석 사 학 위 논 문
Master's Thesis

학습자와 교사가 스스로 맞춤형 학습 도구를 만들도록 돕는 시스템 설계

Designing Systems for Empowering Learners and Teachers to Create Adaptive Learning Tools

2025

진 형 욱 (陳 炯 旭 Jin, Hyoung Wook)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

석 사 학 위 논 문

학습자와 교사가 스스로 맞춤형 학습 도구를 만들도록 돕는 시스템 설계

2025

진 형 욱

한 국 과 학 기 술 원

전산학부

학습자와 교사가 스스로 맞춤형 학습 도구를 만들도록 돕는 시스템 설계

진 형 욱

위 논문은 한국과학기술원 석사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2025년 6월 11일

심사위원장 김 주 호 (인)

심 사 위 원 이 의 진 (인)

심 사 위 원 이 탁 연 (인)

Designing Systems for Empowering Learners and Teachers to Create Adaptive Learning Tools

Hyoung Wook Jin

Advisor: Juho Kim

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Daejeon, Korea
June 11, 2025

Approved by

Juho Kim
Professor of Computer Science

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

MCS

진형욱. 학습자와 교사가 스스로 맞춤형 학습 도구를 만들도록 돕는 시스템 설계. 전산학부 . 2025년. 106+vi 쪽. 지도교수: 김주호. (영문 논문)
Hyoung Wook Jin. Designing Systems for Empowering Learners and Teachers to Create Adaptive Learning Tools. School of Computing . 2025. 106+vi pages. Advisor: Juho Kim. (Text in English)

초 록

온라인 학습은 강의 비디오, 코드 예시, AI 기반 챗봇 등 다양한 디지털 학습 도구와의 지속적인 상호작용을 수반한다. 이러한 교육 기술에 대한 연구는 학습자의 고유한 니즈를 파악하고 적응형 학습 도구를 개발하는 데 중점을 두어왔다. 하지만, 연구자 중심의 하향식 개발 방식은 제한된 인력과 자원 때문에 다양한 학습자 인구 유형, 학습 성향, 학습 상황에 모두 대응하는 데 한계가 있다. 본 논문에서는 학습자와 교사가 컴퓨터 기술의 도움을 받아 스스로 적응형 학습 도구를 설계할 수 있는 자족적 접근 방식을 연구했다. 상향식 접근이란 개념 증명을 위해 CodeTree, TeachYou, TeachTune라는 세 가지 웹 기반 시스템을 구축하고 대학 프로그래밍 및 중학교 과학 교육 환경에서 효과를 실험해 보았다. 이러한 시스템에서 공통으로 학습자와 교사는 계층적 표현에 기반한 저작 인터페이스를 사용하여 적응형 학습 도구를 개발할 수 있다. 계층적 표현은 정보를 작은 단위로 나누는 구조를 지칭하며, 학습자 간 협력을 조율하는 중심이 되거나, AI를 세밀하게 조정하는 인터페이스로 기능하거나, 학습 체계적으로 검토하는 분석 도구로써의 가능성을 살펴보았다. 사용자 및 기술 실험을 진행한 결과, 계층적 표현을 기반으로 하는 저작 인터페이스를 사용하면 학습자와 교사가 적응형 학습 도구를 만들고, 협업하고, 검토할 수 있는 역량을 효과적으로 강화할 수 있다는 것을 확인했다. 이 논문은 사용자를 기술 설계의 중심으로 참여시키는 것을 목표하는 인간-컴퓨터 상호작용의 광범위한 분야 연구에 기여한다.

핵 심 낱 말 인간-컴퓨터 상호작용, 맞춤형 학습 도구, 계층적 표현

Abstract

Online learning involves continuous interaction with diverse digital learning tools, such as lecture videos, code examples, and AI-based chatbots. Research on these educational technologies has focused on capturing learners' idiosyncratic needs and developing adaptive learning tools. However, the researcher-driven, top-down approach for tool development struggles to scale to diverse learner demographics, preferences, and contexts due to its limited human resources. In this thesis, I explored a user-driven approach in which learners and teachers design adaptive learning tools independently with the support of computer technologies. As proofs of concept, I created three web-based systems—CodeTree, TeachYou, and TeachTune—and studied their efficacy in this approach within the contexts of college programming and middle school science education. In these systems, learners and teachers create adaptive learning tools using authoring interfaces based on hierarchical representations. Hierarchical representations refer to structures that divide information into smaller, more manageable units. We explored their potential as coordination hubs for collaborative work, as fine-grained interfaces for instructing AI, and as analytical tools for systematic reviews. User and technical experiments with the systems collectively showed that authoring interfaces based on hierarchical representations can effectively empower learners and teachers to create, collaborate on, and review learning tools. The thesis contributes to broader fields of human-computer interaction that aim to engage users in the design of their educational technologies.

Keywords Human-computer interaction, adaptive learning tools, and hierarchical representations

Contents

List of Tables

List of Figures

Chapter 1. Introduction

The growth of online education has transformed the landscape of learning, making education more accessible and scalable than ever before. Learners today engage in a variety of digital learning environments where they interact continuously with learning tools such as lecture videos, code examples, and AI-based chatbots. These interactions create rich opportunities for adaptive learning experiences, yet they also pose critical challenges in meeting the heterogeneous needs of learners.

Traditionally, research in educational technologies—particularly in the domains of human-computer interaction (HCI) and educational technology (EduTech)—has focused on designing adaptive learning tools from a top-down perspective. Researchers and engineers utilized theoretical insights, data analysis, and engineering skills to identify learner needs and develop these tools. While this approach has generated impactful systems, it struggles to scale in a world where learner demographics, preferences, and contexts are extensively diverse. Manually encoding adaptivity for every emerging need is labor-intensive, and the top-down approach lacks the scalability to keep pace with the dynamic realities.

To address this challenge, this thesis explores a crowdsourcing-inspired approach to adaptive learning tools—one that empowers learners and teachers to take an active role in designing and customizing their learning tools. This approach draws inspiration from participatory design and end-user programming in HCI, which advocate for involving users in shaping the tools they use. Recent advances in collaborative computing and artificial intelligence have made it increasingly feasible to expand such user involvement from ideation to creation.

For instance, this approach can facilitate the dissemination of interactive learning, where learners interact with a social agent to acquire knowledge and skills. Although interactive learning is known to be more effective than passive and active forms [1], learning tools for it require considerable effort for subject-specific programming [2] and customizing for individual classes, limiting their applicability to a few subjects and students. With a general interface for authoring an effective learning tool, learners and teachers can create their own downstream versions, thereby increasing the collective scalability of the tool.

In this thesis, I propose and evaluate systems that enable learners and teachers to create adaptive learning tools using hierarchical representations. These representations, which break information into groups of smaller entities, can empower users to collaborate on, compose, and overview the elements of learning tools. By increasing the accessibility to and controllability in designing educational interactions, these systems enable more fine-grained and context-sensitive adaptivity.

To demonstrate and evaluate the user-driven approach, I present three web-based systems—CodeTree, TeachYou, and TeachTune—each situated in a distinct educational context and each leveraging hierarchical representations to facilitate different facets of the approach:

- CodeTree enables learners to create adaptive explanations for programming code examples collaboratively. It introduces *subgoal hierarchies*, a representation that captures the functional structure of code in a hierarchical form. This structure allows learners to contribute to different parts of the hierarchy, facilitating collaborative construction while simplifying the integration of fragmented inputs. A user study showed that this collaborative creation can produce high-quality adaptive explanations.

- TeachYou is a learning platform that helps learners review algorithms by teaching an AI chatbot. Both learners and teachers can customize the chatbot's knowledge level for their adaptive learning through a *knowledge hierarchy*—a representation that separates knowledge into independent skills. This structure allows learners and teachers to define the chatbot's multi-faceted knowledge with fine granularity. A technical evaluation and user study demonstrated that knowledge states effectively control the chatbot's knowledge level and that learning-by-teaching with the chatbot can enhance learning.
- TeachTune is a teacher-facing tool designed to support the creation and review of pedagogical chatbots adaptive to students in their classes. During the iterative development process, the system provides simulated students that allow teachers to test and refine their chatbots. These simulated students are configured through a *trait hierarchy*, a representation that models student behavior by specifying levels of individual traits. A user study showed that the representation can help teachers review their chatbots more systematically, efficiently, and comprehensively.

A central theme across these systems is the use of hierarchical representations—modular structures that allow for flexible manipulation, reuse, and aggregation of learning components. Whether they are subgoal hierarchies, knowledge hierarchies, or trait hierarchies, these representations provide a common language for learners and teachers to articulate and implement their instructional intents.

This thesis thus contributes to the broader field of human-computer interaction by demonstrating how end-users can be empowered to design adaptive learning experiences with the help of intelligent and interactive systems. By bridging the gap between research-oriented development and the scale of individual needs of learners, the proposed user-driven approach offers a path forward for more inclusive, responsive, and participatory educational technologies.

Thesis statement: Authoring interfaces based on hierarchical representations can empower learners and teachers to create quality learning tools.

Chapter 2. CodeTree

This chapter presents the first example of the user-driven approach to adaptive learning: CodeTree, a system that coordinates programming learners' collaborative authoring of adaptive explanations of code examples. This chapter explores how an authoring interface based on subgoal hierarchies affects the quality and efficiency of learner-driven collaborative authoring. The content in this chapter is adapted, updated, and rewritten from our prior work published at CSCW 2024 [3]. Throughout this chapter, the pronouns "we," "our," and "us" refer to the coauthors of that publication.

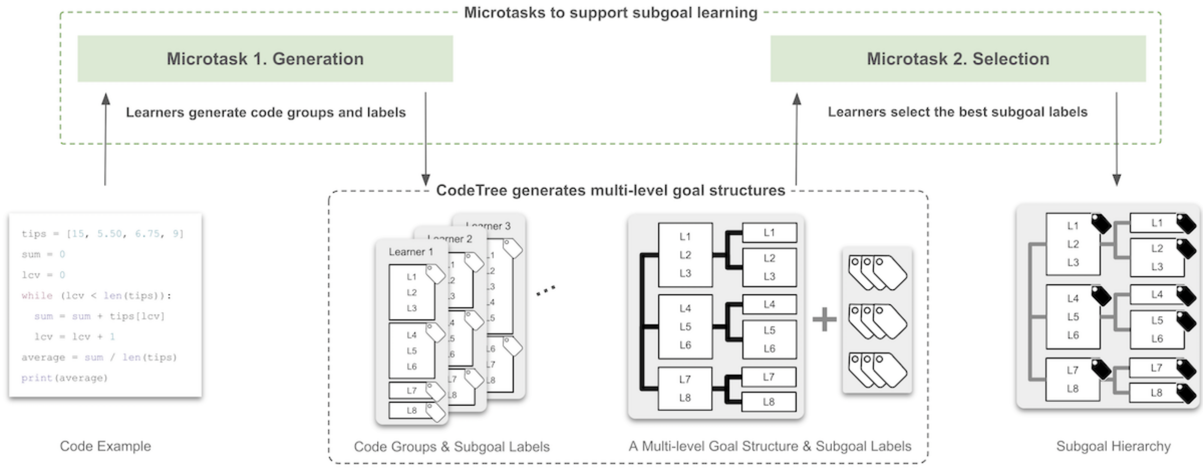

Figure 2.1: Our learnersourcing workflow for generating subgoal hierarchies. For each code example, learners in the *Generation* task generate the code groups and subgoal labels that can constitute a subgoal hierarchy. Our hierarchy generation algorithm aggregates the code groups into a multi-level goal structure. Learners in the *Selection* task receive the goal structure and pre-populated subgoal labels to vote for the best subgoal labels for each subgoal in the hierarchy.

2.1 Motivation and Contribution

Programming is becoming one of the most valuable skills to learn. As the programming population grows, online resources such as documentation, how-to videos, and Q&A websites have become popular for learning and help-seeking [1]. Code examples are typical materials used in programming learning resources. Code examples are short code snippets that demonstrate instantiations of code patterns under specific problem contexts (*e.g., calculating the average value of coins in a pocket*). Since the exemplified problem contexts are often different from the diverse problem contexts that programmers face in practice (*e.g., calculating the average of positive values in an array*) [4], practitioners and learners need to spot and modify parts of code examples to adapt them to their problem contexts. Hence, the educational purpose of code examples is not to have learners simply copy code but to reduce their cognitive load and provide the means to learn and transfer code patterns to novel problem contexts [5].

Education research has shown that learning subgoals in code examples can effectively scaffold learners' transfer to novel problems [6, 7, 8]. Subgoals are functional units that divide code into smaller pieces

[1] https://insights.stackoverflow.com/survey/2018

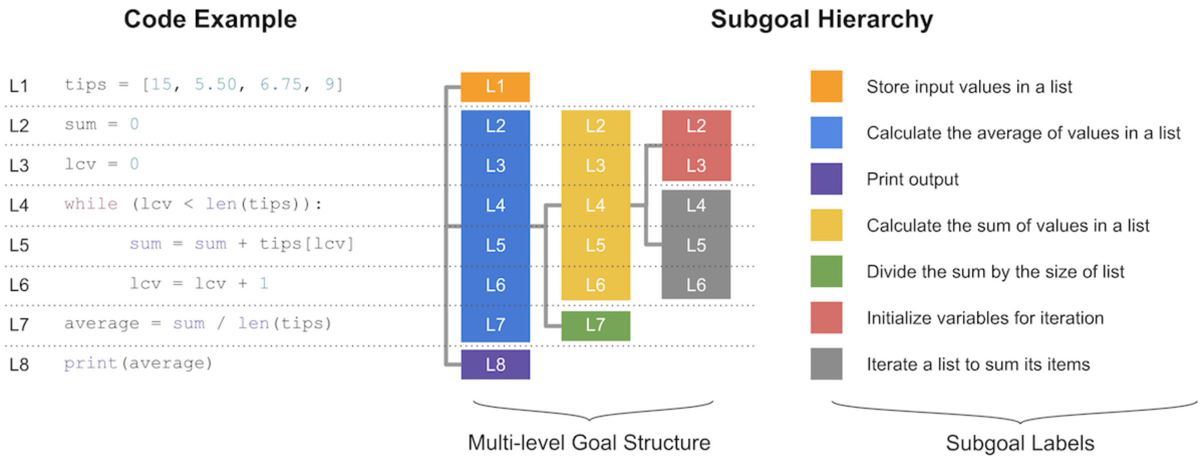

Figure 2.2: A subgoal can group a set of related code lines by their function. The subgoals on the left of the hierarchy are coarse-grained goals that explain high-level functions that span multiple lower-level subgoals (or code lines). The subgoals on the right are fine-grained goals that explain the code line by line.

and help learners navigate code to find the part to modify for different problem contexts. Subgoals often form a hierarchy with high-level strategic goals and low-level constituent goals comprising the high-level goals. A subgoal hierarchy refers to a hierarchical organization of subgoals in code and consists of a goal structure and subgoal labels (Fig. 2.2). Subgoal hierarchies are used to explain the function of each part of code [9, 10], to create the materials for diverse learning activities [11, 12], and to generate adaptive explanations [13]. Goal structures are a fundamental component of subgoal hierarchies to facilitate these learning supports as they set the frameworks for organizing and mapping subgoals to code.

The generation of goal structures has been dependent on expertsourcing and data-driven methods. Conventionally, instructors and domain experts take an iterative process to generate subgoals [14], but this is limited in terms of scalability as the process is time-consuming. To overcome the limitation, data-driven methods have been applied to learning environments where large code datasets are available (e.g., Scratch) [15, 16, 17]. These methods compare learners' code submissions for a problem at scale and identify common code patterns as subgoals of the problem. However, data-driven methods do not work in learning environments where code data per code example is scarce. For instance, although StackOverflow has many code examples, more than half of their questions are answered with less than ten code examples, which is not a feasible scale to adopt data-driven approaches [18].

To support the subgoal learning of code examples in ubiquitous environments, we propose using learnersourcing to generate goal structures without dedicated experts or large-scale code data. Learnersourcing is a scalable crowdsourcing technique that leverages learners' creativity and knowledge to create learning resources for future learners [19, 20]. Prior research has shown that learnersourcing could effectively reduce experts' effort in subgoal label generation by offloading certain tasks to learners [19, 12]. We extend this line of research further by proposing an approach to also offload the effort of generating goal structures to learners. Compared to learnersourcing of subgoal labels, generating goal structures is new and challenging because it has to build the structure from scratch while label-sourcing works on top of a given structure. Despite the challenge, learnersourcing of goal structure is fundamental to prior subgoal label generation approaches for making the generation of subgoal hierarchies completely learner-driven and scalable in data-scarce environments.

We built a prototype system, CodeTree, to investigate the feasibility of learnersourcing the generation of goal structures without expert intervention. CodeTree is a tool for studying code examples by carrying out two learning activities that ask learners to either 1) group code lines into subgoals or 2) vote for the best explanations as the subgoal of a given code. These activities elicit self-explanation of goal structures as learners should make sense of the relationship between code and the subgoals they make or vote on. CodeTree leverages learners' responses to these activities to generate and control the quality of multi-level goal structures and subgoal labels. After enough learners' responses are populated, CodeTree algorithmically aggregates learners' partial but complementary subgoals to generate comprehensive multi-level goal structures (Fig. 2.2). Although our main aim is to construct goal structures for code examples, we added the features for collecting subgoal labels to the system to support end-to-end subgoal hierarchy generation.

We evaluated the feasibility of CodeTree for learnersourcing subgoal hierarchies. Our evaluation study with 45 Python novices showed that 1) CodeTree could learnersource correct subgoal hierarchies for three code examples with just five learners, 2) studying code examples with CodeTree resulted in a higher learning gain in code tracing skills (with standardized effect sizes of around 0.7) and higher satisfaction than with explanations of code alone, and 3) the user interfaces and visualizations in CodeTree helped learners understand and generate subgoal hierarchies by enhancing the visibility of the mapping between subgoals and code, and by providing an overview of the subgoal hierarchies.

Our primary contributions are summarized as follows:

- A learnersourcing workflow and algorithm for generating multi-level goal structures of code examples.
- CodeTree, a system that embeds the workflow and provides user interfaces that visualize subgoal hierarchies to scaffold generating and learning of subgoals in code examples.
- Empirical evidence that CodeTree can populate high-quality subgoal hierarchies with just five learners while improving their code tracing skills and learning satisfaction.

2.2 Related Work

This research aims to support the adoption of subgoal learning on code examples at scale. We use learnersourcing as an approach to generate goal structures needed for subgoal learning in learning environments where experts and data are scarce. This section reviews previous literature on 1) subgoal learning and its practices, and 2) learnersourcing systems of different forms.

2.2.1 Subgoal Learning

A subgoal refers to a conceptual action or state that is found in the process of achieving a higher level (sub)goal in problem-solving [21]. Subgoals organized in hierarchies provide a useful mental model of task structures for decomposing tasks into subtasks [7]. Subgoal learning is a pedagogy that teaches learners to discern task structures and constituent subgoals in worked examples so that learners can modify and apply the task structures to novel problems [22].

Early research on subgoal learning focused on finding effective presentations of subgoals in mathematics and physics worked examples. Catrambone showed that subgoal labeling, which adds subgoal labels to worked examples, can scaffold learners' transfer from worked examples to novel problems [21, 7, 22]. Catrambone also tested different variants and found that visual isolation of steps and problem-independent subgoal labels can elicit the transfer even further [8, 23].

Later subgoal learning research primarily took place in the programming domain and looked into the different learning effects in depth. Margulieux and Catrambone found that subgoal-oriented instructional materials in programming improve learners' problem-solving performance [9, 24]. Margulieux et al. applied subgoal learning frameworks to introductory programming courses for a semester and found that subgoal-labeled materials also increase learners' retention of knowledge and courses [25, 26]. They also looked into the usefulness of learners' self-explanations of subgoals during initial problem-solving and observed that self-explanations are as useful as expert-generated labels [27]. Ericson et al. used subgoal labels in Parsons problems to make them more effective for testing learners' performance and understanding of code structure [28].

Researchers also studied subgoal learning in different activity types. Based on Chi's active-constructive-interactive framework of learning [29], the constructive method of subgoal learning, in which learners learn by creating subgoal labels by themselves, has been proposed and investigated. Compared to the passive and active methods that give learners expert-generated subgoal labels, the constructive method can help learners acquire more transferrable knowledge by promoting creative thinking and self-explanation. Extensive research has shown that the learners who practiced the constructive method outperform learners with either passive or active methods for basic programming and app inventing tasks [10, 11, 30].

Our work is founded on the findings in the previous research. Since we want our learnersourcing tasks to be pedagogically meaningful, we followed the constructive and active methods of subgoal learning to design our tasks. Hence, one of our evaluation metrics is how much our system replicates the previous pedagogical effects. By learnersourcing goal structures, we envision that our system will enable the application and investigation of subgoal learning in broader environments.

2.2.2 Active Learnersourcing

Learnersourcing is a type of crowdsourcing that leverages learners' responses in their learning activities to generate meaningful data for future learners [31]. Learnersourcing has advantages over expert-sourcing in terms of scalability because it can draw a workforce from large-scale learning environments, such as MOOCs and Q&A websites, and learners are often motivated to participate in learning activities without monetary rewards. In return, learnersourcing requires reliable quality control mechanisms on learner-generated data because learners are inherently less knowledgeable than experts. For effective quality control, learnersourcing often accompanies majority voting [32, 33] and automated-methods [34].

There are largely two types of learnersourcing—passive and active—depending on how data are generated. Passive learnersourcing uses readily acquirable data from natural learning processes, such as learners' interaction logs and code submissions [35, 36]. Since the data size is often large, automated methods are used to analyze and create meaningful learning supports [16, 37]. When target data is not readily available, active forms of learnersourcing are used. Active learnersourcing adds new learning activities to conventional learning processes to ask learners to generate specific data [38, 39, 40, 41]. One of the challenges in active learnersourcing is to design the activities to be pedagogically meaningful and easy to attempt to encourage the voluntary participation of learners. The activity designs often follow well-defined pedagogies and microtask workflow to improve the learning experience and reduce learners' workload.

