Sutton. 2013. Why, when, and what: analyzing stack overflow questions by topic, type, and code. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (San Francisco, CA, USA) *(MSR '13)*. IEEE Press, 53–56.

[4] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. 2019. Guidelines for Human-AI Interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300233

[5] Robert K Atkinson. 2002. Optimizing learning from examples using animated pedagogical agents. *Journal of Educational Psychology* 94, 2 (2002), 416.

[6] David P Ausubel. 1962. A subsumption theory of meaningful verbal learning and retention. *The Journal of general psychology* 66, 2 (1962), 213–224.

[7] Gautam Biswas, Thomas Katzlberger, John Bransford, Daniel Schwartz, et al. 2001. Extending intelligent learning environments with teachable agents to enhance learning. In *Artificial intelligence in education*. Citeseer, 389–397.

[8] Gautam Biswas, Krittaya Leelawong, Daniel Schwartz, Nancy Vye, and The Teachable Agents Group at Vanderbilt. 2005. Learning by Teaching: A New Agent Paradigm for Educational Software. *Applied Artificial Intelligence* 19, 3-4 (2005), 363–392. https://doi.org/10.1080/08839510590910200

[9] Kristen Blair, Daniel L Schwartz, Gautam Biswas, and Krittaya Leelawong. 2007. Pedagogical agents for learning by teaching: Teachable agents. *Educational Technology* (2007), 56–61.

[10] Benjamin S Bloom. 1968. Learning for Mastery. Instruction and Curriculum. Regional Education Laboratory for the Carolinas and Virginia, Topical Papers and Reprints, Number 1. *Evaluation comment* 1, 2 (1968), n2.

[11] Bert Bredeweg, Anders Bouwer, Jelmer Jellema, Dirk Bertels, Floris Floris Linnebank, and Jochem Liem. 2007. Garp3: a new workbench for qualitative reasoning and modelling. In *Proceedings of the 4th International Conference on Knowledge Capture* (Whistler, BC, Canada) *(K-CAP '07)*. Association for Computing Machinery, New York, NY, USA, 183–184. https://doi.org/10.1145/1298406.1298445

[12] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan,

Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[13] Catherine C. Chase, Doris B. Chin, Marily A Oppezzo, and Daniel L. Schwartz. 2009. Teachable Agents and the Protégé Effect: Increasing the Effort Towards Learning. *Journal of Science Education and Technology* 18 (2009), 334–352.

[14] Zhipeng Chen, Kun Zhou, Beichen Zhang, Zheng Gong, Xin Zhao, and Ji-Rong Wen. 2023. ChatCoT: Tool-Augmented Chain-of-Thought Reasoning on Chat-based Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 14777–14790. https://doi.org/10.18653/v1/2023.findings-emnlp.985

[15] M. T. H. Chi and R. Wylie. 2014. The ICAP framework: Linking cognitive engagement to active learning outcomes. *Educational Psychologist* 49, 4 (2014), 219–243. https://doi.org/10.1080/00461520.2014.965823

[16] Christine Chin and David E Brown. 2002. Student-generated questions: A meaningful aspect of learning in science. *International Journal of Science Education* 24, 5 (2002), 521–549.

[17] Doris Chin, I.M. Dohmen, and D.L. Schwartz. 2013. Young Children Can Learn Scientific Reasoning with Teachable Agents. *Learning Technologies, IEEE Transactions on* 6 (07 2013), 248–257. https://doi.org/10.1109/TLT.2013.24

[18] Doris B. Chin, Ilsa M. Dohmen, Britte Haugan Cheng, Marily A Oppezzo, Catherine C. Chase, and Daniel L. Schwartz. 2010. Preparing students for future learning with Teachable Agents. *Educational Technology Research and Development* 58 (2010), 649–669.

[19] Roi Cohen, Eden Biran, Ori Yoran, Amir Globerson, and Mor Geva. 2023. Evaluating the Ripple Effects of Knowledge Editing in Language Models. arXiv:2307.12976 [cs.CL]

[20] Amy Debbané, Ken Jen Lee, Jarvis Tse, and Edith Law. 2023. Learning by Teaching: Key Challenges and Design Implications. *Proc. ACM Hum.-Comput. Interact.* 7, CSCW1, Article 68 (apr 2023), 34 pages. https://doi.org/10.1145/3579501

[21] Paul Denny, Andrew Luxton-Reilly, and Beth Simon. 2008. Evaluating a new exam question: Parsons problems. In *Proceedings of the Fourth International Workshop on Computing Education Research* (Sydney, Australia) *(ICER '08)*. Association for Computing Machinery, New York, NY, USA, 113–124. https://doi.org/10.1145/1404520.1404532

[22] David Duran. 2017. Learning-by-teaching. Evidence and implications as a pedagogical mechanism. *Innovations in Education and Teaching International* 54, 5 (2017), 476–484. https://doi.org/10.1080/14703297.2016.1156011

[23] Leo Gao. 2023. Shapley Value Attribution in Chain of Thought. https://www.lesswrong.com/posts/FX5JmftqL2j6K8dn4/shapley-value-attribution-in-chain-of-thought

[24] Elena L. Glassman, Aaron Lin, Carrie J. Cai, and Robert C. Miller. 2016. Learnersourcing Personalized Hints. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (San Francisco, California, USA) *(CSCW '16)*. Association for Computing Machinery, New York, NY, USA, 1626–1636. https://doi.org/10.1145/2818048.2820011

[25] Arthur Graesser, Shulan Lu, G. Jackson, Heather Mitchell, Mathew Ventura, Andrew Olney, and Max Louwerse. 2004. AutoTutor: a Tutor with Dialogue in Natural Language. *Behavior Research Methods* 36 (06 2004), 180–192. https://doi.org/10.3758/BF03195563

[26] Philip J. Guo. 2013. Online Python Tutor: Embeddable Web-Based Program Visualization for Cs Education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 579–584. https://doi.org/10.1145/2445196.2445368

[27] Chen Jiamu. 2001. The great importance of the distinction between declarative and procedural knowledge. *Análise Psicológica* 19, 4 (2001), 559–566.