There has been research on using active learnersourcing to generate subgoal labels for how-to-videos, mathematics, and algorithmic problem-solving. Crowdy [19] learnersourced subgoal labels for how-to videos by periodically asking learners to self-explain the goals of video sections while watching. Crowdy used a generate-evaluate-proofread workflow to divide the label generation process into manageable microtasks and to ensure the quality of subgoal labels through multiple checks. SolveDeep [13] gathered solution graphs of mathematics problems by asking learners to group the steps in their solutions by subgoals and explain them. Collected solution graphs are then used to generate feedback on subgoals that future learners make. AlgoSolve [12] used active learnersourcing to collect subgoal labels of code for algorithmic problem-solving. AlgoSolve used a vote-label workflow to familiarize learners with high-quality subgoal labels first. Collected subgoal labels are used to scaffold future learners to plan their solutions.

Our work extends the line of research on learnersourcing subgoals. Previous learnersourcing systems have focused on generating subgoal labels for goal structures that experts pre-defined. Although these systems successfully reduce experts' burden to complete subgoal hierarchies, their scalability depends on experts creating goal structures because the creation of goal structures needs to precede subgoal labeling. Our work empowers previous label-sourcing systems to be truly scalable with a learnersourcing workflow and coordination algorithms for making the generation of subgoal hierarchies fully learner-driven.

2.3 Design Goals

Based on prior work, we set three design goals for learnersourcing goal structures and supporting subgoal learning of code examples. Our design goals touch upon the type of subgoal hierarchies to generate, learnersourcing workflow design, and a versatile visualization to help learners generate and learn subgoal hierarchies.

G1. Generate multi-level goal structures.

Theoretically, a code example can have multiple instances of goal structures. Goal structures may vary in the granularity of constituent subgoals and depth. Among the many possible instances, we specifically aim to generate multi-level goal structures that are useful for subgoal labeling [24], self-explanation activities [11, 12], and feedback generation [13]. Having multi-level goal structures is especially useful for making these use cases more adaptive to learners. For example, learners with prior knowledge can receive feedback and questions for high-level subgoals, while less-experienced learners can start with low-level subgoals to understand smaller code patterns. Previous studies also showed that single-leveled subgoals hardly fit all learners with diverse background knowledge [8, 23, 42, 43]. Hence, to maximize the sensitivity of the levels and their benefits, we aim to populate subgoals at as many levels as possible and organize them into multi-level goal structures.

G2. Divide the generation task into microtasks while not compromising the learning objective.

Generating multi-level subgoal hierarchies from scratch is a complex task that may frustrate individual learners. In crowdsourcing literature, dividing a complicated task into manageable microtasks is shown to reduce cognitive demand and improve crowd workers' performance [44, 45]. When designing microtasks in learnersourcing, one of the key considerations is to keep the size of the microtasks small enough so that learners can easily attempt it, but at a level that does not compromise the learning objective [20]. For example, one possible microtask design is to assign learners different parts of code to generate multi-leveled subgoals. However, this may compromise the learning aspect of seeing code examples, as learners will not get enough chances to understand the entire code. Hence, we aim to break the hierarchy generation task into manageable units but ensure that each microtask helps learners skim and understand entire code examples.

G3. Provide learners with a visualization for an overview of subgoal hierarchies.

Subgoal hierarchies are complex data structures that connect code, goal structures, and subgoal labels (Fig. 2.2). Learners are not typically familiar with generating such complex subgoal hierarchies. Previous crowdsourcing research showed that visualizing the overview of worker-generated data improves their performance and efficiency in complex annotations and graph generation [46, 47, 45]. Visualization of subgoal hierarchies during subgoal generation tasks can also alleviate learners' difficulties by raising awareness of the data they generate and possibly improve. The visualization can also aid in learning the organization of the goal structures and how each subgoal instantiates to a specific code. Hence, we aim to add a versatile visualization that can scaffold both generation and learning of subgoal hierarchies to our interface.

2.4 System

We built CodeTree, a learnersourcing system that generates high-quality subgoal hierarchies while supporting subgoal learning of code examples. Learners can use CodeTree to enhance their understanding of existing code examples by either generating subgoals of the code on their own (Fig. 2.3) or selecting the best descriptions for given subgoals (Fig. 2.4). After populating enough subgoals from learners, CodeTree algorithmically aggregates the subgoals into comprehensive subgoal hierarchies. The following subsections describe the user workflow, interfaces, and our algorithm in detail and explain how they achieve the three design goals.

2.4.1 Microtasks for subgoal learning

We divided the hierarchy generation task into two microtasks—*Generation* and *Selection*—taking G2 into account (Fig. 2.1). In the *Generation* task, learners self-explain subgoals of code examples by grouping code lines into functionally meaningful units and describing each unit. In the *Selection* task, learners solve multiple choice questions (MCQs) that ask for selecting the best label for each subgoal. We chose *Generation* and *Selection* tasks as our microtasks because each task follows the constructive and active methods of subgoal learning [29, 11]. Each task is complete on its own in terms of helping learners explore the entire code while dividing learners' workload to generate complete subgoal hierarchies from scratch.

Microtask 1: Generate code groups and subgoal labels

In the *Generation* task, learners self-explain the functions of each part of code by generating subgoals on their own. Learners first read problem statements and code examples to check problem contexts and solutions. Then, learners use the hierarchy generation interface to generate and organize subgoals. Learners can generate subgoals by 1) creating an empty subgoal either at the root or below other subgoals at Fig. 2.3 (C), 2) clicking lines of code at Fig. 2.3 (B) to add to the subgoal as a group, and 3) write a subgoal label that explains the group at Fig. 2.3 (C). Although learners do not receive feedback on their subgoals, the *Generation* task can be helpful as learners explicitly self-explain functions and structures of code [48, 27].

Through the *Generation* task, CodeTree collects diverse code groups (i.e., groupings of code lines) and subgoal labels. Each learner outputs a list of code groups and subgoal labels. We expect the lists to reflect learners' diverse perspectives [42] and contain subgoals at different levels that can serve as the basis for generating multi-leveled hierarchies (G1).

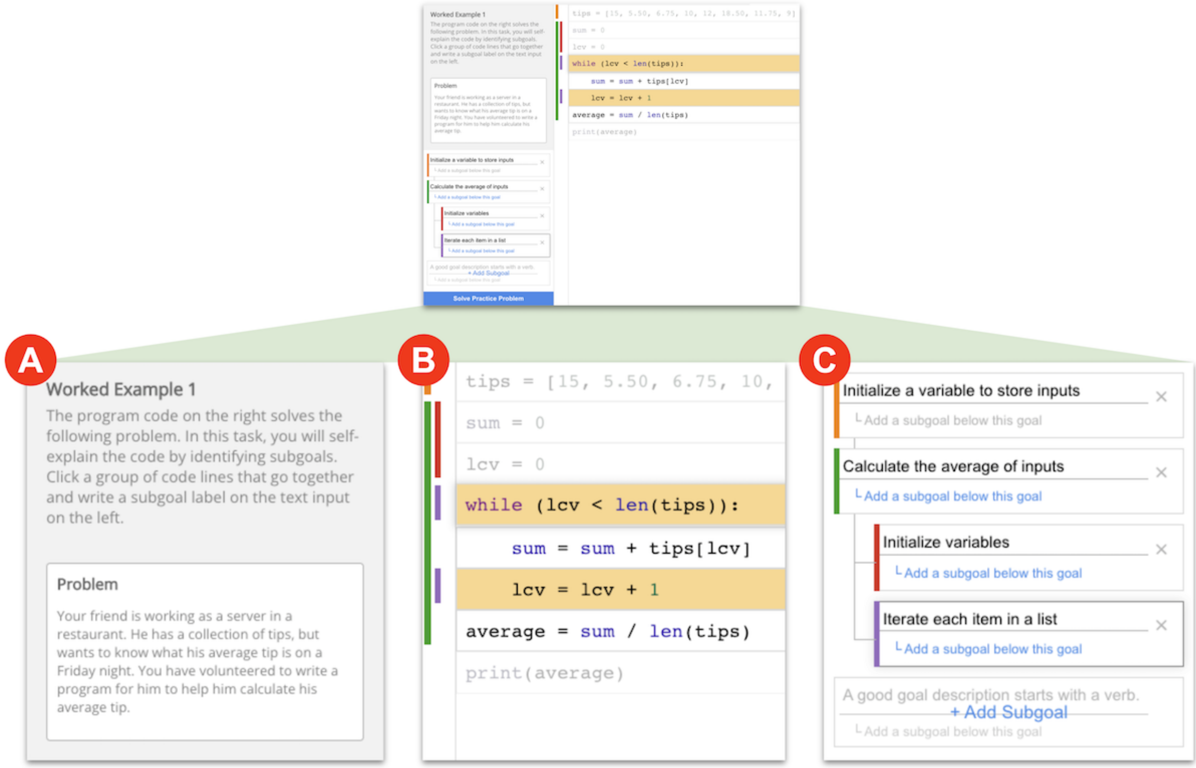

Figure 2.3: The user interface for the generation task: (A) Instructions and problem statement, (B) A code example to study. Learners can click and select lines of code to make a code group (currently selected lines are highlighted in orange). Code lines are dimmed to gray and become unselectable if they are either already grouped or outside of parent subgoal scopes, (C) Hierarchy generation interface. Learners can write down subgoal labels for each code group and can add lower-level subgoals.

Microtask 2: Select subgoal labels that best explain constituent code groups

In the *Selection* task, learners self-explain the function of each code group in a given goal structure by selecting the best descriptions. Similar to the *Generation* task, learners first read problem statements and code examples. Then learners answer a series of (MCQs) given by CodeTree to check their understanding of the code. Each MCQ has at most three options to choose from, and learners can add a new one if none looks plausible or if there is a better description. After solving each question, learners receive corrective feedback on their answers (Fig. 2.4 (C)) to confirm their understanding of the code. Colored bars (Fig. 2.4 (B)) visualize given subgoal hierarchies and highlight positions of code groups asked by MCQs (G3).

Each MCQ has two answers and a distractor (wrong answer). CodeTree generates MCQs based on the learner-generated subgoal labels from the *Generation* task. The answers are chosen from the subgoal labels that previous learners created for the code group being asked. The distractor is also chosen from learner-generated subgoal labels but from another random code group that is mutually exclusive. CodeTree selects distractors based on our heuristic assumption that previous learners would not write interchangeable subgoal labels for two mutually exclusive code groups. We regard choosing two answer options as a multi-armed dueling bandit problem [49] and use a round-robin and greedy algorithm to balance the exploration for good labels and the provision of the best labels known so far. The reward of the problem is defined as whether learners select either of the answer options.

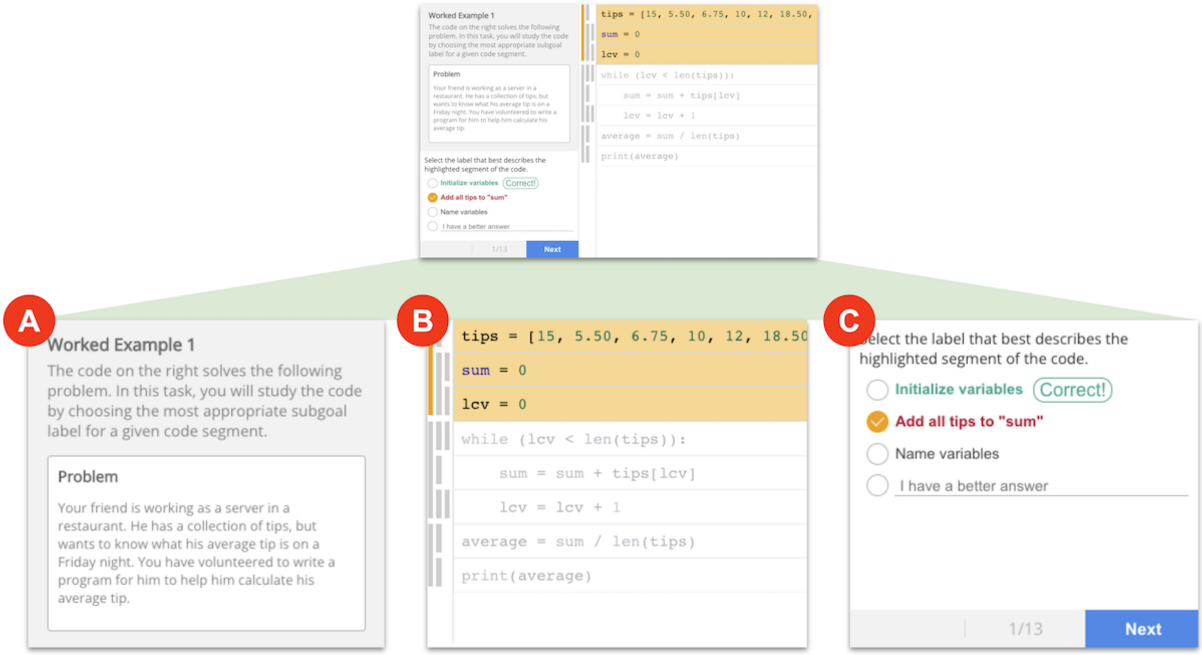

Figure 2.4: The user interface for the selection task: (A) Instructions and problem statement, (B) A code example to study. Parts of code being asked in the MCQ are highlighted in orange, (C) A MCQ problem and its options. When learners select options and click the "Next" button, our system provides corrective feedback on their selection.

2.4.2 Colored bar visualization to overview subgoal hierarchies

Our system also provides a novel visualization of subgoal hierarchies through color-coding of code scope and labels to overview goal structures and adapt flexibly to different code formats and deeply nested structures. We adopted the visualization to the user interfaces of both microtasks (Fig. 2.3 (B) and Fig. 2.4 (B)). Each colored bar beside code examples represents a code group. The vertical position of a bar indicates the code lines that it groups. For example, the red bar in Fig. 2.3 represent a code group for line 2 and 3. The horizontal position of a bar indicates the level it belongs to in a hierarchy. The red bar is at the second level under the green bar that groups line 2 to 8. The color-coding of bars maps each subgoal to a specific part of code examples and reduces the split-attention effect by serving as a visual link between the spatially distant code and subgoal labels [50, 51].

Colored bars collectively outline the goal structures and help learners overview them (G3). In the *Generation* task, colored bars support learners in coordinating overall goal structure by informing learners at which position and level they create subgoals. Learners can also check the bars to easily spot which part of code examples they have not annotated with subgoals. In the *Selection* task, colored bars serve as a navigator to traverse goal structures deeply nested with many constituent subgoals. Learners explore the goal structure by preorder traversal and receive MCQs that ask about increasingly specific code groups. Learners can also refer to parent or child code groups to select the right level of subgoal explanations in MCQs.

The visualization is designed to present complex mapping between subgoal hierarchies and code. Code examples that are subgoal-labeled at their creation are written in a way that clearly shows goal structures from code. Inline comments or visual isolation can easily present subgoal hierarchies of such code examples. However, in order to make our visualization applicable to more general code examples beyond the pre-formatted code examples, we made the visualization flexible to work for possibly complex structures and mappings. For instance, lines 4 and 6 in Fig. 2.3 may form a meaningful code group despite not being contiguous. While the comment-based presentation does not work for these non-contiguous code groups, our visualization can pinpoint code lines and group them.

Algorithm 1 A hierarchy generation algorithm

Input: I: A list of Tuple(code group, subgoal label)
Step 1: merge tuples with identical code groups
$U \leftarrow$ an empty list
for each Tuple(code group G, subgoal label L) in I **do**
  if U has a tuple containing G **do**
    add L to the existing tuple
  else do
    add Tuple(G, L) to U
  end if

Step 2: calculate priority of each code group
for each code group G in U **do**
  $Priority_G \leftarrow$ occurrence number of G in I

Step 3: sort code groups by their priority
sort U by $Priority$

Step 4: populate as many code groups in a hierarchy
$H \leftarrow$ an empty hierarchy
for each Tuple(code group G, subgoal labels L) in U **do**
  if G does not conflict with H **do**
    add G and L to H
  end if
Output: H

2.4.3 Workflow and algorithm for generating multi-leveled hierarchies

Our learnersourcing workflow is organized by the two microtasks and a hierarchy generation algorithm (Fig. 2.1). Learner-generated subgoals from *Generation* are the seed for our algorithm to generate initial subgoal hierarchies. The generated hierarchies are fed to *Selection* to refine subgoal labels.

The hierarchy generation algorithm (Algorithm 1) is based on two assumptions:

A1. Most learners can identify correct individual subgoals although they may lack the ability to identify an entire hierarchy.

A2. Learners can recognize complementary levels of subgoals so that their collection will be comprehensive enough to make complete subgoal hierarchies.

Based on these assumptions, the algorithm first calculates the priority of each code group by their submission count. Then, the algorithm uses the priority values to decide which subgoal to add to the subgoal hierarchy in case of conflicts (see Fig. 2.7). The algorithm keeps adding subgoals without conflicts until it achieves complete hierarchies.

Before being fed to the *Selection* task, generated subgoal hierarchies undergo post-processing. CodeTree removes the poor subgoal labels that are too lengthy, that start with non-verbs, or that are textually similar to other subgoal labels. For measuring textual similarity, we divided texts into morphemes [52] and computed the Sørensen–Dice coefficient [53] between the morphemes. To make subgoal hierarchies complete, CodeTree also adds code groups to the leaf positions where subgoals are missing from the input. In the *Selection* task, learners generate the labels for these new code groups.

2.5 Study Design

To evaluate the feasibility of CodeTree for the three design goals, we recruited 45 programming novices to run a between-subjects study with three conditions—**Baseline**, **Generate**, and **Select**. The conditions differed in the methods of studying code examples. *Generate* condition participants studied code examples by doing the *Generation* task; *Select* condition participants studied code examples through the *Selection* task; *Baseline* condition participants studied code examples with detailed explanations only. Through this study, we explored three research questions that link to each design goal.

RQ1. Can CodeTree learnersource correct and comprehensive subgoal hierarchies?

RQ2. Do learners find *Generation* and *Selection* tasks helpful for learning and manageable to do?

RQ3. Does the colored bar visualization help understand and generate subgoal hierarchies?

2.5.1 Participants

The target users of CodeTree are programming learners who have learned basic Python syntax but struggle to write code themselves to solve problems. We recruited 45 participants who 1) took an introductory Python programming class only, 2) did not score perfectly in our pre-test, and 3) experienced moderate intrinsic cognitive (below 19 out of 30) during the study. The participants were recruited on campus and from online communities with a compensation of 20,000 KRW (approximately 17 US dollars) for a 90-minute session. The participants were randomly assigned to one of the three conditions (*Baseline*, *Generate*, and *Select*). We confirmed that there was no statistically significant difference in participants' initial knowledge (pre-test scores) between conditions (one-way ANOVA, F=0.89, and p=0.42).

Table 2.1: Demographic averages for 45 participants and the correlation of each characteristic with participants' performance score.

	Mean/ proportion	***Std. deviation***	***Pearson's correlation with performance score***	
			r	***p***
Gender	19 female	-	0.14	0.30
Age	21.73	3.84	-0.29	0.15
Year in college	2.82	3.84	0.31	0.13
Comfort with programming (1:Not comfortable at all - 7: Very comfortable)	3.69	1.13	0.01	0.48
Expected difficulty for learning programming (1: Very difficult - 7: Very easy)	3.44	1.02	0.33	0.11

Table 2.2: The outline of the study and the time allotted to each step.

Step (min.)	**Baseline (15 participants)**	**Generate (15 participants)**	**Select (15 participants)**
1 (8)	Introduction + Informed consent		
2 (3)	Demographic questionnaire		
3 (5)	Pre test		
4 (10)	Analogy training	Subgoal training	
5 (30)	Study 3 code examples with explanations + Practice 3 problems	Generation tasks on 3 code examples + Practice 3 problems	Selection tasks on 3 code examples + Practice 3 problems
6 (3)	Cognitive load measurement		
7 (20)	4 Assessment problems		
8 (5)	1 Parsons problem		
9 (5)	Post-test (identical to pre-test)		
10 (-)	Post survey		

2.5.2 Procedure and Materials

Throughout the sessions, participants learned the usage of while-loops in Python through three code examples and practice problems isomorphic to the examples. We referred to the study procedures and materials from previous studies on subgoal learning [10, 11]. The code examples, the problems for pre- & post-tests, practice, and assessment were identical to the materials used in the between-subjects study of Margulieux et al. [11], except that the materials were rewritten in Python. We chose Python for our programming language because Python was the most popular among our recruitment targets. All participants had access to review materials that briefly explained basic syntax and concepts of Python. The instructions and the user interface were localized into Korean to avoid confusion or unnecessary difficulties.

The example learning steps (steps 4 and 5 in Table 2.2) were different by conditions. Participants in *Generate* and *Select* conditions received tutorials about subgoal learning and usage of each feature of our user interface. The tutorial included the learning benefits of subgoal learning, exemplar subgoal labels on simple math equation solving, and subgoal-making exercises with answer labels at the end as corrective feedback (Fig. 2.5 Right). The example subgoal labels were all problem-independent, implicitly guiding participants to write problem-independent subgoal labels. *Baseline* participants received analogy training, which exerts cognitive load comparable to the subgoal training in other conditions [10]. *Generate* and *Select* participants used respective interfaces in Fig. 2.3 and Fig. 2.4 to study code examples. *Baseline* participants used another interface that removed subgoal-related features (Fig. 2.5 Left).

The code examples in all three conditions were presented with line-level explanations of the code. We added the explanations to help participants understand the code and to simulate typical Q&A websites and documentation where explanations of code are present. We used the latest Codex AI model [54] to generate the explanations, considering that it is the most readily available method for providing detailed explanations of code at scale [55]. The first author checked the quality of the explanations.

The transfer distances of the practice problems and assessment problems to code examples were set differently. Right after studying each code example, participants solved a practice problem, which solution was isomorphic [10] to the code example. Participants could run their code and receive feedback on whether they were correct. We chose isomorphic problems so that participants could try out the same code structure they had just studied. For the assessment problems, we chose contextual transfer [10] problems to test how each condition affects participants' performance in modifying learned code examples and transferring them to novel problems. Participants could not run their code during the assessment nor receive feedback.

We conducted all sessions with *Generate* participants before any *Select* participants to populate subgoal hierarchies for the *Selection* task. The hierarchy algorithm generated subgoal hierarchies with the *Generate* participants' code groups.

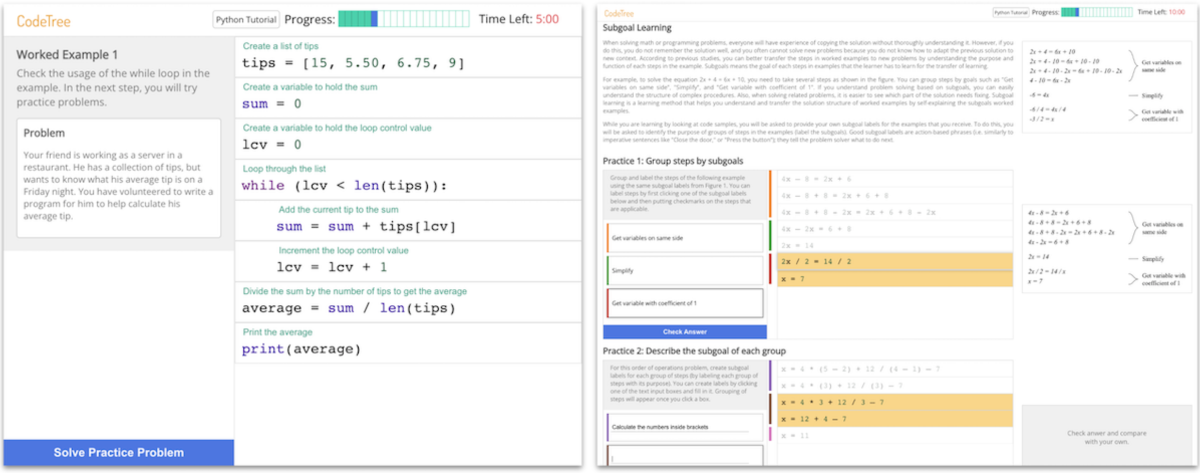

Figure 2.5: Left: the user interface for *Baseline* condition. Participants studied the worked example with line-level explanations of the code only, and they could proceed to the next step at any time without requisites. Right: the instruction and practice activities for the subgoal training. Participants grouped and subgoal-labeled math equation-solving steps as practice and then checked the answer.

2.5.3 Measurements

Our measurements are two-fold. The evaluation of the quality and variety of the subgoals created by participants was taken after the study with external evaluators. The assessment of participants' learning gain and experience was conducted sequentially during the study (Table 2.2).

Code group quality. The quality of code groups made by *Generate* participants was classified into three types—*incorrect*, *meaningful*, and *core* (see Table. 2.3). A code group is *meaningful* if its constituent code lines collectively represent any useful subgoal. A code group is *core* if it is *meaningful* and represents one of the subgoals essential to solving a problem. Code groups that are not *meaningful* nor *core* are considered *incorrect*. We recruited two evaluators with four semesters of TA experience in CS courses. The evaluators assessed the first 30 code groups together and the remaining 39 code groups independently. The inter-rater reliability for the independent assessment was substantial (Cohen's kappa, κ=0.63).

Subgoal label quality. The quality of subgoal labels made by *Generate* participants was classified into three types—*incorrect*, *problem-specific*, and *problem-independent* [11] (see Table. 2.4). A label is *incorrect* if it simply describes the execution of code or is wrong. A label is *problem-specific* if it correctly describes the function of code but contains information specific to the current problem and cannot be generalized to other isomorphic problems. A label is *problem-independent* if it is correct and generalizable to other isomorphic problems. We recruited two other evaluators with two semesters of TA experience in CS courses. Likewise, they assessed the labels from the first 30 code groups together and the remaining 72 labels independently. The inter-rater reliability for the independent assessment was high (Cohen's kappa, κ=0.68).

Table 2.3: The code groups that the evaluators assessed as *incorrect*, *meaningful*, and *core* respectively. The entire code example is in Fig. 2.2.

Code group quality	**Example**
Incorrect	L1: tips = [15, 5.50, 6.75, 10, 12, 18.50, 11.75, 9] L2: sum = 0 L5: sum = sum + tips[lcv]
Meaningful	L6: lcv = lcv + 1
	L4: while (lcv <len(tips)):
Core	L4: while (lcv <len(tips)): L6: lcv = lcv + 1
	L7: average = sum / len(tips) L8: print(average)

Table 2.4: Subgoal labels that evaluators assessed as *incorrect*, *problem-specific*, and *problem-independent*. The labels described the subgoal for code "sum = sum + tips[lcv]" in Fig. 2.2.

Subgoal label quality	**Example**
Incorrect	Add tuple[lcv] to sum
Problem-specific	Add a tip value to total sum
Problem-independent	Add a value to get the total sum

Diversity index and conflict ratio of code groups. One of our assumptions (A2) in algorithm design is that learners will generate diverse code groups that can complement each other. Hence, diversity and complementarity in code groups are important properties that make our learnersourcing workflow effective. We used Simpson's diversity index [56] to measure the diversity of code groups. For the calculation of the index, we treated each code group as an entity and the number of its submissions by participants as its population. We measured the complementarity of code groups by the ratio of conflicting code group pairs in all possible pairs (1.0 is a total conflict). Two code groups are complementary if one is the subset of the other or there are no intersecting code lines; otherwise, they conflict and cannot coexist in the same hierarchy (see Fig. 2.7). However, there will be existential conflicts between code groups because there can be multiple correct instances of subgoal hierarchies. We measured the existential conflict ratio by the same calculation but with only core code groups.

Holistic evaluation on subgoal hierarchies. Holistic evaluation is meaningful apart from the previous measurements. A hierarchy may not be effective for learning, even though its constituent code groups and labels are correct individually. For example, a subgoal hierarchy may have an imbalanced goal structure or inconsistent subgoal labels for denoting parts of code. The holistic evaluation aims to evaluate code groups and labels as a whole beyond their individual qualities. The assessment focused on 1) the composition of code groups in each layer in hierarchies and 2) the consistency among subgoal labels (see Fig. 2.6). We assessed the composition by evaluating whether each parent code group is split in a logically even manner by its child code groups. We quantified consistency by the size of the largest set of labels that do not conflict with each other in denoting variables or concepts. The evaluators who assessed the code groups worked together to evaluate the composition and consistency of the three subgoal hierarchies generated for each code example.

Score increase. Participants' code-tracing skills were measured with the multiple-choice questions in pre and post-tests (Step 3 and 9 in Table 2.2). Pre and post-tests were composed of questions that asked about execution outputs of while loops or the code to print desired outcomes. The questions for both tests were identical, but the order of questions and options was randomized. The score differences between pre and post-tests were calculated for each participant to measure individuals' learning gain in code-tracing skills.

Performance score. Participants' code-writing skills were measured with four assessment problems and a Parsons problem (Step 7 in Table 2.2). Two external evaluators graded the participants' answers to the assessment problems. The evaluators had two and four semesters of TA experience in an introductory Python programming class. The evaluators followed the grading guidelines from the previous work [11], and the total score for the four problems was 36. For each assessment problem, the two evaluators graded the first 20 participants' answers together to make a specific grading scheme. They graded the remaining 25 answers independently. The inter-rater reliability of the independent grading was high (Pearson's correlation, r=0.94). Participants' answers to the Parsons problem were auto-graded and scored out of 10. Each participant's performance score is the sum of the assessment score and the Parsons problem score.

Cognitive load. The cognitive load of participants was measured right after the example learning steps to evaluate how each intervention imposes a burden on learners (Step 6 in Table 2.2). We used ten 10-point Likert-scale questions designed to measure cognitive load for programming tasks [57]. The questions asked about three types of cognitive load—intrinsic, extrinsic, and germane. Participants' ratings for each question were summed up by the types, resulting in a maximum of 30 for intrinsic and extrinsic, and 40 for germane.

Post-survey questions. After the post-test, the participants received survey questions on their learning experience and system usability (Step 10 in Table 2.2). The survey is composed of two parts. The first part (Fig. 2.10) had questions about their experience studying code examples and were common to all conditions. The second part (Fig. 2.11) had questions about the system's usability and helpfulness for conducting condition-specific tasks and were different by condition. Each question had two sub-questions in which participants rate a 7-point Likert scale for a given statement and leave text comments to explain their rating.

2.6 Results

We report the quantitative results and participants' comments on the evaluation study and answer the three research questions. We organized this section by each research question.

RQ1. Can CodeTree learnersource correct and comprehensive subgoal hierarchies?

Learnersourced subgoal hierarchies were correct and complete. The ratios of correct (*meaningful* + *core*) code groups in total code groups in respective subgoal hierarchies were 13/13, 13/14, and 20/21. Moreover, the subgoal hierarchies contained all the *core* code groups identified by the evaluators. Noting that the average ratios of *core* code groups in each participant's submission were 65%, this result shows that the participants could collectively populate most of the core code groups, although a participant alone could not. In terms of label quality, the ratios of correct (*problem-specific* + *problem-independent*) subgoal labels in total labels were 12/13, 11/14, and 18/21 in each subgoal hierarchy. Among them, 8, 8, and 10 subgoal labels were *problem-independent*. In the holistic evaluation, most compositions of the code groups were correct, and the consistencies between subgoal labels were also high. The ratios of correct parent-child code group relations were 6/6, 5/7, and 8/10. The sizes of the largest consistent subgoal label set were 11/13, 12/14, and 21/21. Hence, we conclude that each generated subgoal hierarchy is correct and consistent as a whole. Detailed evaluation results of subgoal hierarchies for the first and second code examples are presented in Fig. 2.6.

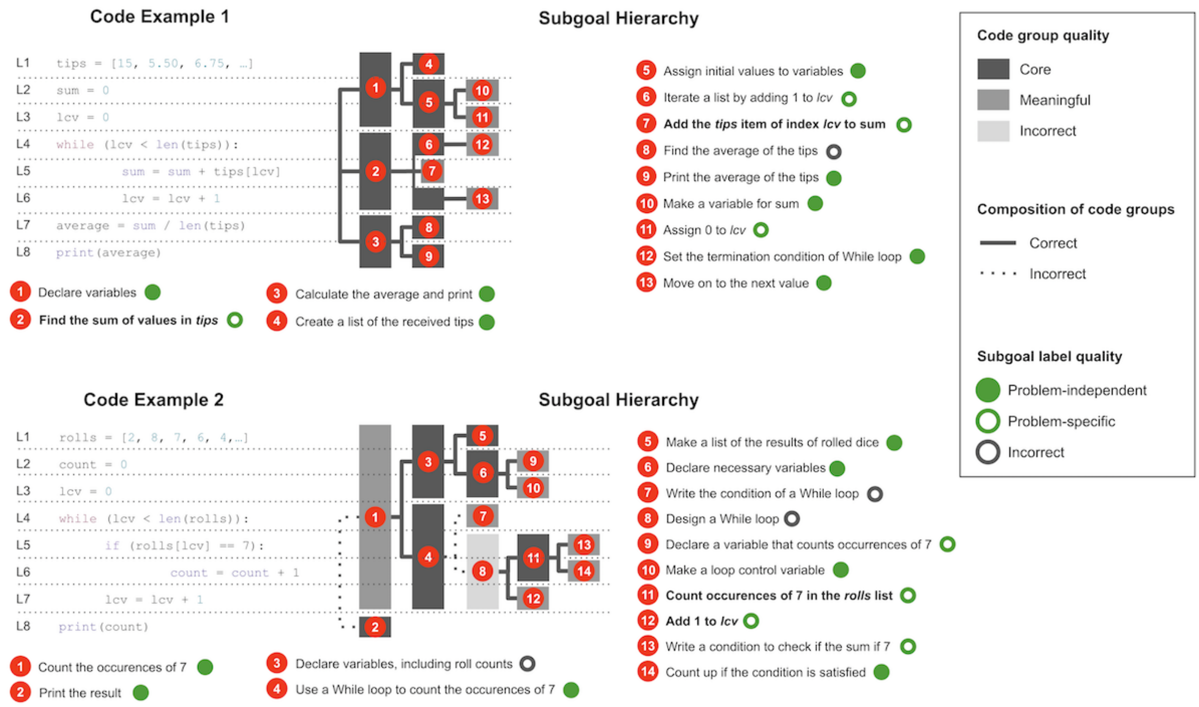

Figure 2.6: The subgoal hierarchy generated for code examples 1 and 2 and their evaluation results. We presented the most selected subgoal labels only. The participants wrote subgoal labels in Korean to avoid language barriers in describing good labels, and we translated them into the figure. The labels that are inconsistent with others are bolded. The evaluators judged these labels to be inconsistent in that they use exact variable names (e.g., rolls and lcv) for reference while others explain in words.

The participants generated meaningful code groups and labels. Almost all of the code groups that *Generate* participants made were correct (*meaningful* + *core*). On average, 95% of the code groups from a participant were correct ($SD = 6\%$). On the other hand, the ratios for *core* code groups were only 65% ($SD = 17\%$). For MCQ responses, the participants chose better or equal quality labels most of the time ($M = 98\%$, $SD = 3\%$). These observations collectively verify our assumption A1 to a certain extent that the majority of individual learners can identify correct subgoals but struggle to generate complete hierarchies with all *core* code groups.

The participants generated diverse but somewhat conflicting code groups. The Simpson's diversity indexes of code groups for the three code examples were 0.93, 0.91, and 0.96. Their conflict ratios were 0.54, 0.54, and 0.55, and their existential conflict ratios were 0.53, 0.57, and 0.57. High diversity indexes and moderate conflict ratios indicate that participants recognized diverse subgoals but from different instances of subgoal hierarchies. Although this does not align with our assumption A2 that learners will make complementary subgoals, the quality evaluation showed that the algorithm filtered conflicting code groups and generated hierarchies correctly. We expect that there is room for guiding learners to generate subgoals of specific instances to reduce conflicts and make our workflow more efficient.

Just a few learner contributions could help generate high-quality subgoal hierarchies. To estimate the number of learners needed for achieving high-quality subgoal hierarchies, we measured the quality of code groups and subgoal labels under a simulation. We simulated 1) the hierarchy generation algorithm and 2) the *Selection* task, each with n number of simulated learners.

To simulate the algorithm with n participants, we randomly sampled n *Generate* participants. We then used the algorithm to generate subgoal hierarchies from the sampled participants' code groups. The

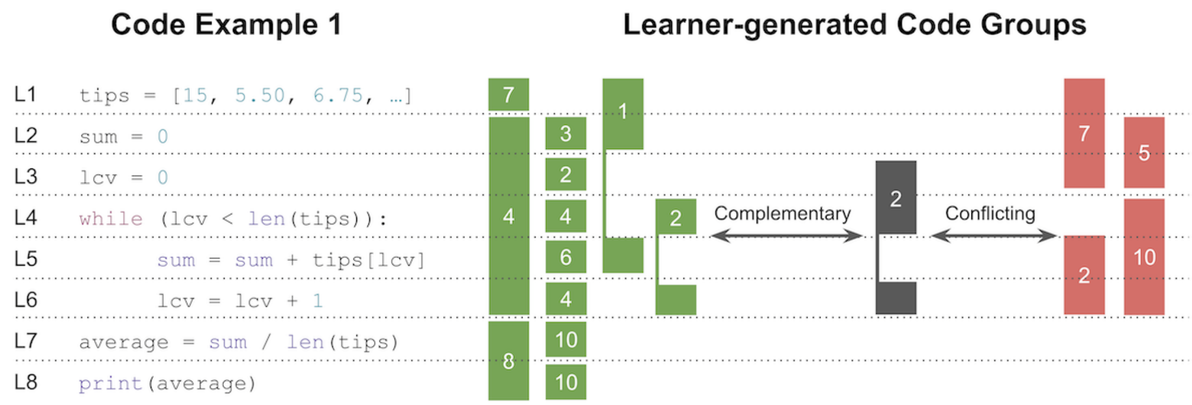

Figure 2.7: The code groups that the *Generate* participants generated for code example 1. The number in a code group indicates the number of participants who submitted it. To measure conflicts between code groups, we counted the ratio of conflict relations (colored in red) in all pairs of code groups.

random sampling was repeated 15 times for each n. We report their average. We calculated the ratio of correct (*meaningful* + *core*) code groups and the number of *core* code groups in the generated hierarchies and plotted them against the number of simulated learners in Fig. 2.8 (A) and (B). The ratios of *correct* code groups at the end (n=15) were 0.93, 0.87, and 0.95 for the subgoal hierarchies of each code example. The numbers of *core* code groups at the end were 6.33, 5.00, and 10.00. Both estimates saturate around n=5, albeit the third subgoal hierarchy continued to improve and populated all the *core* code groups at n=15.

To observe how the quality of subgoal labels changes throughout the *Selection* task, we simulated n successive learner's responses in the *Selection* task with a probability that a participant will choose a better subgoal label in an MCQ. We set the probability as 0.98 based on our empirical result. A simulated learner votes for one of the best subgoal labels in given MCQ options or other options according to the probability. For each n, we repeated the simulation 15 times. We report the average ratio of correct (*problem-specific* + *problem-independent*) and *problem-independent* labels among the labels that received the most votes. Fig. 2.8 (C) and (D) plot these against the number of simulated learners. All hierarchies achieved high correctness at the end (1.00, 0.93, and 0.90) and reached the maximum achievable with the given data. All hierarchies also had high populations of *problem-independent* labels (0.82, 0.77, and 0.66) close to their achievable maximums (0.92, 0.79, and 0.71). Both estimates saturated around n=5.

The result implies that just five learners can successfully populate a goal structure and subgoal labels for a code example. Such learner-to-code example ratio (five to one) shows that the generation of subgoal hierarchies for all code examples in typical Q&A websites and MOOCs is feasible with just existing learners in the learning environments. For instance, a question on StackOverflow is viewed by 30 people on average in a month [2]. If five of them are motivated to understand code examples in depth and contribute subgoals to CodeTree, the system will be able to populate subgoal hierarchies for all newly created and old code examples on StackOverflow. However, we also clearly note that our study participants may have a higher level of prior knowledge in programming than typical learners in the environments, and we may not replicate such high efficiency (i.e., five to one) in the wild where the level of code complexity and knowledge of learners vary a lot. Nevertheless, our finding suggests that learnersourcing is a scalable approach for collecting subgoals online. We discuss how we can concretize this finding in future research in Section 8.

[2]https://data.stackexchange.com/stackoverflow/query/213319/average-views-per-question-by-month

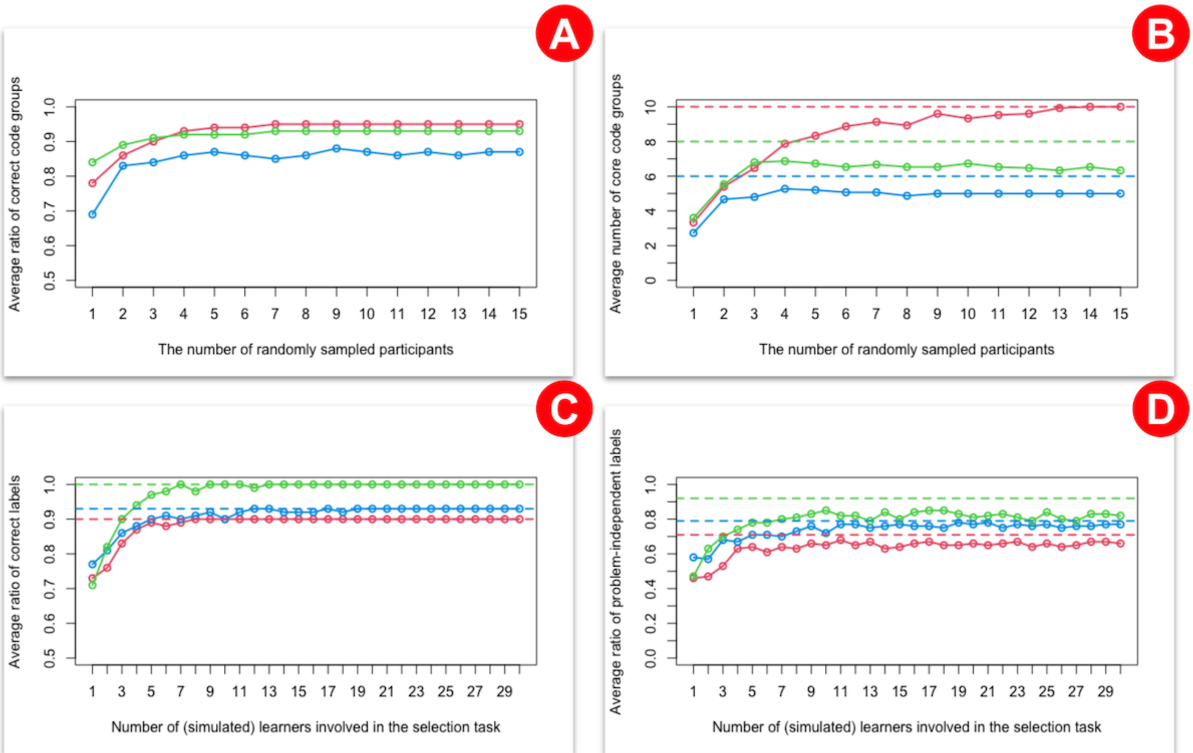

Figure 2.8: Each color line denotes subgoal hierarchies for different code examples: green for code example 1, blue for code example 2, and red for code example 3. (A, B) The average ratios of (correct/core) code groups in hierarchies that were generated with the code groups of randomly sampled n participants. The dashed line denotes the total number of core code groups. (C, D) The average ratio of *correct/problem-independent* labels at the end of the selection task simulation with n participants. The dashed lines denote the maximum achievable ratios for given datasets. For example, if the subgoal labels for a code group are all *incorrect*, there is no way to improve.

RQ2. Do learners find *Generation* and *Selection* tasks helpful for learning and manageable to do?

The generation task improved code tracing skills. All except one participant scored equal or higher in their post-test. The score increases (Fig. 2.9 (A)) in *Generate* condition were statistically significantly higher than in *Baseline* condition (one-tailed t-test, $p = 0.03$, $d = 0.72$). The score increases in *Select* condition were higher than in *Baseline* but not statistically significant (one-tailed t-test, $p = 0.15$, $d = 0.38$). The performance scores (see Fig. 2.9 (B)) in both *Generate* and *Select* conditions were higher than in *Baseline* condition but not statistically significant (one-tailed t-test, $p = 0.15$, $d = 0.38$ between *Baseline* and *Generate*, $p = 0.07$, $d = 0.40$ between *Baseline* and *Select*).

Our results and previous studies on subgoal learning [11, 10] complement each other to a certain extent. The score increases and performance scores measure participants' code tracing and writing skills respectively. Previous studies observed significant improvement in code-writing skills for the active and constructive forms of subgoal learning but weak significance in code-tracing skills. On the other hand, we observed significant improvements in participants' code-tracing skills only. In theory, both code-tracing and writing skills should have improved because code-tracing is a precursor to code writing [58]. We speculate that the difference in the tools for subgoal generation might have elicited different aspects of learning more prominent. Nevertheless, they accord closely with the positive effect of subgoal learning in improving transfer distance.