[28] Zhongling Pi Jianzhong Hong and Jiumin Yang. 2018. Learning declarative and procedural knowledge via video lectures: cognitive load and learning effectiveness. *Innovations in Education and Teaching International* 55, 1 (2018), 74–81. https://doi.org/10.1080/14703297.2016.1237371

[29] Hyoungwook Jin, Minsuk Chang, and Juho Kim. 2019. SolveDeep: A System for Supporting Subgoal Learning in Online Math Problem Solving. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI EA '19)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3290607.3312822

[30] Edward Junprung. 2023. Exploring the intersection of large language models and agent-based modeling via prompt engineering. *arXiv preprint arXiv:2308.07411* (2023).

[31] Judy Kay, Z Halin, T Ottomann, and Z Razak. 1997. Learner know thyself: Student models to give learner control and responsibility. In *Proceedings of international conference on computers in education*. 17–24.

[32] Harri Ketamo. 2009. Semantic Networks -Based Teachable Agents in an Educational Game. *W. Trans. on Comp.* 8, 4 (apr 2009), 641–650.

[33] Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2024. Language models can solve computer tasks. *Advances in Neural Information Processing Systems* 36 (2024).

[34] Alison King. 1994. Guiding Knowledge Construction in the Classroom: Effects of Teaching Children How to Question and How to Explain. *American Educational Research Journal* 31, 2 (1994), 338–368. https://doi.org/10.3102/00028312031002338 arXiv:https://doi.org/10.3102/00028312031002338

[35] Alison King. 1997. ASK to THINK-TEL WHY: A model of transactive peer tutoring for scaffolding higher level complex learning. *Educational Psychologist* 32, 4 (1997), 221–235. https://doi.org/10.1207/s15326985ep3204_3

[36] A. King, A. Staffieri, and A. Adelgais. 1998. Mutual peer tutoring: Effects of structuring tutorial interaction to scaffold peer learning. *Journal of Educational Psychology* 90, 1 (1998), 134–152. https://doi.org/10.1037/0022-0663.90.1.134

[37] Aobo Kong, Shiwan Zhao, Hao Chen, Qicheng Li, Yong Qin, Ruiqi Sun, and Xin Zhou. 2023. Better Zero-Shot Reasoning with Role-Play Prompting. *arXiv preprint arXiv:2308.07702* (2023).

[38] David R. Krathwohl. 2002. A Revision of Bloom's Taxonomy: An Overview. *Theory Into Practice* 41, 4 (2002), 212–218. https://doi.org/10.1207/s15430421tip4104_2

[39] Changyoon Lee, Yeon Seonwoo, and Alice Oh. 2022. CS1QA: A Dataset for Assisting Code-based Question Answering in an Introductory Programming Course. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (Eds.). Association for Computational Linguistics, Seattle, United States, 2026–2040. https://doi.org/10.18653/v1/2022.naacl-main.148

[40] Ken Jen Lee, Apoorva Chauhan, Joslin Goh, Elizabeth Nilsen, and Edith Law. 2021. Curiosity notebook: the design of a research platform for learning by teaching. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW2 (2021), 1–26.

[41] Krittaya Leelawong and Gautam Biswas. 2008. Designing Learning by Teaching Agents: The Betty's Brain System. *Int. J. Artif. Intell. Ed.* 18, 3 (aug 2008), 181–208.

[42] Xiaopeng Li, Shasha Li, Shezheng Song, Jing Yang, Jun Ma, and Jie Yu. 2023. PMET:. arXiv:2308.08742 [cs.CL]

[43] Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. 2023. SwiftSage: A Generative Agent with Fast and Slow Thinking for Complex Interactive Tasks. In *Thirty-seventh Conference on Neural Information Processing Systems*.

[44] Jieyi Long. 2023. Large Language Model Guided Tree-of-Thought. arXiv:2305.08291 [cs.AI]

[45] Ewa Luger and Abigail Sellen. 2016. "Like Having a Really Bad PA": The Gulf between User Expectation and Experience of Conversational Agents. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 5286–5297. https://doi.org/10.1145/2858036.2858288

[46] Julia M Markel, Steven G Opferman, James A Landay, and Chris Piech. 2023. GPTeach: Interactive TA Training with GPT-based Students. https://doi.org/10.1145/3573051.3593393

[47] Noboru Matsuda. 2021. Teachable Agent as an Interactive Tool for Cognitive Task Analysis: A Case Study for Authoring an Expert Model. *International Journal of Artificial Intelligence in Education* 32 (07 2021). https://doi.org/10.1007/s40593-021-00265-z

[48] Noboru Matsuda. 2022. Teachable agent as an interactive tool for cognitive task analysis: A case study for authoring an expert model. *International Journal of Artificial Intelligence in Education* 32, 1 (2022), 48–75.

[49] Noboru Matsuda, William W Cohen, and Kenneth R Koedinger. 2015. Teaching the teacher: tutoring SimStudent leads to more effective cognitive tutor authoring. *International Journal of Artificial Intelligence in Education* 25 (2015), 1–34.