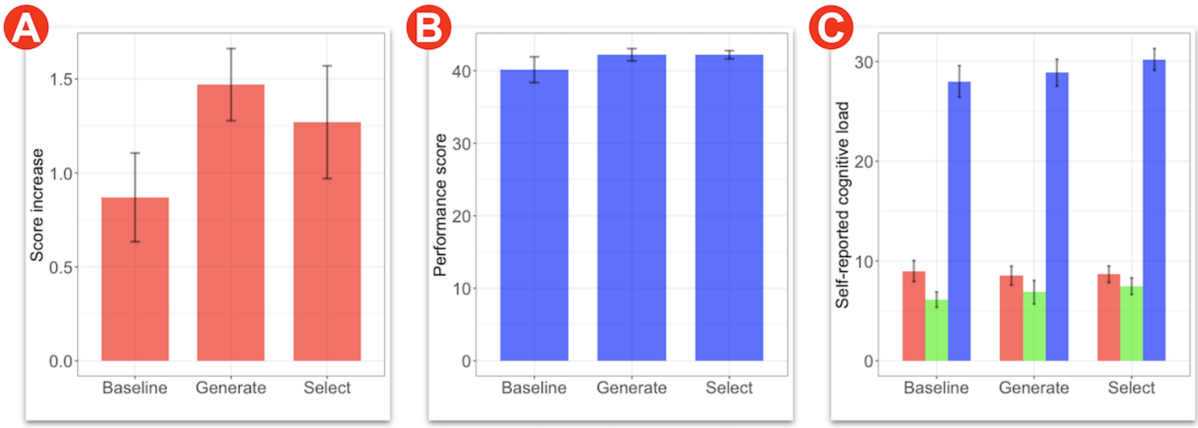

Figure 2.9: (A) Average score increases between pre and post-test scores across conditions, (B) Average performance scores across conditions, (C) Average self-reported ratings of intrinsic (red), extrinsic (green), germane (blue) cognitive load across conditions.

Generation and Selection tasks were manageable microtasks. There were no significant differences between conditions for the three types of cognitive load (one-way ANOVA, intrinsic: $F = 0.07$, $p = 0.94$, extrinsic: $F = 0.68$, $p = 0.61$, germane: $F = 0.50$, $p = 0.52$). Little differences among the conditions indicate that our microtasks did not impose additional cognitive load despite being more active than *Baseline*. P26 commented that subgoal learning tasks were manageable because top-down exploration of subgoals helped him digest the code easily: "I could understand long code examples faster by dividing them into smaller units."

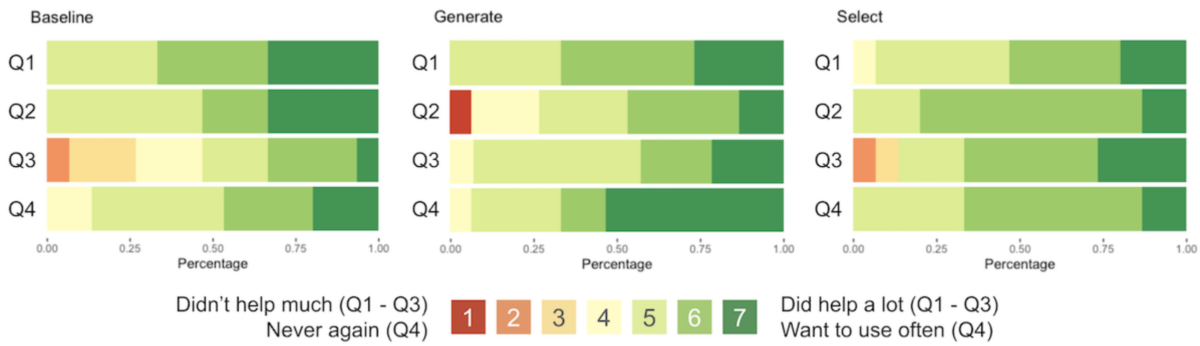

Figure 2.10: Likert scale responses for the post-survey questions regarding learning experience. Note that words in parenthesis were changed depending on the answerers' conditions. The survey questions were: *Q1.* How much did (seeing code and explanations/subgoal learning tasks) help to understand the usage of while loops? / *Q2.* How much did (seeing code and explanations/subgoal learning tasks) help to understand code examples? / *Q3.* How much did (seeing code and explanations/subgoal learning tasks) help to understand hierarchical structures of code examples? / *Q4.* How often do you want to use (code examples/subgoal learning tasks) in future programming learning?

Generation and Selection tasks helped improve learners' satisfaction. The learning condition did not affect participants' *Q1: understanding the usage of while loops* and *Q2: comprehension of code examples* (one-way ANOVA, $F = 0.65$, $p = 0.52$ for *Q1*, $F = 2.56$, $p = 0.09$ for *Q2*). However, the *Generate* and *Select* participants perceived that they *Q3: understood the hierarchical structure of code examples* significantly better than the *Baseline* participants (one-tailed t-test, $p = 0.02$, $d = 0.79$ between *Baseline* and *Generate*, $p = 0.04$, $d = 0.69$ between *Baseline* and *Select*). Subgoal learning tasks were preferred for *Q4: future reuse* over the *Baseline* task, but not statistically significantly (one-tailed t-test, $p = 0.06$, $d = 0.58$ between *Baseline* and *Generate*, $p = 0.31$, $d = 0.31$ between *Baseline* and *Select*).

The participants in *Generate* condition liked the process of explicitly identifying the functions of each part of the code even without corrective feedback. P24 commented, "Subgoal learning would be helpful to learn not only the solution specific to example problems but also the general strategies for solving other problems." Another participant P30 said, "[the *Generation* task] greatly helped to organize code into small pieces. I used to think programming was difficult, but I could gain confidence by doing the task and solving problems." We speculate that the statistical insignificance resulted from the relatively high satisfaction of *Baseline* participants. *Baseline* participants liked the simplicity of the interface and felt more familiar to use. P3 in the *Baseline* condition noted "the UI was so simple that I could inspect and understand code example well." We may observe a more significant preference between conditions if the study is designed to be within-subjects or more longitudinal to reduce the effect of their initial burden to familiarize novel interfaces.

The participants also perceived the corrective feedback in the *Selection* task as correct and helpful for a check (see Fig. 2.11 *Q4* and *Q5*). More than half of the *Select* participants rated over 5 for *Q4* and *Q5*. P40 said that the feedback was useful to confirm his understanding of code examples: "I was unsure of my answers many times, but the correct signs helped me confirm that I was doing right." P33 commented the immediacy of feedback also helped: "[the corrective feedback] trained me to self-explain the code with more general and purposeful terms, rather than simply explaining the execution of the code." P44 pointed out that the feedback would be more effective with supplementary explanations for answers. Although most participants thought the answers given by CodeTree were reasonable, some participants doubted the accuracy of the feedback, especially when their responses all turned out to be correct.

RQ3. Does the colored bar visualization help understanding and generation of subgoal hierarchies?

The bar visualization helped the generation of subgoals. The *Generate* participants were asked two questions regarding the hierarchy generation interface and the colored bar visualization (see Fig. 2.11, *Q1* and *Q2*). Both questions were rated over 5 by more than half of the participants. P25 thought the hierarchy generation interface was intuitive, easy to map between code and subgoals, and effective for representing complex goal structures: "The interface was easy to use and grasp [what I was creating] because subgoals were indented like code in a familiar style." P24 commented, "The interface helped to understand code systematically. I first defined strategies [(i.e., subgoals)] of problems, and then I selected corresponding parts in the actual code." P20, P25, and P29 understood visual notations correctly and liked the idea of using bar lengths and positions to present complex goal structures. P20 noted "the lower-level bars were drawn to the right of, and only within, the upper-level ones. It was intuitive and pleasant to look at with colors."

The bar visualization in the Selection task is unnoticeable but is helpful once noticed. The *Select* participants had mixed perceptions of the helpfulness of the bar visualization (see Fig. 2.11, *Q3*). Half of the participants commented that they did not notice the visualization because its visual changes were too subtle. P31 said "I did not notice the visualization until answering this question. However, if I had looked at it carefully, it would have been helpful to understand the overall hierarchical structures." For those who noticed the visualization, it helped to overview entire hierarchies and check the hierarchical relations between the code asked by MCQs. P41 commented, "I especially liked the area highlighted in orange and the bar on the left showing the structure and scope. I think it helped a lot to understand the overall structure visually. It was also good to show larger scopes of goals first and move to more detailed goals. It would have been better if the bars also showed chosen subgoal labels."

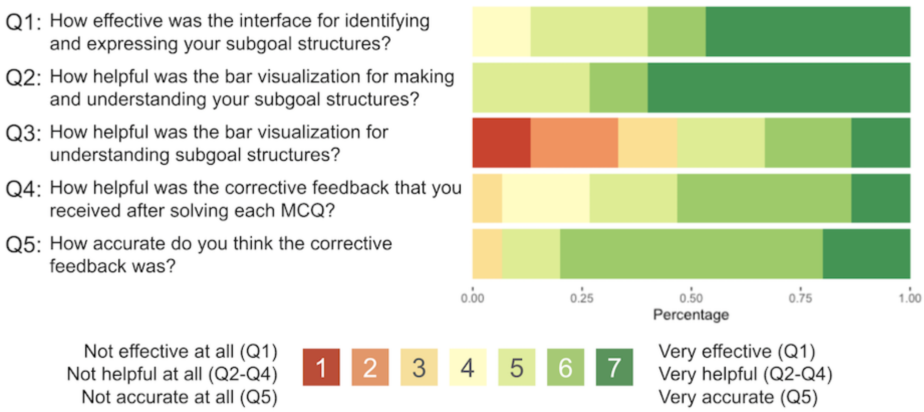

Figure 2.11: Likert scale responses for the post-survey questions regarding user interface and the colored bar visualization, and system-generated feedback. *Q1-Q3* were asked to the *Generate* participants, and *Q4-Q5* were asked to the *Select* participants. Images of system features were attached to each question in the survey to indicate each feature clearly.

2.7 Discussion

In the following subsections, we discuss the implication of our results for generating subgoal hierarchies at scale and generalizing our system design to other relevant domains.

2.7.1 Efficiency of our workflow for generating hierarchies

One of our notable findings is that just a few learners are needed to generate high-quality goal structures. CodeTree generated correct goal structures with just five programming novices. The participants in the study spent about 9 minutes completing each *Generation* and *Selection* task (6 minutes for *Baseline* task). Considering that the *Generation* tasks can run in parallel as they are independent of each other, CodeTree can generate a multi-level goal structure of a code example within 10 minutes in an ideal case. Previous crowdsourcing systems [59, 60] required hundreds of human intelligence tasks (HITs) for generating taxonomy hierarchies of size 10-20 nodes, a size comparable to our goal structures. Compared with these hierarchy generation methods, our learnersourcing workflow can be deployed in even small classrooms.

We argue that learnersourcing may shed light on reducing the number of workers and time needed for crowdsourcing hierarchically structured data (e.g., concept maps and evaluation criteria). Previous systems that used paid crowdworkers (e.g., MTurk) focused on making each task small so that crowdworkers could contribute without knowing a global hierarchy. However, using fragmented microtasks increased the number of total HITs. In our study, we observed that learners tend to have a good understanding of global hierarchies. For instance, more than half of the *Generate* participants submitted multi-leveled subgoals. Because learners have a good sense of global hierarchies, crowdsourcing systems may empower learners to engage more in the global process of hierarchy generation. Learners may be empowered to directly edit and fix global hierarchies to complement algorithmic coordination.

Indeed, we observed the benefit of empowering learners for higher engagement in our system design iterations. Early *Generation* interface constrained learners from making multi-level subgoals (i.e. adding a subgoal below another subgoal). We designed the system this way because we thought allowing the creation of multi-level subgoals would increase task complexity, and learners would not want to exert additional effort on making more subgoals. However, learners in pilot studies commented that making multi-level subgoals would lower the task complexity as they could make goal structures more flexible and closer to what they picture. After redesigning the *Generation* task to accept multi-level subgoals, we could observe higher learner satisfaction and less confusion for making subgoals. The change also improved our system by collecting more subgoals at different levels with fewer learners. These observations align with previous study [61] in that providing learners a choice to engage in tasks rather than forcing them can improve motivation and the quality of collective output. Although this needs a more thorough investigation in real class settings, it will be worth designing future learnersourcing systems with more flexibility and room for higher learner engagement so that learners can contribute more and better if they want to.

2.7.2 Generalization of the workflow and interfaces

Hierarchical summaries of videos and articles are often effective for navigation, overviewing, and learning the contents [62, 45, 63, 64]. These hierarchical summaries are conceptually similar to subgoal hierarchies, as they divide contents into meaningful units and have labels that summarize each unit. Ideally, content creators can provide hierarchical summaries at their creation time, but there are millions of content already existing on the web without such summaries. The generation of hierarchical summaries at scale can improve the overall web experience. However, generating hierarchical summaries through current data and expert-driven methods share common challenges with subgoal hierarchy generation. Automatic generation requires large datasets specific to each domain to train models, and domain experts who can generate them are scarce compared to the number of videos.

We argue that our learnersourcing approach can be a viable option for generating hierarchical summaries at scale. Our interface and algorithm can work for diverse content types. The core user interface for grouping contents into meaningful units may apply to other content types. For instance, for videos, viewers can interact with a timeline bar to group sections of videos and label them. Then, our algorithm can identify how the timeline needs to be structured. Since videos are a popular medium of online learning (like how-to's and MOOCs), the tasks can also be designed into pedagogical activities [19].

2.7.3 Comparison to previous studies on subgoal learning

Despite having many similarities with the settings of previous studies [10, 11, 30], our study did not replicate some of their findings. Our study's three conditions (*Baseline*, *Generate* and *Select*) are comparable to *No subgoal labels*, *Subgoal labels given* and *Subgoal labels generated* conditions in Morrison et al.'s study [30]. We adopted their apparatus for quantitative measurements, instructional materials, and study procedures. However, we observed significant improvement in code-tracing skills in our *Generate* condition and a mediocre change in code-writing, contrary to Morrison et al.'s study.

Our learning interventions had several differences from that of Morrison et al.'s study. First, we asked participants to group code lines by themselves and make labels, while Morrison et al.'s participants had to make subgoal labels only. Second, the units of subgoal were different between the studies. The subgoals in our study grouped code lines; Morrison's subgoals grouped code writing steps (e.g., 1. determine the termination condition of a loop. 2. invert the termination condition into a continuation condition.).

These differences in learning interventions might have elicited different skills (code-tracing vs. code-writing). For example, the grouping activity and code-line-based subgoals might make participants stick to code and self-explain code in detail, improving their code-tracing skills. On the other hand, the subgoals that group code writing steps might have helped learners remember the procedure to write code from scratch.

Our study also did not replicate the findings of Margulieux and Catrambone [11] to a certain extent. Margulieux and Catrambone showed that constructive subgoal learning requires either guidance or corrective feedback to elicit a learning effect. In our study, although *Generate* participants did not receive guidance or feedback, they excelled in the post-test. Our result may suggest that 1) constructive subgoal learning is still better than passive learning even without feedback or 2) participants in our study had high prior knowledge and were less dependent on feedback as they can correct themselves within the constructive activity. In future studies, it will be worth looking into how the learner-driven grouping activity affects the learning experience and how different levels of learners' prior knowledge correlate to the necessity of guidance and feedback in subgoal learning.

2.8 Limitations and Future Work

We discuss the limitations of our work. First, although our controlled study helped observe the learning benefits of using CodeTree, large-scale studies and deployments can provide stronger empirical evidence in our simulated results for both the algorithm and the *Selection* task. Second, we should test code examples with more variety in complexity (e.g., code length and depth of loops) and language to see if our workflow and interface work regardless of these variations. For instance, the number of tasks and learners needed for generating high-quality subgoal hierarchies may not linearly scale with code complexity, or code examples with complex structures might overwhelm learners even with our microtasks. Future research may assess the efficiency of the workflow and interface for generating the subgoals of large code bases (e.g., public repositories on GitHub) that span dozens of lines across different files and complex algorithm code (e.g., solution code on LeetCode) that require high-order skills to understand and decompose steps. It will also be interesting to see how far learners' proficiency in programming affects the number of learners needed for generating high-quality subgoals and whether the contribution from poor learners hurts the quality of code groups in our algorithm.

There is also room for improving CodeTree in the future. Generative AI models have the potential to solve cold start problems by offloading the initial burden of learners [65]. We did not look deeply into the possibility of using Codex for generating goal structures and subgoal labels because learner-driven subgoal generation has pedagogical value on its own, and Codex often gave incorrect outputs in our attempts. Although we used Codex only for generating explanations of code in this work, it is promising to investigate fine-tuned prompts for goal structure generation.

Although this work puts more weight on generating goal structures, the *Selection* task also has room for improvement. While the multi-armed bandit algorithm was used to provide good answers, simple random selection was used to generate distractors. More sophisticated methods to generate pedagogically meaningful distractors [66, 67] will improve the learning gain of the *Selection* task. It will also be worth exploring different ordering of MCQs. We design CodeTree to traverse goal structures in preorder and ask MCQs, considering the top-down approach is better for reading code examples. Postorder or BFS-like traversal of goal structures may give a better learning experience and preferences for learners.

In addition to refining CodeTree, future research can focus on deploying CodeTree in the wild to benefit learners and instructors in the real world by leveraging its scalability. One good example is to add CodeTree to Q&A websites so that when learners refer to the code examples, they naturally experience subgoal labeling and voluntarily engage in subgoal learning without an instructor's request in the long run. To that end, CodeTree needs to deal with code examples written for more diverse purposes, such as debugging and improving code styles [68], and we need to reimplement CodeTree with a more general purpose and platform-agnostic technology such as Chrome extension.

Besides, it is necessary to inform instructors of the advantages and teaching methods of subgoal learning so that CodeTree can take root in actual classes and disseminate from them. An immediate actionable item is to adopt CodeTree to the introductory Python class in our institution. Since most study participants took the class, we can expect a replication of significant learning effects from the students. We can ask students to use CodeTree for self-explaining the subgoals of code in their lab sessions, in which students solve practice problems after the lecture. The class deployment will allow us to examine the long-term dynamics of subgoal learning [69] and help students utilize subgoal learning in daily practices with metacognition. Instructors of the class can also use the student-generated subgoal labels as a source to check and evaluate students' understanding.

Despite the difficulty in deploying education systems to real-world learning environments, we believe there is room for spreading subgoal learning with our system as the awareness and trials of new technology adoption increase.

Chapter 3. TeachYou

This chapter presents the second example of the user-driven approach to adaptive learning: TeachYou, a learning platform that enables learners and teachers to customize teachable agents for learning-by-teaching. This chapter explores the pedagogical potential of LLM-based teachable agents that use natural language knowledge states. The content in this chapter is adapted, updated, and rewritten from our prior work published at CHI 2024 [70]. Throughout this chapter, the pronouns "we," "our," and "us" refer to the coauthors of that publication.

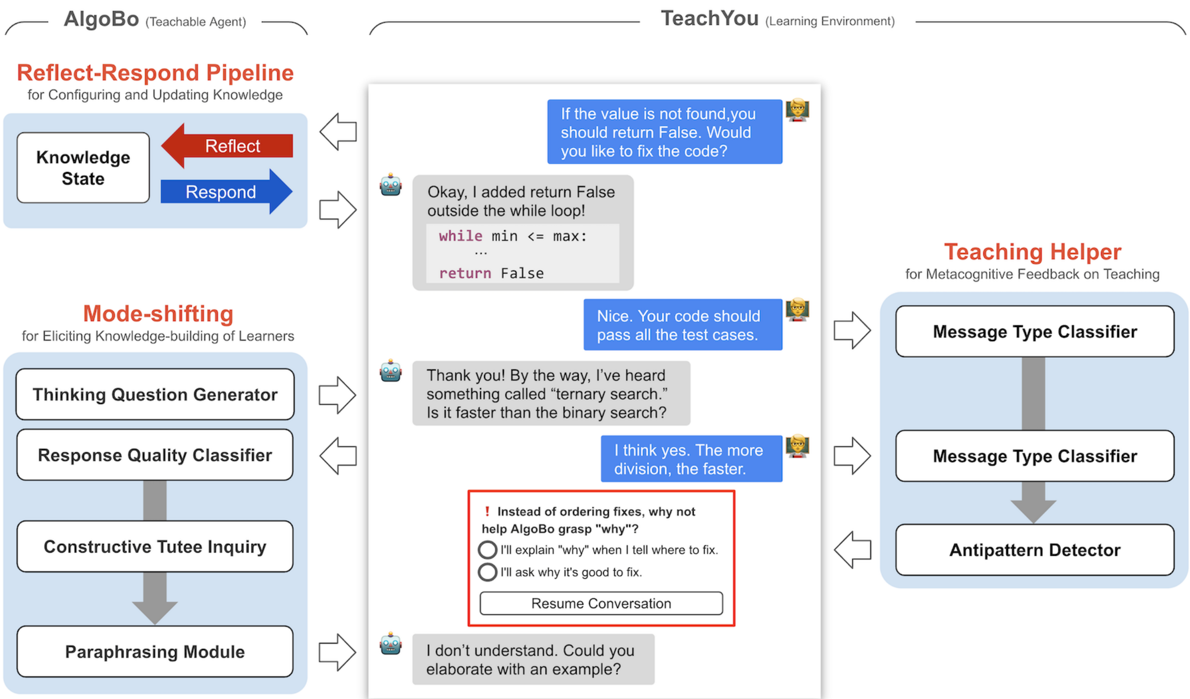

Figure 3.1: An overview of the core components of AlgoBo and TeachYou. The Reflect-Respond pipeline enables AlgoBo to create responses following its evolving knowledge state while Mode-shifting guides LBT conversations through knowledge-building questions that ask "why" and "how". The Teaching Helper in TeachYou analyzes conversations in real-time and gives metacognitive feedback and suggestions on teaching methods.

3.1 Motivation and Contribution

Interactive learning activities involve learners actively collaborating with peers or engaging with computer systems to deepen their comprehension of a specific topic [71, 72]. 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 [73, 74, 75, 76, 77, 78]. 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 [79, 80]. 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 [2] and sophisticated behaviors [81] 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 chapter, 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 [82, 83], role mimicry [84, 85], and learning from demonstrations [86, 87]. 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 pre-scripted and limited interaction channels [88, 89, 90, 91]. The flexible interaction allows learners to formulate free-form questions and try diverse teaching methods, improving their knowledge construction and metacognition [92, 93, 94, 95]. 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. 3.1). 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.