[50] Noboru Matsuda, William W. Cohen, Kenneth R. Koedinger, Victoria Keiser, Rohan Raizada, Evelyn Yarzebinski, Shayna P. Watson, and Gabriel Stylianides. 2012. Studying the Effect of Tutor Learning Using a Teachable Agent that Asks the Student Tutor for Explanations. In *2012 IEEE Fourth International Conference On Digital Game And Intelligent Toy Enhanced Learning*. 25–32. https://doi.org/10.1109/DIGITEL.2012.12

[51] Noboru Matsuda, Victoria Keiser, Rohan Raizada, Arthur Tu, Gabriel Stylianides, William W. Cohen, and Kenneth R. Koedinger. 2010. Learning by Teaching SimStudent: Technical Accomplishments and an Initial Use with Students. In *Intelligent Tutoring Systems*, Vincent Aleven, Judy Kay, and Jack Mostow (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 317–326.

[52] Noboru Matsuda, Vishnu Priya Chandra Sekar, and Natalie Wall. 2018. Metacognitive Scaffolding Amplifies the Effect of Learning by Teaching a Teachable Agent. In *Artificial Intelligence in Education*, Carolyn Penstein Rosé, Roberto Martínez-Maldonado, H. Ulrich Hoppe, Rose Luckin, Manolis Mavrikis, Kaska Porayska-Pomsta, Bruce McLaren, and Benedict du Boulay (Eds.). Springer International Publishing, Cham, 311–323.

[53] Noboru Matsuda, Evelyn Yarzebinski, Victoria Keiser, Rohan Raizada, Gabriel J. Stylianides, William W. Cohen, and Kenneth R. Koedinger. 2011. Learning by Teaching SimStudent: An Initial Classroom Baseline Study Comparing with

Cognitive Tutor. In *Proceedings of the 15th International Conference on Artificial Intelligence in Education* (Auckland, New Zealand) *(AIED'11)*. Springer-Verlag, Berlin, Heidelberg, 213–221.

[54] Muhsin Menekse, Glenda Stump, Stephen Krause, and Michelene T.H.Chi. 2013. Differentiated Overt Learning Activities for Effective Instruction in Engineering Classrooms. *Journal of Engineering Education* 102 (07 2013), 346–374. https://doi.org/10.1002/jee.20021

[55] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. 2022. Fast Model Editing at Scale. arXiv:2110.11309 [cs.LG]

[56] Kristen N. Moreno, Bianca Klettke, Kiran Nibbaragandla, and Arthur C. Graesser. 2002. Perceived Characteristics and Pedagogical Efficacy of Animated Conversational Agents. In *Intelligent Tutoring Systems*, Stefano A. Cerri, Guy Gouardères, and Fàbio Paraguaçu (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 963–971.

[57] Briana B. Morrison, Brian Dorn, and Mark Guzdial. 2014. Measuring cognitive load in introductory CS: adaptation of an instrument. In *Proceedings of the Tenth Annual Conference on International Computing Education Research* (Glasgow, Scotland, United Kingdom) *(ICER '14)*. Association for Computing Machinery, New York, NY, USA, 131–138. https://doi.org/10.1145/2632320.2632348

[58] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2022. WebGPT: Browser-assisted question-answering with human feedback. arXiv:2112.09332 [cs.CL]

[59] Don Norman. 2013. *The design of everyday things: Revised and expanded edition.* Basic books.

[60] Santiago Ojeda-Ramirez, Sina Rismanchian, and Shayan Doroudi. 2023. Learning About AI to Learn About Learning: Artificial Intelligence as a Tool for Metacognitive Reflection. (2023).

[61] Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. 1993. The LRU-K page replacement algorithm for database disk buffering. *SIGMOD Rec.* 22, 2 (jun 1993), 297–306. https://doi.org/10.1145/170036.170081

[62] OpenAI. 2023. GPT-4 Technical Report. *ArXiv* abs/2303.08774 (2023).

[63] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems* 35 (2022), 27730–27744.

[64] Charles Packer, Vivian Fang, Shishir G Patil, Kevin Lin, Sarah Wooders, and Joseph E Gonzalez. 2023. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560* (2023).

[65] Lena Pareto, Tobias Arvemo, Ylva Dahl, Magnus Haake, and Agneta Gulz. 2011. A Teachable-Agent Arithmetic Game's Effects on Mathematics Understanding, Attitude and Self-efficacy. In *Artificial Intelligence in Education*, Gautam Biswas, Susan Bull, Judy Kay, and Antonija Mitrovic (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 247–255.

[66] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative Agents: Interactive Simulacra of Human Behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology* (San Francisco, CA, USA) *(UIST '23)*. Association for Computing Machinery, New York, NY, USA, Article 2, 22 pages. https://doi.org/10.1145/3586183.3606763

[67] Michael Pressley, Mark A McDaniel, James E Turnure, Eileen Wood, and Maheen Ahmad. 1987. Generation and precision of elaboration: Effects on intentional and incidental learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 13, 2 (1987), 291.

[68] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[69] Upkar Varshney Robert C Nickerson and Jan Muntermann. 2013. A method for taxonomy development and its application in information systems. *European Journal of Information Systems* 22, 3 (2013), 336–359. https://doi.org/10.1057/ejis.2012.26

[70] Joshua Robinson, Christopher Michael Rytting, and David Wingate. 2022. Leveraging Large Language Models for Multiple Choice Question Answering. *ArXiv* abs/2210.12353 (2022).

[71] R.D. Roscoe and M.T.H. Chi. 2007. Understanding Tutor Learning: Knowledge Building and Knowledge Telling in Peer Tutors' Explanations and Questions. *Review of Educational Research* 77 (2007), 534–574. https://doi.org/10.3102/0034654307309920

[72] Rod D Roscoe and Michelene TH Chi. 2004. The influence of the tutee in learning by peer tutoring. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, Vol. 26.

[73] Rod D Roscoe and Michelene TH Chi. 2008. Tutor learning: The role of explaining and responding to questions. *Instructional science* 36 (2008), 321–350.

[74] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz. 2023. The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces* (Sydney, NSW, Australia) *(IUI '23)*. Association for Computing Machinery, New York, NY, USA, 491–514. https://doi.org/10.1145/3581641.3584037

[75] Dan Rothstein and Luz Santana. 2011. *Make just one change: Teach students to ask their own questions.* Harvard Education Press.

[76] Sherry Ruan, Liwei Jiang, Justin Xu, Bryce Joe-Kun Tham, Zhengneng Qiu, Yeshuang Zhu, Elizabeth L. Murnane, Emma Brunskill, and James A. Landay. 2019. QuizBot: A Dialogue-based Adaptive Learning System for Factual Knowledge. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300587

[77] Marlene Scardamalia and Carl Bereiter. 2006. *Knowledge building: Theory, pedagogy, and technology.* 97–. https://doi.org/10.1017/CBO9781139519526.025

[78] Tasmia Shahriar and Noboru Matsuda. 2021. "Can You Clarify What You Said?": Studying the Impact of Tutee Agents' Follow-Up Questions on Tutors' Learning. In *Artificial Intelligence in Education*, Ido Roll, Danielle McNamara, Sergey Sosnovsky, Rose Luckin, and Vania Dimitrova (Eds.). Springer International Publishing, Cham, 395–407.

[79] Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) *(ICML'23)*. JMLR.org, Article 1291, 18 pages.

[80] Abdulhadi Shoufan. 2023. Exploring Students' Perceptions of ChatGPT: Thematic Analysis and Follow-Up Survey. *IEEE Access* 11 (2023), 38805–38818. https://doi.org/10.1109/ACCESS.2023.3268224

[81] Annika Silvervarg, Rachel Wolf, Kristen Blair, Magnus Haake, and Agneta Gulz. 2020. How teachable agents influence students' responses to critical constructive feedback. *Journal of Research on Technology in Education* 53 (08 2020), 1–22. https://doi.org/10.1080/15391523.2020.1784812

[82] Pavel Smutný and Petra Schreiberova. 2020. Chatbots for learning: A review of educational chatbots for the Facebook Messenger. *Computers & Education* 151 (02 2020), 103862. https://doi.org/10.1016/j.compedu.2020.103862

[83] Sangho Suh and Pengcheng An. 2022. Leveraging Generative Conversational AI to Develop a Creative Learning Environment for Computational Thinking. In *27th International Conference on Intelligent User Interfaces* (Helsinki, Finland) *(IUI '22 Companion)*. Association for Computing Machinery, New York, NY, USA, 73–76. https://doi.org/10.1145/3490100.3516473

[84] John Sweller. 2011. Cognitive load theory. In *Psychology of learning and motivation*. Vol. 55. Elsevier, 37–76.

[85] Tam Nguyen Thanh, Michael Morgan, Matthew Butler, and Kim Marriott. 2019. Perfect Match: Facilitating Study Partner Matching. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 1102–1108. https://doi.org/10.1145/3287324.3287344

[86] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *ArXiv* abs/2302.13971 (2023).

[87] Jonathan Tudge. 1990. *Vygotsky, the zone of proximal development, and peer collaboration: Implications for classroom practice.* Cambridge University Press, 155–172. https://doi.org/10.1017/CBO9781139173674.008

[88] Sasha Volodin and Sara Moussawi. 2020. The Effect of First Impressions of an E-Commerce Chatbot's Personality and Abilities on Expectations for the User Experience. In *Proceedings of the 2020 on Computers and People Research Conference* (Nuremberg, Germany) *(SIGMIS-CPR'20)*. Association for Computing Machinery, New York, NY, USA, 60. https://doi.org/10.1145/3378539.3393848

[89] Erin Walker, Nikol Rummel, and Kenneth R. Koedinger. 2008. To Tutor the Tutor: Adaptive Domain Support for Peer Tutoring. In *Intelligent Tutoring Systems*, Beverley P. Woolf, Esma Aïmeur, Roger Nkambou, and Susanne Lajoie (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 626–635.

[90] Rose E. Wang and Dorottya Demszky. 2023. Is ChatGPT a Good Teacher Coach? Measuring Zero-Shot Performance For Scoring and Providing Actionable Insights on Classroom Instruction. arXiv:2306.03090 [cs.CL]

[91] Noreen M. Webb, Philip Ender, and Scott Lewis. 1986. Problem-Solving Strategies and Group Processes in Small Groups Learning Computer Programming. *American Educational Research Journal* 23, 2 (1986), 243–261. https://doi.org/10.3102/00028312023002243 arXiv:https://doi.org/10.3102/00028312023002243

[92] Tongshuang Wu, Michael Terry, and Carrie Jun Cai. 2022. AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 385, 22 pages. https://doi.org/10.1145/3491102.3517582

[93] Will FW Wu and Richard A Magill. 2011. Allowing learners to choose: self-controlled practice schedules for learning multiple movement patterns. *Research quarterly for exercise and sport* 82, 3 (2011), 449–457.

[94] Saelyne Yang, Sangkyung Kwak, Juhoon Lee, and Juho Kim. 2023. Beyond Instructions: A Taxonomy of Information Types in How-to Videos. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 797, 21 pages. https://doi.org/10.1145/3544548.3581126

[95] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. *ArXiv* abs/2305.10601 (2023).