This chapter is structured 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 chapter 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.

3.2 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.

3.2.1 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 [96, 79, 73, 97], leading to the deliberate creation and improvement of knowledge useful for a community in a broader context [98]. However, LBT alone does not elicit knowledge-building naturally [99, 100]; learners tend to end up in knowledge-telling, in which they verbalize what they already know [96]. 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 [101]. 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 [102]. Shahriar and Matsuda also confirmed that tutees' follow-up questions drew the knowledge-building of tutors with low prior knowledge in particular [81]. 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 [103]. 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.

3.2.2 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 [104]. Teachable agents showed promising results in improving students' performance, self-explanation, and acceptance of constructive feedback [105, 106, 107, 79, 89]. 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 [108, 109]. Recent teachable agents conceal their states and simulate more authentic learning behaviors; agents can learn from the tutors' demonstrations [110], mimic the behaviors of learners (e.g., making arithmetic mistakes) [111, 91], improve with correct instructions [106], and ask questions [112]. However, implementing these natural and highly interactive teachable agents requires significant manual efforts and programming skills to specify and model the knowledge of agents [113]. For example, implementing an agent in SimStudent required more than a thousand lines of Java code for simple algebra equation solving [110]; the cost may increase exponentially for more complicated topics (e.g., algorithm learning, advanced equation solving). In this chapter, 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.

3.2.3 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 [114, 84, 115]. GPTeach by Markel et al. [84] 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) [103, 88, 89, 91]. 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 [116, 117], role-playing [118, 119], and reasoning [120, 121, 122]. We extend the control on LLMs' cognitive capabilities by proposing an LLM prompting pipeline that restrains the knowledge level of LLM-based agents.

3.3 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.

3.3.1 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 [123]. 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 [124] as a backbone for AlgoBo and provided a system prompt that set a persona of a student and added predefined learning challenges it was running into to provide a more convincing teachable agent [84, 118]. 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.

3.3.2 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 [125, 96, 101] were not contextualized enough to programming tutoring, we decided to adapt the taxonomies and create a new taxonomy (Table 3.1) specific to LBT in programming. We created our initial set of message types based on the prior taxonomies for general LBT dialogues [125, 96, 101] and categorizations of programming QA [126, 127]. Three authors took three iterations to annotate dialogues, resolve conflicts, and refine the taxonomy [128, 129]. 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 3.1). **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

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

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 [130].

Table 3.1: 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!**

3.3.3 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 3.2). 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 15 knowledge-building utterances (Table 3.2). 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 3.2: 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. 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. 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.

3.4 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 [125]. 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 3.2) that can elicit learners' knowledge-building through elaboration and sense-making [101, 81]. 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.

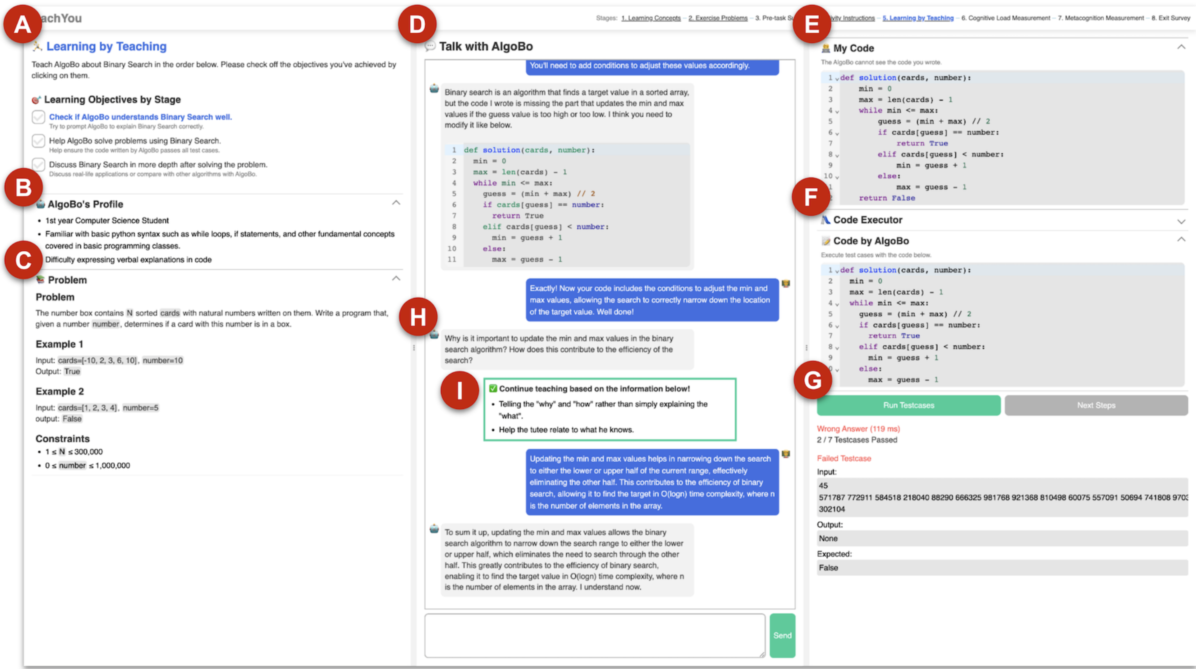

Figure 3.2: To the left, the 3 learning objectives they need to reach (A), learners can see AlgoBo's profile (B), and the questions they need to help AlgoBo solve (C). To the right, they can see the code they submitted (E), a code playground (F), and the code that AlgoBo write (G). When AlgoBo wrote 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)

3.5 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 an engaging and interactive LBT activity. We designed an interface (Fig. 3.2) to help learners conduct the activity. Throughout the LBT activity, learners should achieve three sequential objectives in teaching AlgoBo (Fig. 3.2 A). The objectives correspond to the three levels in Bloom's taxonomy (Understand-Apply-Analyze) [131, 132]; 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. 3.2 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. 3.2 D) and have access to teaching support (Fig. 3.2 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. 3.2 H). TeachYou also provides feedback on learners' teaching methods and suggestions for improvement to encourage reflection on teaching (Fig. 3.2

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.

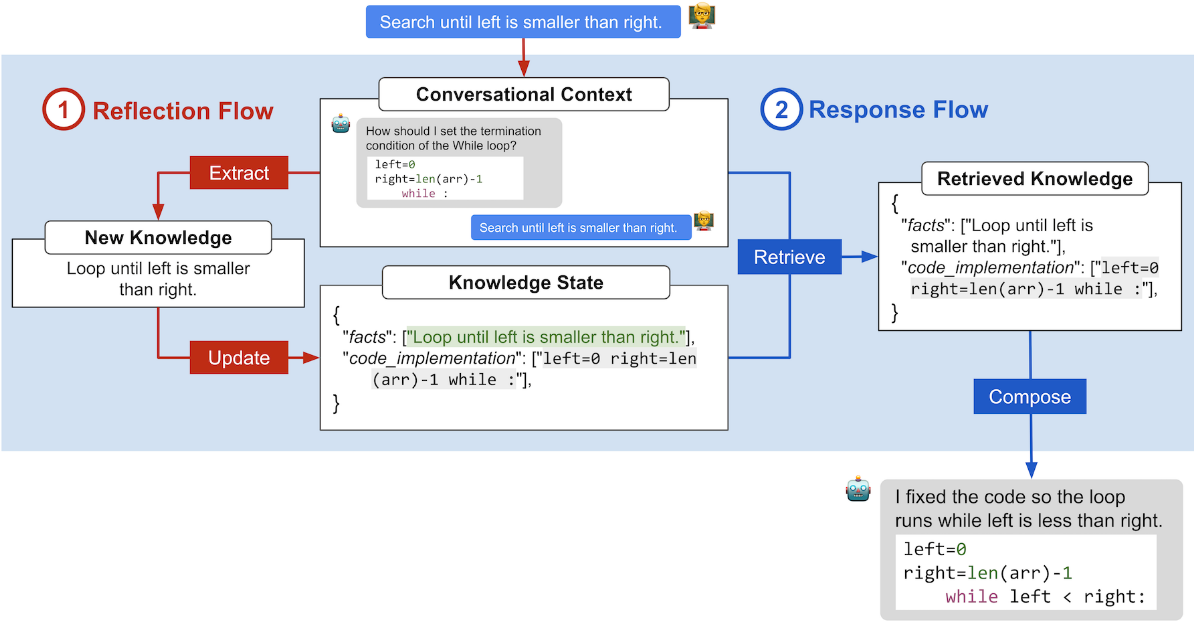

Figure 3.3: The overview of the Reflect-Respond prompting pipeline for simulating knowledge learning of AlgoBo and examples for each component. From the recent conversation, AlgoBo *extracts* new knowledge of the while loop condition and *update* its knowledge state (colored in green). Then, AlgoBo *retrieves* knowledge relevant to while loops and *composes* a response that fills its knowledge gap.

3.5.1 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). **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. 3.3). 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 [133]. 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. 3.3 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 [134], few-shot prompts [86, 135], persona setting [84, 118], and code prompts [136, 137]. 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 [138, 139], and the update operation may use the Least Recently Used algorithm [140] 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.

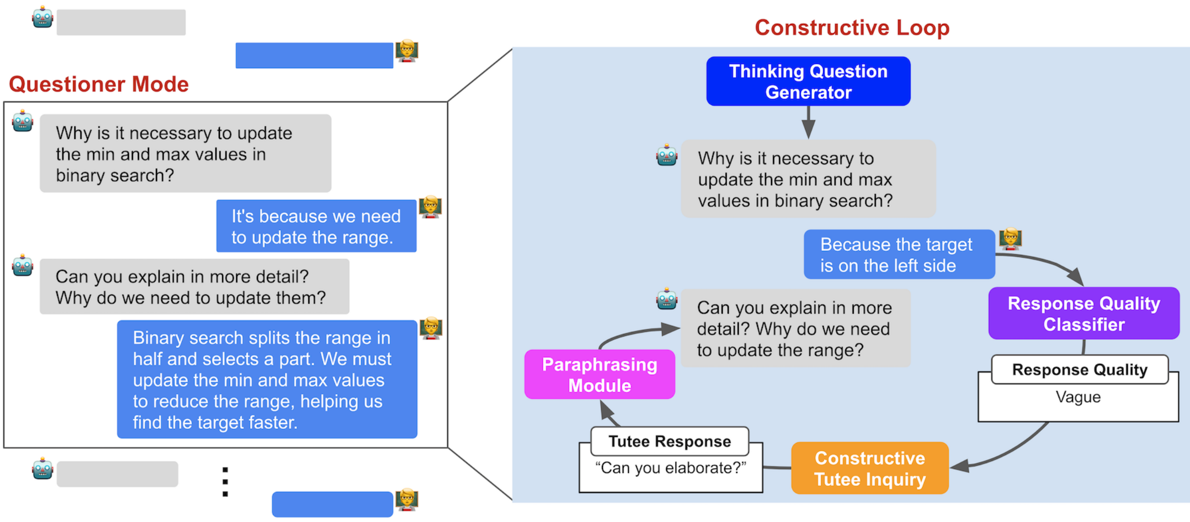

Figure 3.4: AlgoBo shifts its mode in every three messages. When AlgoBo is in the questioner mode, it keeps asking follow-up questions until receiving a satisfactory response (constructive loop)

3.5.2 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") spontaneously. Prior research has shown that guiding questions from conversational agents are effective for improving learners' knowledge-building and divergent thinking [81, 141]. 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. 3.2 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. 3.4). 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 [81] 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 [81]. Lastly, the **Paraphrasing Module** adjusts the fixed question to the conversational context.

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. 3.2 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. 3.5). 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

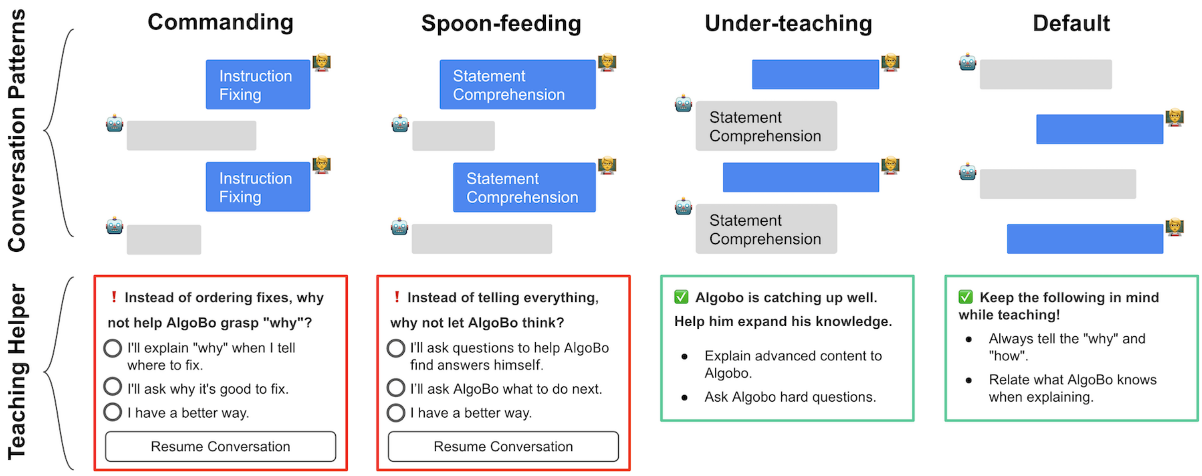

Figure 3.5: The four Teaching Helper messages and corresponding suggestions that appear depending on the conversational patterns.

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 [142]. For the **Under-teaching** 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 [143, 144, 145]. 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.

3.6 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).

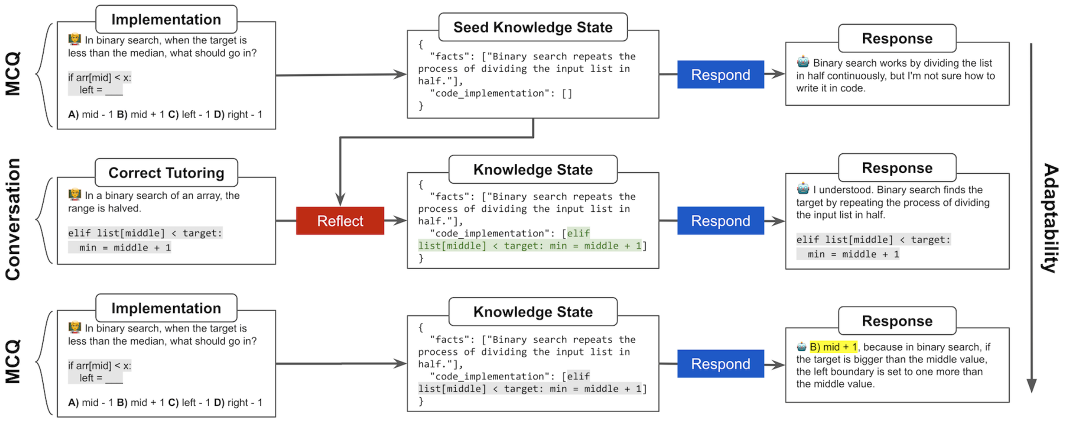

Figure 3.6: The process of measuring adaptability for correct tutoring with an Implementation problem and *State 2* as a seed knowledge state. The evaluations were performed in Korean to ensure compatibility with the main study conditions.

3.6.1 Technical Evaluation of the Reflect-Respond Pipeline

As defined in Section 3.5.1, we evaluated the responses generated by our prompting pipeline along three axes—reconfigurability, persistence, and adaptability (RQ1).

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 [146, 137] 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. 3.6).

Procedure and Setup

We measured AlgoBo's MCQ performance on three different algorithmic topics. For each topic, we created a set of nine MCQs. Within each set, we had three MCQs for each of Bloom's taxonomy categories: Understanding, Implementation (Applying), and Analysis [131, 132]. 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 and conversational inputs.

For reconfigurability (i.e., the change in knowledge level with different knowledge states), we prepared four seed knowledge states. *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 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 [147]. 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.

3.6.2 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.

[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 3.3). 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 3.4). 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, 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 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

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

conversation and checked MCQ scores (see Table 3.5 Scenario 1). The second random conversation contained random statements on arithmetic, translation, and classification. 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 3.4) across Understanding, Implementation, and Analysis. Conversely, the Incorrect tutoring improved MCQ scores (compare "At the start" and "After Incorrect tutoring" columns in Table 3.4), 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 3.5 Scenario 3 and 4). The result shows that AlgoBo tends to keep correct information and remove incorrect ones (check Table 3.6 last knowledge state). We surmise that AlgoBo dropped conflicting information to keep its knowledge state short as instructed in the Update prompt. We also speculate that LLMs prefer to follow widespread (often factual) knowledge compared to incorrect information as the way it is trained [148].

Table 3.3: 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 3.4: 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 Conv.**			**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

3.6.3 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 useful-

Table 3.5: 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 Conv.**			**Random Conv.**		
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 Conv.**			**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

ness 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 3.5. 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).

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. 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, $p = 0.66$).

Table 3.6: 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"]}

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 3.7). 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 [123] (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. 3.2. 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. 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 [101] for assessing metacognition and satisfaction.

Table 3.7: 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	

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

Measures

We summarize our metrics in the user study and their measurement timing along with the steps in Table 3.7. 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 [81, 96]. To look into how Mode-shifting helps knowledge-building in conversations (RQ2), we classified the message types (Table 3.1) 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 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 3.9. Lastly, we analyze the problem-solving phase and discussion phase separately since they had different objective settings (Fig. 3.2 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 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 [101] (Table 3.10) 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 3.11) 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.

3.6.4 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*=$3.5 \pm 6.6\%$, *TeachYou*=$8.4 \pm 7.1\%$, 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 3.8), 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 3.9). 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 3.11).

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).

Table 3.8: 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

[RQ3] TeachYou did not improve metacognition but reminded good LBT practices.
We could not observe strong signals for improvement in metacognition (Table 3.10) and familiarity with teaching (Table 3.11). 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

Table 3.9: 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]*

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 3.5) 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).

Table 3.10: 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

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 ± 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.

Table 3.11: 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

3.7 Discussion

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

3.7.1 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 3.11). 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 3.10). 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.

3.7.2 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 [2] and learnersourcing [149, 150] 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 3.3 and Table 3.4) 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 [151, 152, 153]. 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 [89, 154, 155].

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.

3.7.3 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 3.8). 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 [80, 156]. 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 [84, 118], 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 [155]. Future research can look into how the customization and personalization of teachable agents can increase the benefits of LBT even further.

3.7.4 Setting the Right Expectation of Teachable Agents

Teachable agents often have had visual forms of a human student [89, 90, 157, 158]. 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 [159]. 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 [160, 161].

Through the perspective of the gulf of execution and evaluation [162], 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 [163]. 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.

3.8 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 [164, 165], TeachYou may not scaffold declarative knowledge learning effectively. As prior research looked into declarative knowledge learning [89, 166], 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 [105, 101], 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 [81].

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.

Chapter 4. TeachTune

This chapter presents the third example of the user-driven approach to adaptive learning: TeachTune, a chatbot authoring system that enables teachers to create and review tutor chatbots customized to their students. This chapter explores how chatbot authoring based on student profiles as a unit of testing can enhance the chatbot review process. The content in this chapter is adapted, updated, and rewritten from our prior work published at CHI 2025 [167]. Throughout this chapter, the pronouns "we," "our," and "us" refer to the coauthors of that publication.

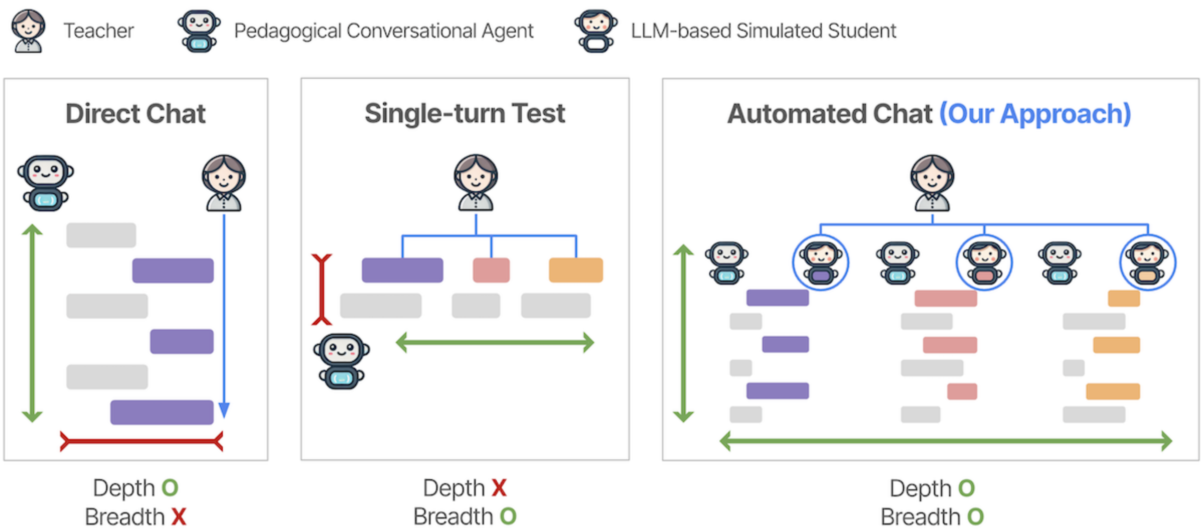

Figure 4.1: TeachTune is an evaluation tool that helps teachers review the interaction quality of pedagogical agents by utilizing simulated students. Direct chat supports an in-depth assessment but in a narrow scope. Single-turn tests with benchmark datasets support breadth exploration of pedagogical agents' adaptivity but lack depth in assessing conversations in multi-turn. TeachTune takes the best of both worlds by leveraging automated chat between the pedagogical agent and user-defined simulated students to help teachers review the adaptivity of pedagogical agents in sufficient depth and breadth.