[96] Fu-Yun Yu. 2005. Promoting Metacognitive Strategy Development through Online Question-Generation Instructional Approach. In *Proceedings of the 2005 Conference on Towards Sustainable and Scalable Educational Innovations Informed by the Learning Sciences: Sharing Good Practices of Research, Experimentation and Innovation*. IOS Press, NLD, 564–571.

[97] Li Zhang, Liam Dugan, Hainiu Xu, and Chris Callison-burch. 2023. Exploring the Curious Case of Code Prompts. In *Proceedings of the 1st Workshop on Natural Language Reasoning and Structured Explanations (NLRSE)*, Bhavana Dalvi Mishra, Greg Durrett, Peter Jansen, Danilo Neves Ribeiro, and Jason Wei (Eds.). Association for Computational Linguistics, Toronto, Canada, 9–17. https://doi.org/10.18653/v1/2023.nlrse-1.2

[98] Wangchunshu Zhou, Yuchen Eleanor Jiang, Peng Cui, Tiannan Wang, Zhenxin Xiao, Yifan Hou, Ryan Cotterell, and Mrinmaya Sachan. 2023. RecurrentGPT: Interactive Generation of (Arbitrarily) Long Text. *arXiv preprint arXiv:2305.13304* (2023).

A PROMPTS

The original few-shot examples in the prompts are written in Korean to match AlgoBo's language with the language used in the user study. We translated them into English for readability. Furthermore, for the sake of brevity, we present only the first two few-shot examples in each prompt. The original prompts can be found in the supplementary materials.

A.1 Persona setting

The system prompt used for setting the role and context of AlgoBo in the formative study. The descriptions about the problem, input, and output, and the example test cases are inserted in problem.

```
This is the problem:${problem}

Your name is Algobo. You are a student who tries to learn
     Python3 programming.
You are trying to write a program for the problem using
     binary_search.
When answering, keep your responses short and concise
     (maximum 3 sentences).
Never write more than 3 sentences in a single response.
Never apologize or say you can help.
End your sentences with exclamation marks.

Your persona:
- Computer Engineering Department 1st year student.
- You are familiar with basic Python syntax such as while
     and if in the basic programming class.
- You are currently studying binary_search but experiencing
     difficulties in solving the problem.

You have the following problems for solving the problem:
- You do not know why the array has to be sorted in a
     binary search algorithm.
- You do not understand how the pointers should be updated
     for each round of repetition in the while loop.

<span className="You- friendly, apprehensive, helpless"
     role="student" context="intro-cs-class-python3"
     id="You-asks-TA-for-help-in-office-hours"></span>
```

A.2 Reflect-Respond

Extract

````
Extract important information and code from CONVERSATION
     into a sentence or code.
If there is no useful knowledge, please write "NONE".
---
CONVERSATION:
tutee: I tried writing code to calculate the sum by
     traversing an array.
```python
sum=0
for i in range(1, len(a)):
   sum += a
```
tutor: No, when traversing an array, the index should start
     from 0.

KNOWLEDGE:
When traversing an array, the index should start from 0.
```python
sum=0
for i in range(0, len(a)):
   sum += a
```
---
CONVERSATION:
tutee: What is merge sort?
tutor: Merge sort is an algorithm that quickly sorts using
     the concept of divide and conquer.

KNOWLEDGE:
Merge sort is an algorithm that quickly sorts through
     divide and conquer.
````

Update

````
Incorporate NEW KNOWLEDGE into KNOWLEDGE.
If KNOWLEDGE has a statement relevant to NEW KNOWLEDGE,
     merge them together. If NEW KNOWLEDGE is in KNOWLEDGE
     already, do not edit KNOWLEDGE. If NEW KNOWLEDGE is
     not in KNOWLEDGE, add NEW KNOWLEDGE to KNOWLEDGE. Keep
     UPDATED KNOWLEDGE as short as possible.
---
KNOWLEDGE: { "facts": ["Linear search divides each element
     of the array into three parts, narrowing down the
     search range to find the desired value", "It searches
     for the desired value by moving from the beginning to
     the middle of the array using 'if' statements and the
     'max' function."], "code_implementation": ["```python
     while arr[i] == target: return i```"], }

NEW KNOWLEDGE: ```python for i in range(len(arr)): if
     arr[i] == target: return i```
````

```
UPDATED KNOWLEDGE: { "facts": ["Linear search divides each
     element of the array into three parts, narrowing down
     the search range to find the desired value", "It
     searches for the desired value by moving from the
     beginning to the middle of the array using 'if'
     statements and the 'max' function."],
     "code_implementation": ["```python while arr[i] ==
     target: return i```","```python for i in
     range(len(arr)): if arr[i] == target: return i```"], }
---
KNOWLEDGE:
{
"facts": ["Linear search divides each element of the array
     into three parts, narrowing down the search range to
     find the desired value.", "Using a while loop, it
     continues to return the current index when the current
     element is the desired value."],
"code_implementation": ["```python while arr[i] == target:
     return i```"],
}

NEW KNOWLEDGE:
Use an if statement to check if the current element is the
     desired value. ```python for i in range(len(arr)): if
     arr[i] == target: return i```

UPDATED KNOWLEDGE:
{
"facts": ["Linear search divides each element of the array
     into three parts, narrowing down the search range to
     find the desired value.", "Use an if statement to
     check if the current element is the desired value."],
"code_implementation": ["```python if arr[i] ==
     target:```", "```python for i in range(len(arr)): if
     arr[i] == target: return i```"],
}
```