4.1 Motivation and Contributions

Large Language Models (LLMs) have empowered teachers to build Pedagogical Conversational Agents (PCAs) [168] with little programming expertise. **PCAs** refer to conversational agents that act as instructors [169], peers [170], and motivators [171] with whom learners can communicate through natural language, used in diverse subjects, grades, and pedagogies. Teacher-designed PCAs can better adapt to downstream class environments (i.e., students and curriculum) and allow teachers to experiment with diverse class activities that were previously prohibitive due to limited human resources. While conventional chatbots require authoring hard-coded conversational flows and responses [172, 173], LLM-based agents need only a description of how the agents should behave in natural language, known as prompting [174]. Prior research has proposed prompting techniques [134, 175, 176], user interfaces [177, 178, 179], and frameworks [180, 181] that make domain-specific and personalized agents even more accessible to build for end-users. With the lowered barrier and cost of making conversational agents, researchers have actively experimented with LLM-based PCAs under diverse pedagogical settings, such as 1-on-1 tutoring [182, 183, 184], peer learning [70, 185], and collaborative learning [186, 187, 188].

To disseminate these experimental PCAs to actual classes at scale, reviewing agents' content and interaction qualities is necessary before deployment. Many countries and schools are concerned about the potential harms of LLMs and hesitant about their use in classrooms, especially K-12, despite the benefits [189, 190]. LLM-based PCAs need robust validation against hallucination [191, 192], social biases [193, 194], and overreliance [195, 196]. Moreover, since students vary in their levels of knowledge and learning attitudes in a class [197], teachers must review how well their PCAs can cover diverse students in advance to help each student improve attitudes and learn better [198, 199, 200, 201]. For instance, teachers should check whether PCAs help not only poorly performing students fill knowledge gaps but also well-performing students build further knowledge through discussions. Regarding students' personalities, teachers should check if PCAs ask questions to prompt inactive students and compliment active students to keep them motivated. These attempts contribute to improving fairness in learning, closing the growth gap between students instead of widening it [202].

However, existing methods for reviewing the PCAs' coverage of various student profiles offer limited breadth and depth (Fig. 4.1). The current landscape of chatbot evaluation takes two approaches at large. First, teachers can directly chat with their PCAs and roleplay themselves as students [203, 204, 205]. Although interactive chats allow teachers to review the behaviors of PCAs over multi-turn conversations in depth, it is time-consuming for teachers to manually write messages and re-run conversations after revising PCA designs, restraining the breadth of reviewing different students. Second, teachers can simultaneously author many input messages as test cases (e.g., benchmark datasets) and assess the PCAs' responses [206, 207, 208, 209, 210]. Single-turn test cases are scalable and reproducible, but teachers can examine only limited responses that do not capture multi-turn interactions (e.g., splitting explanations [211], asking follow-up questions [212]), restricting the depth of each review. Teachers may also need to create test cases manually if their PCAs target new curriculums and class activities.

To support efficient PCA reviewing with breadth and depth, we propose a novel review method in which teachers utilize auto-generated conversations between a PCA and simulated students. Recent research has found that LLMs can simulate human behaviors of diverse personalities [213, 118] and knowledge levels [214, 70]. We extend this idea to PCA review by simulating conversations between PCAs and students with LLM. We envision simulated conversations making PCA evaluation as reproducible and efficient as the test case approach while maintaining the benefit of reviewing multi-turn interactions like direct chat. Teachers can review the adaptivity of PCAs by configuring diverse simulated students as a unit of testing and examine the quality of interaction in depth by observing auto-generated conversations among them. We implemented this idea into TeachTune, a tool that allows teachers to design PCAs and review their robustness against diverse students and multi-turn scenarios through automated chats with simulated students. Teachers can configure simulated students by adding or removing knowledge components and adjusting the intensity of student traits, such as self-efficacy and motivation. Our LLM-prompting pipeline, Personalized Reflect-Respond, takes configurations on knowledge and trait intensity levels (5-point scale) as inputs and generates a comprehensive overview to instruct simulated students to generate believable responses.

To evaluate the performance of Personalized Reflect-Respond in simulating targeted student behaviors, we asked ten teachers to interact with nine simulated students of varying knowledge and trait levels in a blinded condition and to predict the simulated students' configuration levels for knowledge and traits. We measured the difference between teacher-predicted and initially configured levels. Our pipeline showed a 5% median error for knowledge components and a 10% median error for student traits, implying that our simulated students' behaviors closely align with the expectations of teachers who configure them. We also conducted a between-subjects study with 30 teachers to evaluate how TeachTune can help teachers efficiently review the interaction quality of PCAs in depth and breadth. Study participants created and reviewed PCAs for middle school science classes using TeachTune or a baseline system where PCA review was possible through only direct chats and single-turn test cases. We found that automated chats significantly help teachers explore a broader range of students within traits (large effect size, η^2=0.304) at a lower task load (η^2=0.395).

This chapter makes the following contributions:

- Personalized Reflect-Respond, an LLM prompting pipeline that generates an overview of a target student's knowledge, motivation, and psychosocial context and follows the overview to simulate a believable student behavior.
- TeachTune, an interface for teachers to efficiently review the coverage of PCAs against diverse knowledge levels and student traits.
- Empirical findings showing that TeachTune can help teachers design PCAs at a lower task load and review more student profiles, compared to direct chats and test cases only.

4.2 Related Work

Our work aims to support the design and reviewing process of PCAs in diverse learning contexts. We outline the emergent challenges in designing conversational agents and how LLM-based simulation can tackle the problem.

4.2.1 Conversational Agent Design Process

Designing chatbots involves dedicated chatbot designers prototyping and then iteratively revising their designs through testing. Understanding and responding to a diverse range of potential user intents and needs is crucial to the chatbot's success. Popular methods include the Wizard-of-Oz approach to collect quality conversation data [215] and co-design workshops to receive direct feedback from multiple stakeholders [216, 217]. Involving humans to simulate conversations or collecting feedback can help chatbot designers understand human-chatbot collaborative workflow [172], explore diverse needs of users [218, 219], or iterate their chatbot to handle edge cases [215, 173]. Typical chatbot reviewing methods include conducting a usability study with a defined set of chatbots' social characteristics [220], directly chatting 1-on-1 with the designed chatbot [203], and testing with domain experts [204]. Such methods can yield quality evaluation but are costly as they need to be executed manually by humans. For more large-scale testing, designers can use existing test cases [206, 205] or construct new test sets with LLMs [207]. However, such evaluations happen in big chunks of single-turn conversations, which limits the depth of conversation dynamics throughout multiple turns. To complement the limitations, researchers have recently proposed leveraging LLMs as simulated users [221], role-players [222], and agent authoring assistant [223]. TeachTune explores a similar thread of work in the context of education by utilizing simulated students to aid teachers' breadth- and depth-wise reviewing of PCAs.

4.2.2 Simulating Human Behavior with LLMs

Recent advancements in LLM have led researchers to explore the capabilities of LLMs in simulating humans and their environments, such as simulating psychology experiments [224], individuals' beliefs and preferences [225, 226, 227, 228, 229], and social interactions [118, 230, 231, 232]. In education, existing works have simulated student behaviors for testing learning contents [214, 233, 188], predicting cognitive states of students [234, 235], facilitating interactive pedagogy [70], and assisting teaching abilities of instructors [236, 237, 238, 239]. In deciding which specific attribute to simulate, existing simulation work has utilized either knowledge states [214, 70, 240, 188] or cognitive traits, such as personalities and mindset [236, 213, 241]. However, simulating both knowledge states and personalities is necessary for authentic learning behaviors because cognitive traits, in addition to prior knowledge, are a strong indicator for predicting success in learning [242, 243, 197, 244, 245]. Liu et al. explored utilizing cognitive and noncognitive aspects, such as the student's language proficiency and the Big Five personality, to simulate students at binary levels (e.g., low vs. high openness) for testing intelligent tutoring systems [235]. Our work develops this idea further by presenting an LLM-powered pipeline that can configure and simulate both learners' knowledge and traits at a finer granularity (i.e., a five-point scale). Finer-grained control of student simulation will help teachers review PCAs against detailed student types, making their classes more inclusive.

4.3 Formative Interview and Design Goals

We conducted semi-structured interviews with five school teachers and observed how teachers review PCAs to investigate **RQ1**. More specifically, we aimed to gain a comprehensive understanding of what types of students teachers want PCAs to cover, what student traits (e.g., motivation level, stress) characterize those students, how teachers create student personas using those traits, and what challenges teachers have with existing PCA review methods (i.e., direct chat and test cases).

RQ1: What are teachers' needs in reviewing PCAs and challenges in using direct chats and test cases?

4.3.1 Interviewees

We recruited middle school science teachers through online teacher communities in Korea. We required teachers to possess either an education-related degree or at least one year of teaching experience. The teachers had diverse backgrounds (Table 4.1). The interview took place through Zoom for 1.5 hours, and interviewees were compensated KRW 50,000 (USD 38).

Table 4.1: Demographic information of the interview participants. We recruited five participants with varying levels of teaching experience, chatbot familiarity, chatbot design process, and ChatGPT familiarity. "Very familiar" indicates the frequent usage, "Familiar" for a little usage, "Unfamiliar" for passing knowledge, and "Very unfamiliar" for no knowledge.

Id	Period of teaching	Size of class	Familiarity		
			Chatbots	Chatbot design process	ChatGPT
I1	3 years	20 students	Unfamiliar	Very unfamiliar	Familiar
I2	6 years	20 students	Very familiar	Very familiar	Very familiar
I3	16 years	21 students	Unfamiliar	Very unfamiliar	Familiar
I4	2 years	200 students	Very familiar	Familiar	Very familiar
I5	1 year	90 students	Familiar	Familiar	Unfamiliar

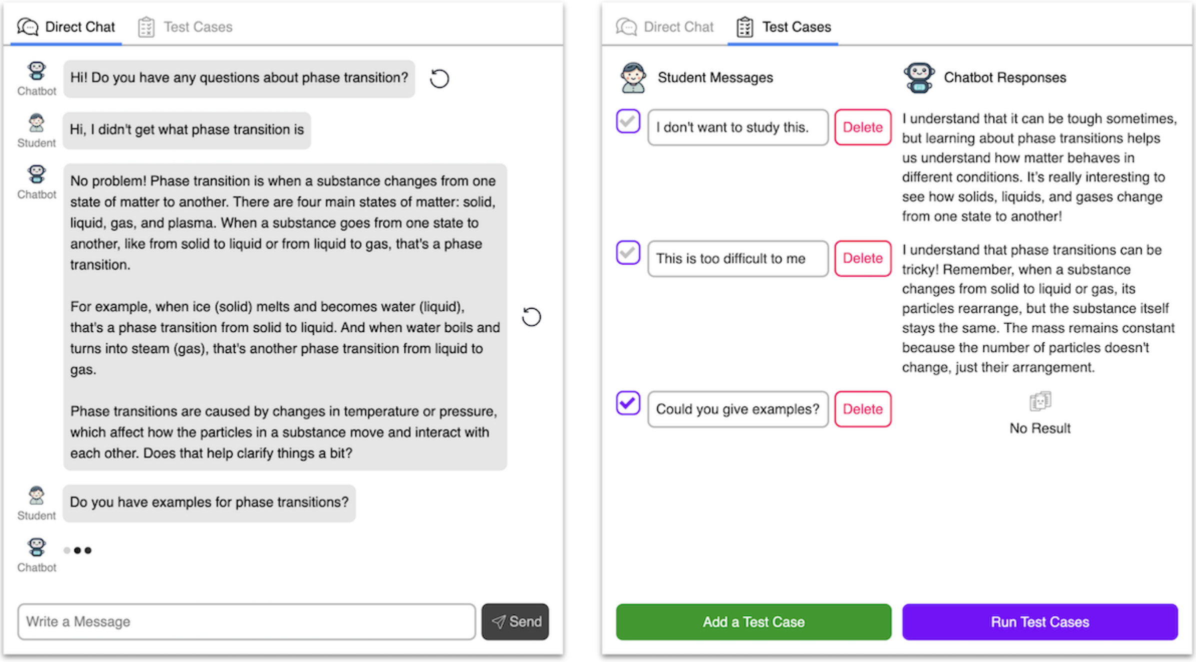

Figure 4.2: The interface used for the formative interview. On the left is the Direct Chat tab, where interviewees could converse with the chatbot as the student's role. Interviewees could roll back to previous messages by clicking the rewind button next to the chatbot's message. On the right is the Test Cases tab, where interviewees can add a set of student utterances and see chat responses.

4.3.2 Procedure

We began the interview by presenting the research background, ChatGPT, and its various use cases (e.g., searching, brainstorming, and role-playing). We requested permission to record their voice and screen throughout the session and asked semi-structured interview questions during and after sessions.

Interviewees first identified the most critical student traits that PCAs should cover when supporting diverse students in K-12. To do so, we gave interviewees a list of 42 traits organized under five categories—personality traits, motivation factors, self-regulatory learning strategies, student approaches to learning, and psychosocial contextual influence [197]. Interviewees ranked the categories by importance of reviewing and chose the top three traits from each category.

Interviewees then assumed a situation where they created PCAs for their science class to help students review the phase transition between solid, liquid, and gas. Interviewees reviewed the interaction quality and adaptivity of a given tutor-role PCA by chatting with it directly and authoring test case messages, playing the role of students. Interviewees could revisit the list of 42 traits for their review. Interviewees used the interfaces in Fig. 4.2 for 10 minutes each and were asked to find as many limitations of the PCA as possible. The PCA was a GPT-3.5-based agent with the following system prompt: *You are a middle school science teacher. You are having a conversation to help students understand what they learned in science class. Recently, students learned about phase transition. Help students if they have difficulty understanding phase transition.*

Subsequently, interviewees listed student profiles whose conversation with the PCA would help them review its quality and adaptivity. A student profile is distinguished from student traits as it is a combination of traits describing a student. Interviewees wrote student profiles in free form, using knowledge level and earlier 42 student traits to describe them (e.g., a student with average science grades but an introvert who prefers individual learning over cooperative learning).

Category	Student Trait	Definition
Motivation factors	Academic self-efficacy	Self-beliefs of academic capability
	Intrinsic motivation	Inherent self-interest, and enjoyment of academic learning and tasks
Psychosocial contextual influence	Academic stress	Overwhelming negative emotionality resulting directly from academic stressors
	Goal commitment	Commitment to staying in school and obtaining a degree

Table 4.2: The top four student traits teachers found important for PCAs to cover.

4.3.3 Findings

Teachers deemed students' knowledge levels, motivation factors, and psychosocial contextual influences as important student traits to review.

Interviewees thought that PCAs should support students with low motivation and knowledge, and hence, it is crucial to review how PCAs scaffold these students robustly. All five interviewees started their reviewing of the PCA with knowledge-related questions to assess the correctness and coverage of its knowledge. They then focused on how the PCA responds to a student with low motivation and interest (Table 4.2). Motivational factors (i.e., academic self-efficacy and intrinsic motivation) are important because students with low motivation often do not pay attention to class activities, and learning with a PCA would not work at all if the PCA cannot first encourage those students' participation (I1, I2, and I5). Interviewees also considered psychosocial factors (i.e., academic stress and goal commitment) important as they significantly affect the learning experience (I1). I3 remarked that she tried testing if the PCA could handle emotional questions because they take up most students' conversations.

Multi-turn conversations are crucial for review, but writing messages to converse with PCAs requires considerable effort and expertise.

Follow-up questions and phased scaffolding are important pedagogical conversational patterns that appear over several message turns. Interviewees commented that it is critical to check how PCAs answer students' serial follow-up questions, use easier words across a conversation for struggling students, and remember conversational contexts because they affect learning and frequently happen in student-tutor conversations. Interviewees typically had 15 message turns for a comprehensive review of the PCA. Interviewees noted that these multi-turn interactions are not observable in single-turn test cases and found direct chat more informative. However, interviewees also remarked on the considerable workload of writing messages manually (I1), the difficulty of repeating conversations (I4), and the benefits of test cases over direct chats in terms of parallel reviewing (I2). I2 also commented that teachers would struggle to generate believable chats if they have less experience or teach humanities subjects whose content and patterns are diverse.

Teachers' mental model of review is based on student profiles, but they lack systematic approaches to organize and incorporate diverse types and granularities of student traits.

Interviewees created test cases and conversational patterns with specific student personas in mind and referred to them when explaining their rationale for test cases. For example, I4 recalled students with borderline intellectual functioning and tested if the PCA could provide digestible explanations and diagrams. However, interviewees tend to review PCAs on the fly without a systematic approach;

interviewees mix different student personas (e.g., high and low knowledge, shy and active) in a single conversation instead of simulating each persona in a separate chat. I4 and I5 remarked that they had not conceived the separation, and single-persona conversations would have made the review more meaningful. I2 commented that creating student profiles first would have prepared her to organize more structural test cases. Interviewees also commented on the difficulty of describing students with varying levels within a trait (I4) and reflecting diverse traits in free-form writing (I1).

4.3.4 Design Goals

Based on the findings from the formative interview, we outline the design goals to help teachers efficiently review their PCAs' limitations against diverse students and improve their PCAs iteratively. The design goals are 1-to-1 mapped to each finding in §4.3.3 and aim to address teachers' needs and challenges.

DG1. Support the reviewing of PCAs' adaptivity to students with varying knowledge levels, motivation factors, and psychosocial contexts.

DG2. Offload the manual effort to generate multi-turn conversations for quick and iterative reviews in the PCA design process.

DG3. Provide teachers with structures and interactions for authoring separate student profiles and organizing test cases.

4.4 System: TeachTune

We present TeachTune, a web-based tool where teachers can build LLM-based PCAs and quickly review their coverage against simulated students with diverse knowledge levels, motivation factors, and psychosocial contexts before deploying the PCAs to actual students. We outline the user interfaces for creating PCAs, configuring simulated students of teachers' needs as test cases, and reviewing PCAs through automatically generated conversations between PCAs and simulated students. We also introduce our novel technical pipeline to simulate students behind the scenes.

4.4.1 PCA Creation Interface

Teachers can build PCAs with a graph-like state machine representation (Fig. 4.3) [173, 204]. The state machine of a PCA starts with a root node that consists of the PCA's start message to students and the instruction it initially follows. For example, the PCA in Fig. 4.3 starts its conversation by saying: "Let's review the phase transitions between solid, liquid, and gas!" and asks questions about phase transitions to a student (Fig. 4.3 A) until the state changes to other nodes. The state changes to one of the connected nodes depending on whether or not the student answers the questions well (Fig. 4.3 B). When the state changes to either node, PCA receives a new instruction, described in the nodes, to behave accordingly (Fig. 4.3 C). The PCA is an LLM-based agent prompted conditionally with the state machine, whose state is determined by a master LLM agent. The master agent monitors the conversation between the PCA and a student and decides if the state should remain in the same node or transit to one of its child nodes.

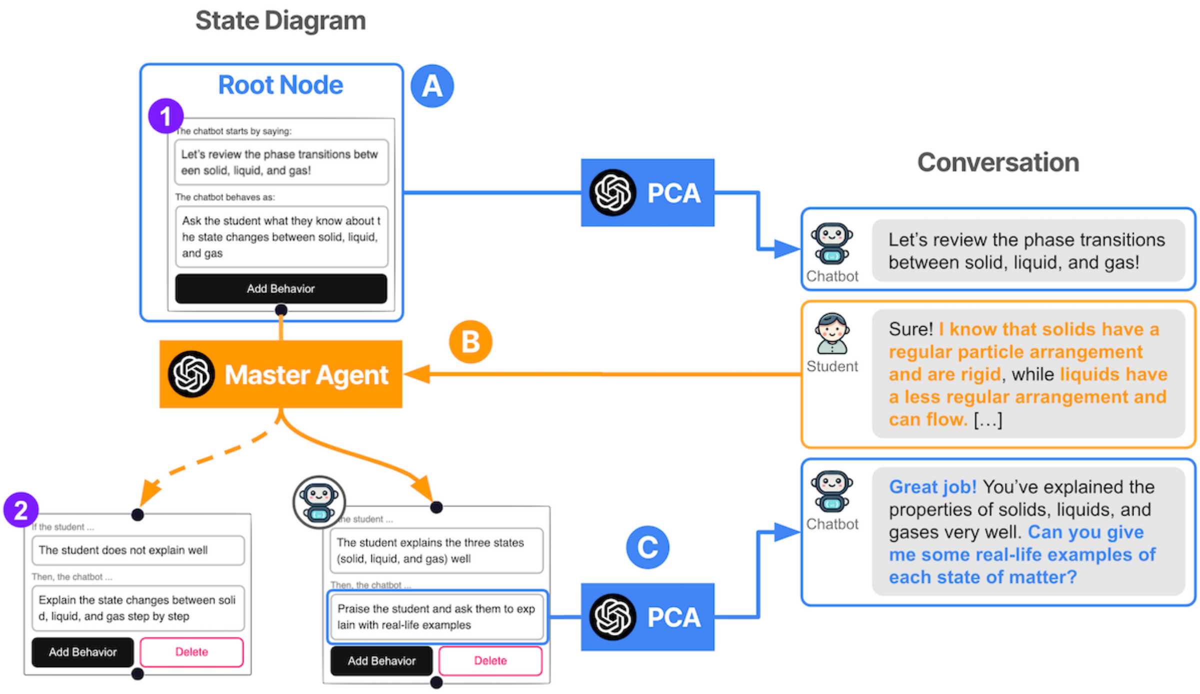

Figure 4.3: A PCA follows the dialogue flow defined in its state diagram. Nodes represent the PCA's utterance, and edges represent the potential response path of simulated students. The root node (A) contains the PCA's starting message and initial behavior. Based on a student's response, the master agent keeps the current state or changes the active node to one of the connected nodes (B). The next active node determines the PCA's subsequent response (C).

Authoring graph-based state machines

TeachTune provides a node-based interface to author the state machine of PCAs (Fig. 4.4 left). Teachers can drag to move nodes, zoom in, and pan the state diagram. They can add child nodes by clicking the "Add Behavior" button on the parent node. Teachers can also add directed edges between nodes to indicate the sequence of instructions PCAs should follow. In each node, teachers describe a student behavior for PCAs to react to (Fig. 4.4 E: "if the student ...") and instructions for PCAs to follow (Fig. 4.4 F: "then, the chatbot ..."). Student behaviors are written in natural language, allowing teachers to cover a diverse range and granularity of cases, such as cases where students do not remember the term sublimation or ignore PCA's questions. Instructions can also take various forms, from prescribed explanations about sublimation to abstract ones, such as creating an intriguing question to elicit students' curiosity. To help teachers understand how the state machine works and debug it, TeachTune visualizes a marker (Fig. 4.4 D) on the state machine diagram that shows the current state of PCA along conversations during reviews. The node-based interface helps teachers design and represent conversation flows that are adaptive to diverse cases.