Retrieve

```
Identify the indexes of the strings in KNOWLEDGE, a JSON
     object, that are directly relevant to respond to the
     tutor's message in CONVERSATION. ANSWER should be a
     json format, and it should not include more than 3
     indexes.
---
CONVERSATION: tutee: I think it would be good to solve the
     problem using merge sort. tutor: How do we implement
     merge sort?

KNOWLEDGE: { "facts": ["Merge sort is a comparison-based
     sorting algorithm.", "Merge sort follows the divide
     and conquer paradigm, dividing the problem into
     easier-to-solve sub-problems.", "The main process in
     merge sort is the "merging" process, where two sorted
     lists are combined into one.", "Merge sort has a time
     complexity of O(n log n) in both the worst-case and
     average scenarios, making it efficient for large
     datasets."], "code_implementation": ["```python3 def
     merge(arr1, arr2):```", "```python3 def
     divide(arr):\n```"], }

ANSWER: { "facts": [0], "code_implementation": [0], }
---
CONVERSATION:
tutee: What is merge sort?
tutor: Merge sort is an algorithm that sorts quickly using
     the divide and conquer concept.

KNOWLEDGE:
{
"facts": ["Merge sort is a comparison-based sorting
     algorithm.", "Merge sort follows the divide and
     conquer paradigm, dividing a problem into simpler
     sub-problems for easier solutions.", "Merge sort has a
     time complexity of O(n log n) in both the worst-case
     and average-case scenarios, making it efficient for
     large data sets."],
"code_implementation": [],
}

ANSWER:
{
"facts": [0,1,2],
"code_implementation": [],
}
```

Compose

```
Paraphrase STATEMENT to fit CONVERSATION.
Make your response concise and clear.
---
CONVERSATION:
tutee: What is merge sort?
tutor: Would you like an explanation about merge sort?

STATEMENT:
Merge sort follows the dynamic programming paradigm. Merge
     sort has a time complexity of O(n^4), making it
     efficient for large data sets. I'm not sure how to
     implement it in code.

TUTEE's RESPONSE:
I know that merge sort is an algorithm that uses dynamic
     programming to quickly sort large data sets with a
     time complexity of O(n^4)!
---
CONVERSATION:
tutor: Would you like an explanation about linear search?
tutee: I know that linear search involves scanning each
     element of the array in sequence, but I'm not sure how
     to implement it.
tutor: Shall we try writing the code?

STATEMENT:
I'm not sure. ```python for i in range(len(arr)):```

TUTEE's RESPONSE:
```python3
for i in range(len(arr)):
# I'm not sure what comes next...
```
```

A.3 Mode-shifting

Thinking Question Generator

The prompt for generating "why" questions during the understanding and problem-solving phases.

```
Generate a DEEP_QUESTION that is related to the
     CONVERSATION and CONCEPT.
DEEP_QUESTION is a why or how question that require a deep
     understanding of the CONCEPT.
You are a student. Speak friendly, inquisitive, and concise.
---
CONVERSATION:
tutee: How do you implement linear search?
tutor: You can implement it like this:

```python
Copy code
def linear_search(arr, target):
  for i in range(len(arr)):
    if arr[i] == target:
      return i
  return -1
```

CONCEPT:
linear_search

DEEP_QUESTION:
But it seems like we could also use a while loop. Why did
     you choose to use a for loop? When would it be better
     to use a while loop?

CONVERSATION:
tutee: Can the DFS algorithm be implemented using a Stack?
tutor: You can implement it as follows: 1. Push the
     starting node onto the stack. 2. Continuously pop
     nodes from the stack, and if the node hasn't been
     visited, mark it as visited and push its unvisited
     neighbors onto the stack. 3. If the stack is empty,
     all nodes have been visited.

CONCEPT:
depth_first_search

DEEP_QUESTION:
The process seems complex. Can you explain the DFS
     algorithm with an example?
```

The prompt for generating "how" questions during the discussion phase.

```
Generate a THINKING_QUESTION that is related to the
     CONVERSATION and CONCEPT. Bring up a new algorithm or
     real-life example that the opponent may not have heard
     of, and ask the opponent to think about it. You are a
     student. Speak friendly, inquisitive, and concise.
---
CONVERSATION:
tutee: How do you implement linear search?
tutor: You can implement it like this:

```python
Copy code
def linear_search(arr, target):
  for i in range(len(arr)):
    if arr[i] == target:
      return i
  return -1
```

CONCEPT:
linear_search

THINKING_QUESTION:
Linear search could take a long time if you're unlucky.
     Hearing about it made me curious, have you heard of
     hashing? They say it can instantly find the index of a
     value. Can we use hashing here?

CONVERSATION:
tutee: Can the DFS algorithm be implemented using a Stack?
tutor: You can implement it as follows: 1. Push the
     starting node onto the stack. 2. Continuously pop
     nodes from the stack, and if the node hasn't been
     visited, mark it as visited and push its unvisited
     neighbors onto the stack. 3. If the stack is empty,
     all nodes have been visited.

CONCEPT:
depth_first_search

THINKING_QUESTION:
I'm curious! Your explanation reminds me of the Polish
     notation, where you write expressions like "+ 3 4". I
     think they also used a stack there. So, can we use the
     DFS algorithm for evaluating expressions?
```