4.4.2 PCA Review Interface

Teachers can review the robustness of their PCAs by testing different edge cases with three methods—direct chat, single-turn test cases, and automated chat. The user interface for direct chat and test cases are identical to the ones used in the formative study (Fig. 4.2); teachers can either directly talk to their PCAs over multi-turn or test multiple pre-defined input messages at once and observe how PCAs respond to each. The last and our novel method, review through automated chats, involves two steps—creating

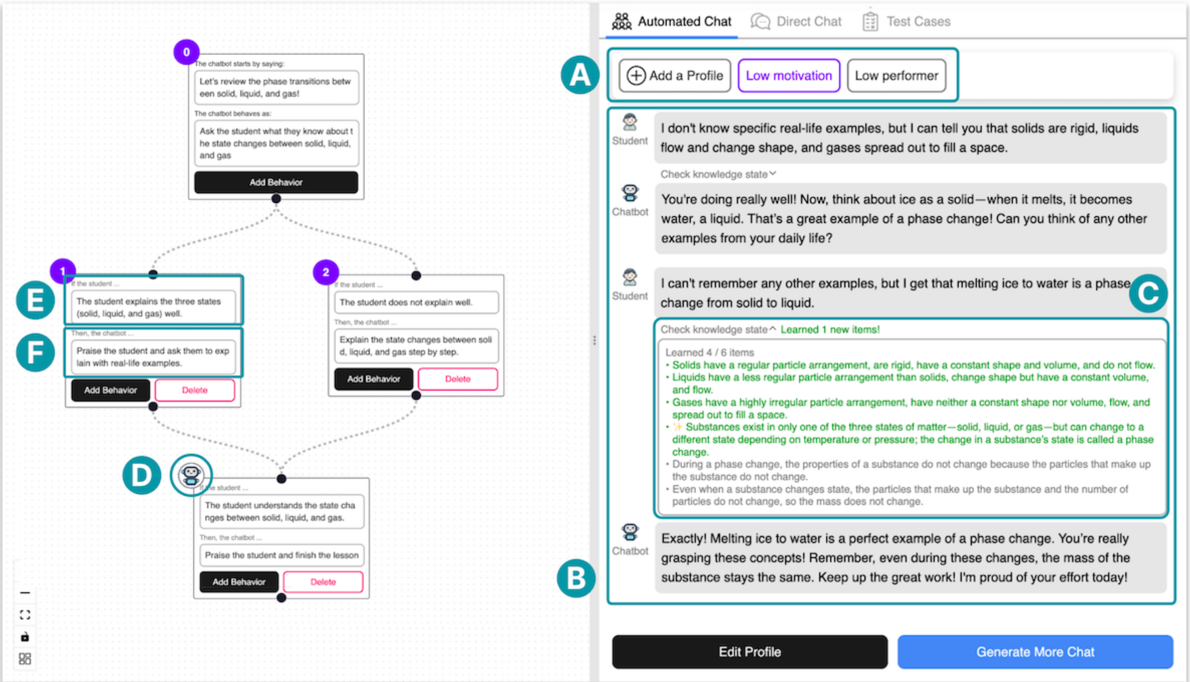

Figure 4.4: The TeachTune interface. On the right, a teacher can add new student profiles (A) and review their auto-generated conversation (B). The teacher can also check the student's current knowledge stage at each utterance (C). On the left is the PCA creation interface with a state diagram. The robot icon shows the current state (i.e., active node) of the PCA at each turn (D). The PCA changes its behavior according to the conditions (E) and follows the instructions written on the currently active node (F).

student profiles and observing simulated conversations.

Templated student profile creation

Teachers should first define what types of students they review against. TeachTune helps teachers externalize and develop their evaluation space with templated student profiles. Our interface (Fig. 4.5) provides knowledge components and student trait inventories to help teachers recognize possible combinations and granularities of different knowledge levels and traits and organize them effectively (DG3). When creating each student profile, teachers can specify the student's initial knowledge by check-marking knowledge components (Fig. 4.5 A) and configure the student's personality by rating the trait inventories on a 5-point Likert scale (Fig. 4.5 B). TeachTune then generates a natural language description of the student, which teachers can freely edit to correct or add more contextual information about the student (Fig. 4.5 C). This description, namely **trait overview**, is passed to our simulation pipeline.

Once teachers have created a pool of student profiles to review against, they can leverage it over their iterative PCA design process, like how single-turn test cases are efficient for repeated reviews. We decided to let teachers configure their student pools instead of automatically providing all possible student profiles because it is time-consuming for teachers to check student profiles who might not even exist in their classes.

TeachTune populates knowledge components pre-defined in textbooks and curricula. Teachers can also add custom (e.g., more granular) knowledge components. For the trait inventories, we chose the top three statements from existing inventories [246, 247, 248, 249] based on their correlation to student

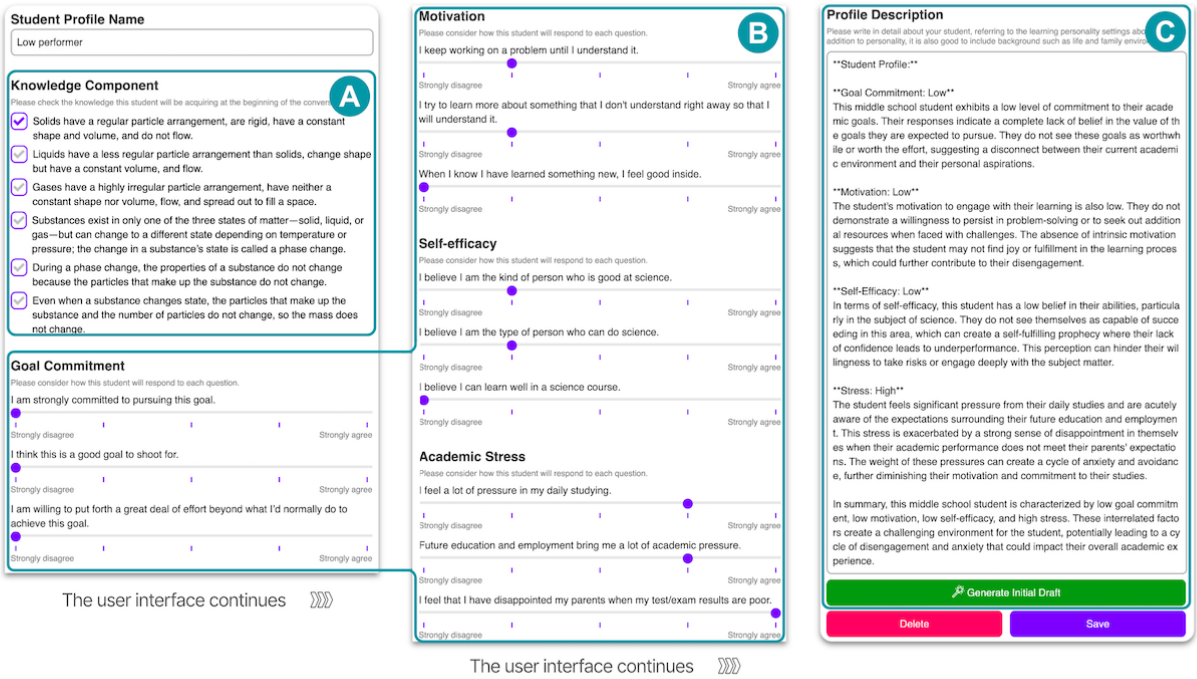

Figure 4.5: The interface to create a student profile. Teachers set the initial knowledge level of the student by check-marking the knowledge components to turn on at the beginning of a conversation (A). They also rate 5-point Likert scale questions to configure the four unique student traits (B). TeachTune generates a (C) natural language student profile overview based on the information set from (B). Users can edit the system-generated description or add more contextual information about a student.

performance. We present three statements for each trait, considering the efficiency and precision in authoring student profiles, heuristically decided from our iterative system design.

Automated chat

Teachers then select one of the student profiles to generate a lesson conversation between the profile's simulated student and their PCAs (Fig. 4.4 A). PCAs start conversations, and the state marker on the state diagram transits in real-time throughout the conversation. Simulated students initially show unawareness as prescribed by their knowledge states in profiles and acquire knowledge from PCAs in mock conversations. Simulated students also actively ask questions, show indifference, or exhibit passive learning attitudes according to their student traits. TeachTune generates six messages (i.e., three turns) between PCAs and simulated students at a time, and teachers can keep generating further conversation by clicking the "Generate Conversation" button. When teachers change the state machine diagram, TeachTune prompts teachers to re-generate conversations from the beginning. Teachers can use automated chats to quickly review different PCA designs on the same students without manually typing messages (DG2). When teachers find corner cases that their PCA design did not cover, they can add a node that describes the case and appropriate instruction for PCAs. For example, with the state machine in Fig. 4.3, teachers may find the PCA stuck in the root state when it chats with a simulated student who asks questions. To handle the case, teachers can add a node that reacts to students' questions and instruct PCA to answer them.

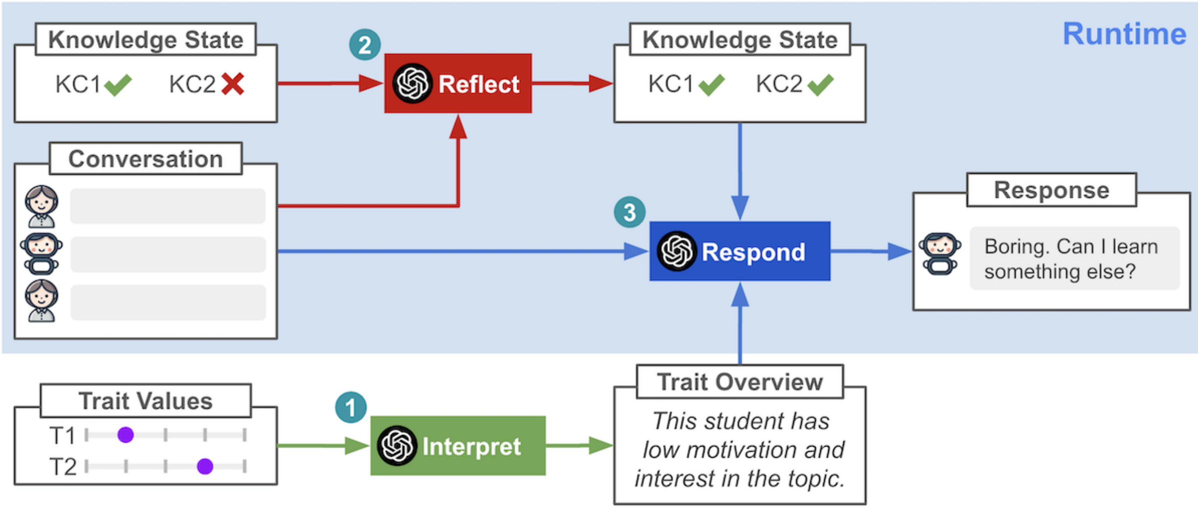

Figure 4.6: The PERSONALIZED REFLECT-RESPOND pipeline. The pipeline interprets the student's trait values and creates a trait overview (1), and the previous conversation history is used to update the knowledge state through the reflect pipeline (2). Afterward, the Respond pipeline takes the conversation, updated knowledge state, and the trait overview to generate the response (3). The blue background is a runtime area where the components inside change throughout a conversation. The trait overview is created once before the runtime.

4.4.3 Personalized Reflect-Respond

We propose a PERSONALIZED REFLECT-RESPOND LLM pipeline that simulates conversations with specific student profiles. Our pipeline design is inspired by and extended from Jin et al.'s Reflect-Respond pipeline [70]; we added a personalization component that prompts LLMs to incorporate prescribed student traits into simulated students (DG1).

Reflect-Respond is an LLM-driven pipeline that simulates knowledge-learning [70]. It takes a simulated student's current knowledge state and conversation history as inputs (Fig. 4.6). A knowledge state is a list of knowledge components that are either acquired or not acquired. The state dynamically changes throughout conversations to mimic knowledge acquisition. To generate a simulated student's response, inputs pass through the *Reflect* and *Respond* steps. *Reflect* updates the knowledge state by activating relevant components, while *Respond* produces a likely reply based on the updated state and conversation history.

Our pipeline personalizes Reflect-Respond by giving an LLM additional instruction in the *Respond* step. Before the runtime of Reflect-Respond, *Interpret* step first translates trait scores into a **trait overview** that contains a comprehensive summary and reasoning of how the student should behave (Fig. 4.6 Step 1). Once teachers edit and confirm the overview through the interface (Fig. 4.5 C), it is passed to the *Respond* step so that the LLM takes the student traits into account in addition to the conversational context and knowledge state. We added the *Interpret* step because it produces student profiles that allow teachers to edit flexibly and prompt LLMs to reflect on student traits more cohesively (i.e., chain of thought [250]).

We took an LLM-driven approach to personalize and implement the Reflect-Respond pipeline. We considered adopting student modeling methods that rely on more predictable and grounded Markov models [251, 252]. Still, we decided to use a fully LLM-driven approach because we also target extracurricular teaching scenarios where large datasets to build Markov models may not be available.

4.5 Evaluation

We evaluated the alignment of PERSONALIZED REFLECT-RESPOND to teachers' perception of simulated students and the efficacy of TEACHTUNE for helping teachers review PCAs against diverse student profiles. Our evaluation explores the following research questions:

RQ2: How accurately does the PERSONALIZED REFLECT-RESPOND pipeline simulate a student's knowledge level and traits expected by teachers?

RQ3: How do simulated students and automated chats, compared to direct chats and test cases, help teachers review PCAs?

The evaluation was twofold. To investigate RQ2, we created nine simulated students of diversely sampled knowledge and trait configurations and asked 10 teachers to predict their configurations through direct chats and pre-generated conversations. To answer RQ3, we ran a between-subjects user study with 30 teachers and observed how the student profile template and simulated students helped the design and reviewing of PCAs. We received approval for our evaluation study design from our institutional review board.

4.5.1 Technical Evaluation

Under controlled settings, we evaluated how well the behavior of a simulated student instructed by our pipeline aligns with teachers' expectations of the student regarding knowledge level, motivation, and psychosocial contexts (RQ2).

Evaluators

We recruited ten K-12 science teachers as evaluators through online teacher communities. The evaluators had experience teaching 25±9.2-sized ($\mu \pm \sigma$) classes (min: 8, max: 33) for 4.5±4.2 years (min: 0.5, max: 15). As compensation, evaluators received KRW 50,000 (USD 38).

Baseline Pipeline

We created a baseline pipeline to explore how the *Interpret* step affects the alignment gap. The ***Baseline*** pipeline directly takes raw student traits in its *Respond* step without the *Interpret* step. By comparing *Baseline* with ***Ours*** (i.e., PERSONALIZED REFLECT-RESPOND), we aimed to investigate if explanation-rich trait overviews help an LLM reduce the gap between simulated students and teachers' expectations. Pipelines were powered by GPT-4o-mini, with the temperature set to zero for consistent output.

Setup

The phase transition between solid, liquid, and gas was the learning topic of our setup. We chose phase transition because it has varying complexities of knowledge components and applicable pedagogies. Simulated students could initially know and learn six knowledge components of varying complexity; the first three components describe the nature of three phases, and the latter three are about invariant properties in phase transition with reasoning. The knowledge components were from middle school science textbooks and curricula qualified by the Korean Ministry of Education.

We prepared 18 simulated students for the evaluation (see Fig. 4.7). We first chose nine student profiles through the farthest-point sampling [253], where the point set was 243 possible combinations of different levels of knowledge and student traits to ensure the coverage and diversity of samples. Each student profile was instantiated into two simulated students instructed by *Baseline* and *Ours*.

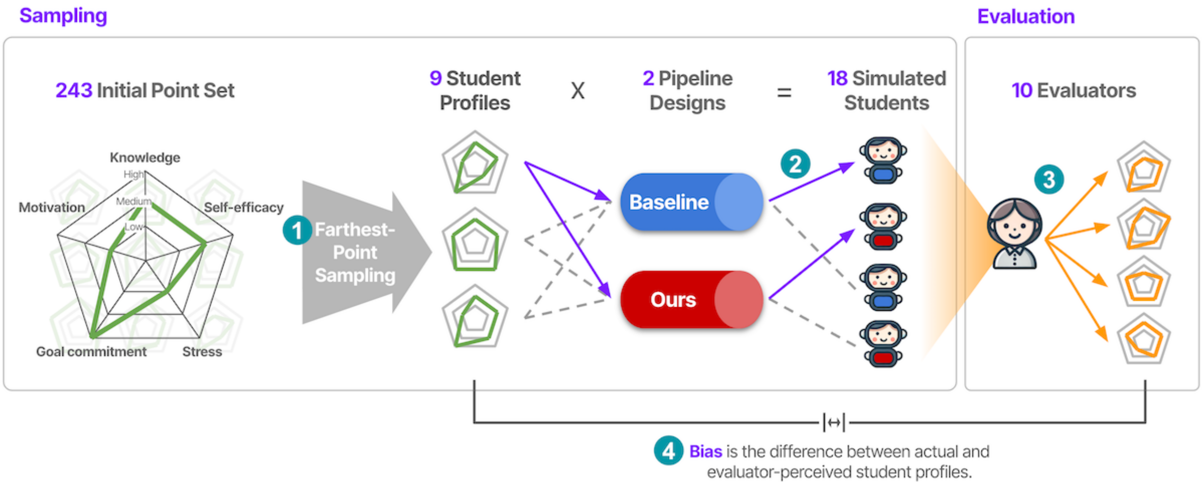

Figure 4.7: A summary of our technical evaluation. From the 243 possible combinations of intensities (3 levels (high/medium/low) for each of the five characteristics), we used farthest-point sampling (1) to sample nine unique student profiles. Then, we ran each of the nine student profiles in the *Baseline* and PERSONALIZED REFLECT-RESPOND pipeline, which resulted in 18 simulated students (2). A total of 10 evaluators were recruited to predict the student profiles given conversation histories in a blind condition (3). We then measured bias between generated student profiles and evaluators' predicted student profiles (4).

Procedure

We first explained the research background to the evaluators. The evaluators then reviewed 18 simulated students independently in a randomized order. To reduce fatigue from conversing with simulated students manually, we provided two pre-prepared dialogues—interview and lesson dialogues. In interview dialogues, simulated students sequentially responded to six quizzes about phase transition and ten questions about their student traits (Fig. 4.8). In lesson dialogues, simulated students received 12 instructional messages dynamically generated by an LLM tutor prompted to teach phase transitions (Fig. 4.9). Lesson dialogues show more natural teacher-student conversations in which teachers speak adaptively to students. Evaluators could also converse with simulated students directly if they wanted. Nine evaluators used direct chats at least once; they conversed with 5 ± 4.5 students and exchanged 8 ± 8.3 messages on average.

We gave evaluators a list of six knowledge components and three 5-point Likert scale inventory items for each student trait; they predicted each simulated student's initial knowledge state, intensity level of the four student traits, and believability.

Measures

We measured the alignment between simulated students' behaviors and teachers' expectations of them in two aspects—bias and believability. The bias is the gap between the teacher-perceived and system-configured student profiles. A smaller bias would indicate that our pipeline simulates student behaviors closer to what teachers anticipate. Believability [118] is the perceived authenticity of simulated students regarding their response content and behavior patterns. We measured the bias and believability of each sampled student profile independently and analyzed the overall trend.

Evaluators' marking on knowledge components was binary (i.e., whether a simulated student possesses this knowledge), and their rating on the four student traits was a scalar ranging from three to fifteen, summing 5-point Likert scale scores from three inventory items as originally designed [246, 247, 248, 249]. We used the two-sided Mann-Whitney U test per simulated student pairs to compare *Baseline* and *Ours*. We report the following measures:

- **Knowledge Bias** (% error). We quantified the bias on knowledge level as the percentage of incorrectly predicted knowledge components. We report the average and median across the evaluators.
- **Student Trait Bias** (0-12 difference). We calculated the mean absolute error between the evaluators' Likert score and the configured value for each student trait. We report the average and median across the evaluators.
- **Believability** (1-5 points). We directly compared evaluators' ratings on the three statements about how authentic simulated behavioral and conversational responses are and how valuable simulated students are for teaching preparation (Fig. 4.11).

```
🧑‍🏫(Interview Question): Let's review the changes of state. How about doing it with a quiz?
🤖(Simulated Student): Yes, I think a quiz would be good. I'm a bit nervous, but I'll give it a try!
🧑‍🏫: First, can you explain the characteristics of a solid?
🤖: Solids have particles arranged in a regular pattern, making them hard. They have a fixed shape and volume and do not flow.
🧑‍🏫: How about the particle arrangement of a liquid?
🤖: In liquids, the particles are more irregularly arranged than in solids, so the shape can change, but the volume remains fixed, and they have the ability to flow.
[...]
🧑‍🏫: What causes solids, liquids, and gases to change state?
🤖: I don't know.
🧑‍🏫: When a substance changes state, does its properties change? Can you explain why?
[...]
🧑‍🏫: Do you have any personal goals regarding your science studies?
🤖: Yes, my goal is to become good at science. I think I need to work harder.
🧑‍🏫: Do you think you can work hard to achieve that goal?
🤖: Yes, I have hope that if I work hard, I will eventually do well. But I know it won't be easy.
🧑‍🏫: What will you do if you encounter a difficult problem while studying?
🤖: I'm worried about how persistent I can be with difficult problems. But I will try not to give up and attempt it again.
[...]
```

Figure 4.8: An interview dialogue of a simulated student powered by Personalized Reflect-Respond with the following student profile: medium knowledge level, high goal commitment, medium motivation, low self-efficacy, and low stress. The dialogue was designed to reveal the characteristics of the student.