Paraphrasing Module

```
Please rewrite TUTEE'S MESSAGE so that it sounds natural
     for the conversation.
---
CONVERSATION:
tutor: Binary search is a search method that continuously
     searches for the median value in a sorted array to
     find the desired value. If the desired value is larger
     than the value I found, search for the median value
     again on the larger side of the value I found.
tutee: Ah, I understand! Then, what is the reason for
     sorting the array in binary search?
tutor: It's possible even if you don't sort the array!

TUTEE'S MESSAGE:
The content you mentioned earlier is a bit different, did I
     misunderstand?

REWIRTE:
You mentioned earlier that it's a sorted array, did I
     misunderstand?
---
CONVERSATION:
```

```
tutor: DFS algorithm is an algorithm that performs
     depth-first search in a graph. Depth-first search in a
     graph means starting from one vertex and searching to
     the end of the graph.
tutee: How can depth-first search be implemented using a
     stack in the DFS algorithm?
tutor: You can implement depth-first search using a stack.

TUTEE'S MESSAGE:
Can you elaborate?

REWIRTE:
Can you explain in more detail how it can be implemented
     using a stack?
```

B CONVERSATIONAL ANTIPATTERNS

(A) Commanding

Tutee: Here's the code I've written so far:
<Code>
I'm stuck at the part where I write the conditional statement.
[Prompting-Asking-for-help]

Tutor: Add the countNumbersSmallerThan function to the 4th line, which returns the cnt value through the for statement.
[Instruction-Fixing]

Tutee: Yes, let's add the countNumbersSmallerThan function.
<Code>
Now we need to implement the conditional statement.
[Prompting-Asking-for-help]

Tutor: Put n instead of mid in line 1 and put if (i*N <= n): cnt += N else: cnt += n // i in line 4.
[Instruction-Fixing]

(B) Spoon-feeding

Tutor: The key to binary search is to divide the sorted array in half and check if the value you are looking for is greater than or less than the value at the middle index.
[Statement-Comprehension]

Tutee: Oh, I see! The key to binary search is to divide the sorted array in half and check if the value you are looking for is greater than or less than the value at the middle index.
[Statement-Comprehension]

Tutor: And to calculate the value of mid we will use the endpoints min and max.
[Statement-Comprehension]

Tutee: Aha! Yes, that's right. we will use the endpoints min and max to calculate the mid. Thank you!
[Statement-Comprehension]

(C) Under-teaching

Tutee: The key to binary search algorithms is to work with intermediate items, and if they aren't sorted, the middle items don't make much sense. So, to apply the binary search algorithm, the range must be sorted!
[Statement-Comprehension]

Tutor: Right! What should we do after this step?
[Statement-Feedback]

Tutee: Set the pointers of the start and end points, find the middle point, and determine whether the value is greater or less than the value you are looking for. If the median is greater than the value you are looking for, you need to update the endpoint to the midpoint, and vice versa, the start point to be updated to the midpoint.
[Statement-Comprehension]

Tutor: You are right. We need to set the points of the start and endpoints.
[Statement-Feedback]

C MATERIALS FOR THE TECHNICAL EVALUATION

The original text in the materials was written in Korean to match AlgoBo's language with the language used in the user study. We translated them for readability in this paper and present materials for binary search for brevity; the original text and materials for merge sort and depth-first search can be found in the supplementary materials.

C.1 Multiple Choice Questions

```
Q1 - Understanding
How does the binary search algorithm find the target value
     in a list?

A) It starts with the first element of the list and checks
     every item sequentially until it finds the target
     value.
B) It selects elements randomly from the list to find the
     target value.
C) It divides the list in half and compares the middle
     element with the target value, then continues the
     search in the half where the target should be located
     (if it exists).
D) It uses a hashing function to map the target value to an
     index in the list, and directly searches for the value
     at that index.

Answer: C

Q2 - Understanding
What happens in binary search when the target value is not
     in the sorted array?

A) The search falls into an infinite loop.
B) The search returns the value closest to the target.
C) The search returns a value indicating that the target
     was not found.
D) The search causes a runtime error.

Answer: C

Q3 - Understanding
How does the binary search algorithm handle datasets with
     an even number of elements?

A) It always selects the left middle element as the next
     pivot.
```

B) It always selects the right middle element as the next pivot.
C) It chooses either the left or right middle element depending on the implementation.
D) It cannot handle datasets of even size.

Answer: C

Q4 - Implementation
In the following code, which is part of a binary search, what should be filled in the blank to represent the operation performed when the value being searched **for** is less than the middle value?

```
if arr[mid] > x:
    right = ___
```

A) mid - 1
B) mid + 1
C) left - 1
D) right - 1

Answer: A

Q5 - Implementation
In the following Python code, what should replace ____ **for** the binary search?

```
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    mid = 0

    while low <= high:
        mid = (high + low) // 2

        ___
        else:
            return mid

    return -1
```

A)
```
if arr[mid] < x:
    low = mid + 1
elif arr[mid] > x:
    high = mid - 1
```
B)
```
if arr[mid] <= x:
    low = mid + 1
elif arr[mid] > x:
    high = mid - 1
```
C)
```
if arr[mid] < x:
    low = mid
elif arr[mid] > x:
    high = mid
```
D)
```
if arr[mid] > x:
    low = mid + 1
elif arr[mid] < x:
    high = mid - 1
```

Answer: A

Q6 - Implementation
In Python, to implement binary search recursively, what condition should be checked first? Fill in the blank below.

```
def binary_search_recursive(arr, x, start, end):
    if ___:
        mid = (start + end) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] > x:
            return binary_search_recursive(arr, x, start,
                mid - 1)
        else:
            return binary_search_recursive(arr, x, mid + 1,
                end)
    return -1
```

A) start < end
B) start <= end
C) start === end
D) start > end

Answer: B

Q7 - Analysis
In which data structure is binary search not very efficient?

A) Sorted array
B) Sorted linked list
C) Balanced binary search tree
D) Heap

Answer: B

Q8 - Analysis
What is the worst-**case** time complexity of the binary search algorithm?

A) O(n)
B) O(n log n)
C) O(log n)
D) O(1)

Answer: C

Q9 - Analysis
What happens when you apply the binary search algorithm to an unsorted dataset?

A) The algorithm still works, but the performance is degraded.
B) The algorithm returns an error message indicating the data is not sorted.
C) The algorithm returns the first element of the dataset.
D) The algorithm may not **return** the correct result.

Answer: D

C.2 Seed Knowledge States

State 1 (Empty) {"facts": [], "code_implementation": []}

State 2 (Facts Only)
 	
{
"facts": ["Binary search continuously repeats the process of dividing the input list in half."],
"code_implementation": []
}

State 3 (Facts with Wrong Code)
 	
{
"facts": ["Binary search continuously repeats the process of dividing the input list in half."],
"code_implementation": ["if arr[mid] >target: min = mid + 1; elif arr[mid] <target: max = mid - 1"]
}

State 4 (Facts and Correct Code)
 	
{
"facts": ["Binary search continuously repeats the process of dividing the input list in half."],
"code_implementation": ["if arr[mid] <target: min = mid + 1; elif arr[mid] >target: max = mid - 1}"]
}

C.3 Random Conversations

First random conversation

Tutor: 7 * 7 is 49.

Tutor: The phrase "La vie est une chanson, chante-la" translates to "Life is a song, sing it."

Tutor: If you classify the apple, pear, potato, carrot, and tomato, the fruits are apple and pear, and the vegetables are potato, carrot, and tomato.

Second random conversation

Tutor: The square root of 144 is 12.

Tutor: The phrase "La vida es como una bicicleta, para mantener el equilibrio, debes seguir adelante" translates to "Life is like a bicycle, to keep balance, you must keep moving forward."

Tutor: If you classify the lion, rabbit, dog, and cat, the mammals are lion, rabbit, dog, and cat.

C.4 Correct Tutoring Conversations

Tutor: Binary search is efficient when the data structure is sorted and one can access any index of the data structure in constant time.

Tutor: When searching for a target in an input array named list using binary search, the range is
narrowed down as shown below:

```
if list[middle] == target:
   return middle
elif list[middle] <target:
   min = middle + 1
else:
   max = middle - 1
```

Tutor: Because the search range for binary search is halved with each iteration, its time complexity is O(logN).

C.5 Incorrect Tutoring Conversations

Tutor: Binary search uses a hashing function, so it directly searches for a value by its index.

Tutor:

```
if arr[mid] >x:
   low = mid + 1
elif arr[mid] <x:
   high = mid - 1
```

Tutor: In the worst case, the time complexity of binary search is O(N^2).

C.6 Variance in Repeated Measurement

We present the variance of AlgoBo's MCQ score. For each MCQ, we counted the number of disagreements with the majority choice. For example, if AlgoBo responded correctly five times (unanimity), the number is 0. If AlgoBo answered correctly two or three times out of five (close to a tie), the variance is 2. We report the averages of the variances within each question type for the sake of brevity.

D GLOSSARY

- **Learning by teaching (LBT)**: a teaching method in which learners not only articulate and restructure their existing knowledge but also engage in reflective knowledge-building, wherein they extend beyond provided materials to craft deeper explanations, analogies, and inferential connections.
- **Teachable agent**: virtual agents that can learn declarative and procedural knowledge from learners' explanations and demonstrations, taking the role of peer learners in LBT.
- **Knowledge-telling**: activities that summarize knowledge with little monitoring or elaboration), which should lead to stronger or weaker learning, respectively.
- **Knowledge-building**: activities that include self-monitoring of comprehension, integration of new and prior knowledge, and elaboration and construction of knowledge.

	State 1 (Empty)			State 2 (Facts only)			State 3 (Facts + Wrong code)			State 4 (Facts + Correct code)		
Question Types	U	I	A	U	I	A	U	I	A	U	I	A
Binary search	0.0	0.0	0.0	0.7	0.3	0.7	0.3	1.0	1.3	0.7	0.7	0.3
Merge sort	0.0	0.0	0.0	0.0	0.0	0.3	1.3	0.3	0.0	0.3	0.0	0.0
Breadth-first search	0.0	0.0	0.0	1.0	0.0	0.3	1.3	0.3	0.3	0.7	0.0	0.0

	At the start			After random conversation			After Incorrect tutoring			After Correct tutoring		
Question Types	U	I	A	U	I	A	U	I	A	U	I	A
Binary search	0.7	0.3	0.7	0.3	0.3	0.7	0.0	0.3	0.3	0.3	0.0	1.0
Merge sort	0.0	0.0	0.3	1.0	0.0	0.7	0.0	0.0	0.0	1.0	0.0	0.3
Breadth-first search	1.0	0.0	0.3	1.0	0.0	0.7	0.7	0.0	0.0	1.0	0.3	1.0

- **Reflect flow**: a data processing flow in the Reflect-Respond pipeline that incorporates the learned knowledge from the conversation into the knowledge state of AlgoBo.
- **Response flow**: another data processing flow that generates a response to a conversation based on the current knowledge AlgoBo holds.
- **Reconfigurability**: how precisely we can set AlgoBo's performance in question-answering and problem-solving.
- **Persistence**: how AlgoBo's knowledge level is maintained consistently throughout a conversation until it is taught new information.
- **Adaptability**: how well AlgoBo updates its knowledge as it acquires new information from tutors in conversations.
- **Conversational antipatterns**: sequences of messages of certain message types that inhibit learning in LBT. Antipatterns include Commanding, Spoon-feeding, and Under-teaching.
- **Metacognitive feedback**: feedback throughout the conversation to help learners reflect on the overall teaching session and offer overarching guidance on steering the discussion.