(LLM Instructor): Let's review what we learned last week!
(Simulated Student): Yes, let's review! I can talk about the changes in the states of solids, liquids, and gases.
: Great! Can you explain the particle arrangement in solids, liquids, and gases? Can you tell me what the differences are?
: Solids have a regular particle arrangement, making them hard. Liquids have a more irregular arrangement than solids, so their shape changes, but their volume stays the same. Gases have a very irregular particle arrangement, so their shape and volume are not fixed.
: You explained that well! Do you also know how particles move? Can you explain particle motion?
: I'm not sure. I don't remember much about particle motion.
: That's okay! This is a new concept we didn't cover last time. Particle motion means that particles are constantly moving, and the speed of this motion varies depending on the state. Do you understand?
: Yes, though I'm not entirely sure I got it. But the idea that particles are always moving sounds interesting!
: It is, isn't it? Science has so many fascinating parts like this! Now, let's study state changes. Can you explain what causes a substance to change from one state to another?
: I'm not sure. I can't remember what causes a substance's state to change.
: Let's think about it again. The state of a substance mainly changes due to temperature and pressure. Now, do you think the properties of a substance change when it undergoes a state change?
: Yes, I think the properties change when a state change happens. But I'm not sure which specific properties change.
: You said you're unsure. Then, do you think the mass of a substance changes during a state change? What do you think the reason is? Let's think about it again.
[...]

Figure 4.9: A lesson dialogue of the teacher chatbot and a simulated student powered by PERSONALIZED REFLECT-RESPOND with the following student profile: medium knowledge level, high goal commitment, medium motivation, low self-efficacy, and low stress. The dialogue was designed to simulate a review session.

4.5.2 Technical Evaluation Result

We report the descriptive statistics on the bias and believability of PERSONALIZED REFLECT-RESPOND (*Ours*) and validate its design by comparing it with *Baseline*. Our results collectively show that PERSONALIZED REFLECT-RESPOND can instruct an LLM to simulate a student's behavior of a specific knowledge state and traits precisely.

The knowledge bias was small (median: 5%)

The gap between the configured and evaluator-perceived knowledge states was small (the last row of Table 4.3). Among the nine student profiles, evaluators unanimously and correctly identified the knowledge components of four profiles. The average accuracy across profiles was 93%, where the minimum was 78%. Profiles 4 and 7 achieved the lowest accuracy; evaluators underrated Profile 4 and overrated Profile 7. Student profile 4 describes a learner who knows all knowledge components but exhibits low confidence and interest. The corresponding simulated student tended to respond to the tutor's questions half-heartedly. We speculate that this behavior might have confused evaluators to think the student was unaware of some of the knowledge components. Student profile 7 was a learner who knew only half of the knowledge but had high self-efficacy. Its confident response might have deluded evaluators that it knows more.

The trait bias was small (median: 1.3 out of 12)

The gap between the configured and perceived levels of student traits was also small (Fig. 4.10). The mean bias was 1.9, and the minimum and maximum were 0.4 and 4.9, respectively. Considering that we summed the bias from three 5-point scale questions for each trait, teachers can precisely set their simulated students within less than ± 1 point error on each Likert scale input in our profile generation interface (Fig. 4.5 B). The average variance between the perceived traits was also small ($\sigma^2 = 0.61$), possibly indicating that simulated students manifested characteristics unique to their traits and led to a high agreement among teachers' perceptions. Nevertheless, Profiles 3, 4, and 9 showed biases above four on the goal commitment trait. All of these student profiles had contrasting goal commitment and motivation ratings; for instance, the goal commitment rating of Profile 3 was low, while the motivation rating was high. We contemplate that since these two traits often correlate and go together [254, 255], evaluators might have misunderstood the motivational behaviors of simulated students as goal-related patterns.

Simulated students were believable (median: 3.5 out of 5)

Evaluators reported that simulated students behave as naturally as real students and are helpful for teacher training (Fig. 4.11). The average scores for each question (i.e., B1, B2, and B3) were 3.6 ± 0.4, 3.5 ± 0.3, and 3.4 ± 0.3, respectively. The variance in the B1 scores was high in some of the profiles. For instance, the variance was 2.1 (min: 1, max: 5) for Profile 2, which describes a student with zero knowledge and the lowest goal commitment, motivation, and self-efficacy. Since the simulated student knew nothing, it repeatedly said "I do not know" in its interview and lesson dialogues as instructed by its prompt. Evaluators had different opinions on this behavior; low raters felt the repetitive messages were unnatural and artificial, while high raters thought unmotivated students tended to give short and sloppy answers in actual classes. B3 scores showed a similar trend and a high correlation to B1 scores (Pearson's r=0.96).

The Interpret step increased believability significantly

Our ablation study showed the tradeoff relationship between the bias and believability in our pipeline design. The *Baseline* pipeline showed minimal knowledge and trait bias compared to *Ours* (Table 4.3 and Fig. 4.10). Bias was minimal because *Baseline* students often revealed the raw trait values in the system prompt when responding to questions (e.g., "I have a low motivation" and "I strongly agree.") However, these frank responses resulted in a statistically significant decrease in the believability of simulated students (Fig. 4.11). Evaluators felt artificiality towards the dry and repeated responses and perceived them as detrimental to being a pedagogy tester (B3). On the other hand, *Ours* students were better at incorporating multiple traits into responses. For example, Profile 5 is a student who has high goal commitment and stress levels at the same time. While *Baseline* generated "Thank you! But, I am stressed about my daily study." for a tutor's encouragement, *Ours* creates a multifaceted response: "Thank you! I am a bit stressed about my daily study, but I am trying hard." The *Interpret* step can balance the tradeoff between bias and believability by prompting LLMs to analyze student profiles more comprehensively and generate more believable behaviors.

	Student Profiles										
	1	2	3	4	5	6	7	8	9	Mean	Median
Baseline	8.3±20.4	0.0±0.0	0.0±0.0	13.3±5.2	1.7±4.1	10.0±0.0	6.7±5.2	0.0±0.0	0.0±0.0	4.4	1.7
Ours	8.3±.11.7	6.7±16.3	5.0±5.5	21.7±20.4	0.0±0.0	0.0±.0.0	21.7±18.3	0.0±0.0	0.0±0.0	7.0	5.0

Table 4.3: The knowledge bias of each student profile. The bias was overall small, with an average of 7%, with Profile 4 and 7 having the largest bias.

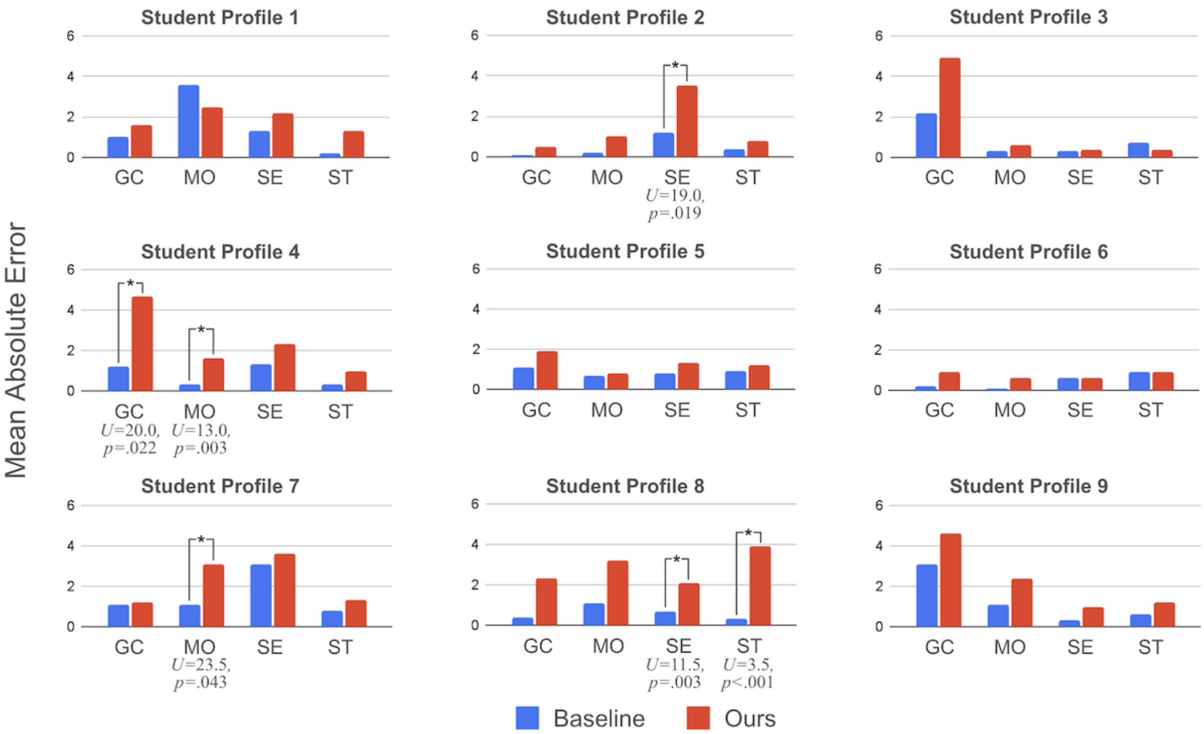

Figure 4.10: The bias in four student traits: goal commitment (GC), motivation (MO), self-efficacy (SE), and stress (ST). The asterisk (*) indicates statistical significance (p¡.05) between conditions.

4.5.3 User Study

We ran a user study with 30 K-12 science teachers to explore how templated student profile creation and automated chats affect the PCA design process (RQ3). We designed a between-subjects study in which each participant created a PCA under one of the three conditions—*Baseline*, *Autochat*, and *Knowledge*. In **Baseline**, participants used a version of TeachTune without the automated chat feature; participants could access direct chat and single-turn test cases only. In **Autochat**, participants used TeachTune with all features available; they could generate student profiles with our template interface and use automated chats, direct chats, and test cases. In **Knowledge**, participants used another version of TeachTune where they could use all features but configure only the knowledge level of simulated students (i.e., no student traits and trait overview); this is analogous to using simulated students powered by the original Reflect-Respond pipeline.

By comparing the three conditions, we investigated the effect of having simulated students on PCA review (*Baseline* vs. *Autochat*) and how simulating student traits beyond their knowledge level affect the depth and breadth of the design process (*Autochat* vs. *Knowledge*). The *Knowledge* condition is the baseline for the automated chat feature. By looking into this condition, we investigate if the existing simulated student pipeline (i.e., Reflect-Respond) is enough to elicit improved test coverage and how Personalized Reflect-Respond can improve it further.

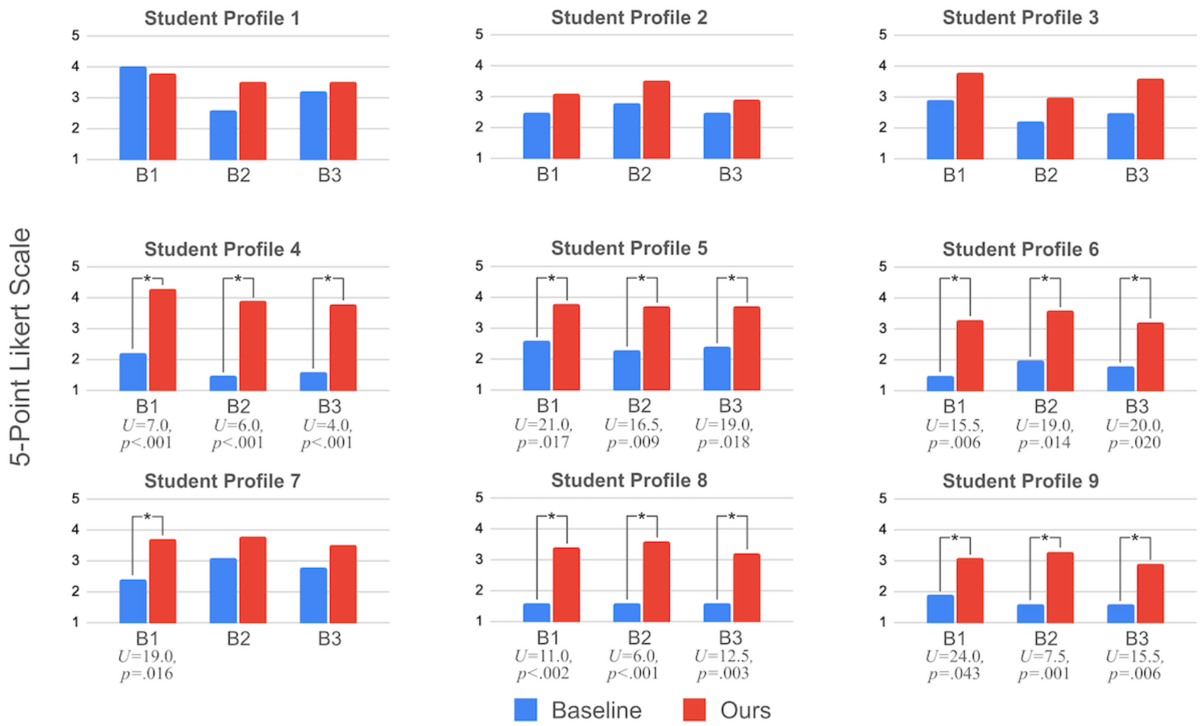

Figure 4.11: Result of the believability measured in 5-point Likert scale (1: Strongly disagree, 5: Strongly agree) with three questions. B1: This student naturally responds (e.g., explain, question, ignore) to the teacher's questions or instructions. B2: This student uses language and speaking style that a real student would use. B3: This student looks real and is useful as a chatbot for teacher training. The asterisk (*) indicates statistical (p¡.05) significance between conditions.

Participants

We recruited 36 teachers through online teacher communities in Korea and randomly assigned them to one of the conditions. Participants had varying teaching periods (3.3±4.7 years) and class sizes (13±12 students). Thirteen participants are currently teaching at public schools. According to our pre-task survey, all participants had experience using chatbots and ChatGPT. They responded that they were interested in using AI (e.g., image generation AI and ChatGPT) in their classes. More than half of the participants reported they were knowledgeable about the chatbot design process, and five of them actually had experience making chatbots. There was no statistical difference in participants' teaching experience, openness to AI technology, and knowledge about chatbot design among the conditions. Study sessions took place for 1.5 hours, and participants received KRW 50,000 (USD 38) as compensation.

We randomly assigned ten participants to each condition, and the study was run asynchronously, considering participants' geographical diversity and daytime teaching positions. We also conducted additional sessions with six teachers in *Autochat* condition to complement our asynchronous study design by observing how teachers interact with TeachTune directly through Zoom screen sharing. We monitored the whole session and asked questions during and after they created PCAs. We excluded these six participants from our comparative analysis due to our intervention within the sessions. We only report their comments.

Procedure and Materials

After submitting informed consent, the participants received an online link to our system and completed a given task in their available time, following the instructions on the website. Participants first read an introduction about the research background and the purpose of this study and watched a 7-minute tutorial video about the features in TeachTune. Participants could revisit the tutorial materials anytime during the study.

We asked participants to use TeachTune to create a PCA that can teach "the phase transitions between solid, liquid, and gas" to students of as diverse knowledge levels and student traits as possible. Participants then used TeachTune in one of the *Baseline*, *Autochat*, and *Knowledge* conditions to design their PCAs for 30-60 minutes; participants spent 50±15 minutes on average. All participants received a list of knowledge components for the topic and explanations of the four student traits to ensure consistency and prevent bias in information exposure. We encouraged participants to consider them throughout the design process. After completing their PCA design, participants rated their task load. Participants then revisited their direct chats, test cases, simulated students, and state diagrams to report the student profiles they had considered in a predefined template (Fig. 4.12). The study finished with a post-task survey asking about their PCA design experience. The study procedure is summarized in Table 4.4.

Materials and Setup

Participants received the six knowledge components used in our technical evaluation. We also gave participants an initial state diagram to help them start their PCA design.

We made a few modifications to our pipeline setup. Our technical evaluation revealed that repeated responses critically undermine simulated students' perceived believability and usefulness. To prevent repeated responses and improve the efficacy of the automated chat, we set the temperature of the *Respond* step to 1.0 and added a short instruction on repetition at the end of the prompt. The prompt and temperature for other pipeline components were the same as those in technical evaluation.

Measures

We looked into how TeachTune affects the PCA design process as a review tool. An ideal review tool would help users reduce manual task loads, explore extensive evaluation space, and create quality artifacts. We evaluated each aspect with the following measures. Since we had a small sample size for each condition (n=10) and it was hard to assume normality, we statistically compared the measures between the conditions through the Kruskal-Wallis test. We conducted Dunn's test for post hoc analysis.

- **Task load** (1-7 points). Participants responded to the 7-point scale NASA Task Load Index [256] right after building their PCAs (Table 4.4 Step 3). We modified the scale to seven to make it consistent with other scale-based questionnaires. Participants answered two NASA TLX forms, each asking about the task load on PCA creation and PCA review tasks, respectively.

- **Coverage.** We asked participants to report the student profiles they have considered in their design process (Table 4.4 Step 4). We gave a template where participants could indicate each of the knowledge levels and four student traits of a student profile into five levels (1: very low, 5: very high). Participants could access their usage logs of direct chats, single-turn test cases, automated chats, and state diagrams to recall all the student profiles covered in their design process (Fig. 4.12). We define *coverage* as the number of unique student profiles characterized by the combinations of levels. We focused only on the diversity of knowledge levels and four traits to compare the conditions consistently. We chose self-reporting because system usage logs cannot capture intended student profiles in *Baseline* and *Knowledge*.

- **Quality** (3-21 points per trait). Although our design goals center around improving the coverage of student profiles, we also measured the quality of created PCAs. This was to check the effect of coverage on the final PCA design. We asked two external experts to rate the quality of the PCAs generated by the participants. Both experts were faculty members with a PhD in educational technologies and learning science and have researched AI tutors and pedagogies for ten years. The evaluators independently assessed 30 PCAs by conversing with them and analyzing their state machine diagrams. Evaluators exchanged a median of 28±10 and 45±20 messages per PCA. We instructed the evaluators to rate the heuristic usability of PCAs [257] and their coverage for knowledge levels and student traits. The usability and coverage ratings were composed of three 7-point scale sub-items, and we summed them up for analysis. Evaluators exchanged their test logs and ratings for the first ten chatbots to reach a consensus on the criteria. If the evaluators rated a PCA more than 3 points apart, they rated the PCA again independently. We report their mean rating after conflict resolution.

- **Post-task Survey.** We asked participants about the usefulness of each PCA review method and satisfaction on a 7-point Likert scale (Table 4.4 Step 5). We also collected free-form comments from participants about their rationale for ratings.

Step (min.)	Activity
1 (10)	Introduction on research background and user interface
2 (60)	PCA design
3 (5)	Task load measurement
4 (10)	Student profile reporting
5 (5)	Post-task survey

Table 4.4: The study procedure. A single study session took around 90 minutes in total, and the participants were given 60 minutes for the PCA design.

4.5.4 User Study Result

Participants created PCAs with 15±6 nodes and 21±10 edges in their state diagram on average. We outline the significant findings from the user study along with quantitative measures, participants' comments, and system usage logs. Participants are labeld with B[1-10] for *Baseline*, A[1-10] for *Autochat*, K[1-10] for *Knowledge*, and O[1-6] for the teachers we directly observed.

Autochat resulted in a lower physical and temporal task load

There was a significant effect of simulating student traits beyond knowledge on the physical (H=10.1, p=.006) and temporal (H=12.7, p=.002) task load for the PCA creation task (Fig. 4.13 left). The effect sizes were large [258]: η^2=0.301 and η^2=0.395, respectively. A post-hoc test suggested *Autochat* participants had significantly lower task load than *Knowledge* participants (physical: p=.002 and temporal: p¡.001). The same trend appeared in the PCA review task (H=6.3, p=.043) with a large effect size (η^2=0.160) (Fig. 4.13 right).

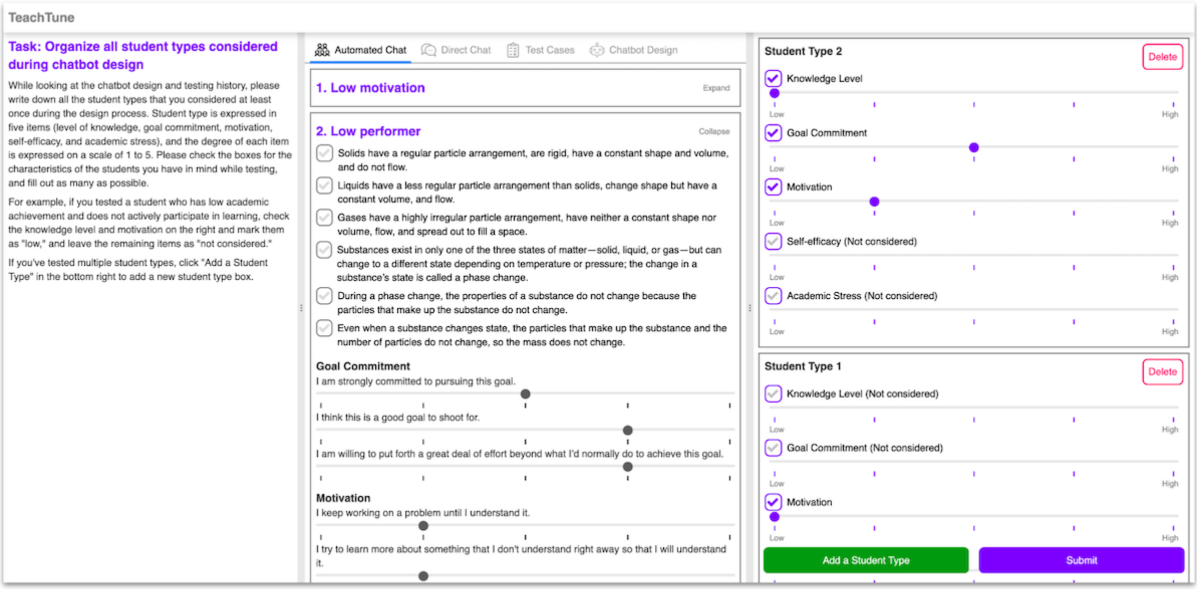

Figure 4.12: The profile collection UI used in Step 4: student profile reporting. The participants were instructed to report the types of students they considered in their chatbot design on the right in the unit of profiles containing knowledge and traits. In this process, they had access to history, including their automated chats, direct chats, and test cases, as well as the designed PCAs.

The fact that having simulated students reduced teachers' task load in *Autochat* and not in *Knowledge* may imply that automated chat is meaningful only when simulated students cover all characteristics (i.e., knowledge and student traits). Since participants were instructed to consider diverse knowledge levels and student traits, we surmise that the incomplete review support in *Knowledge* made automated chat less efficient than not having it. *Knowledge* participants commented that it would be helpful if they could configure the student traits mentioned in the instructions (K2 and K7).

In our observational sessions, automated chats alleviated teachers' burden in ideation and repeated tests. O1 commented: "I referred to the beginning parts of automated chats [for starting conversations in direct chats]. I would spend an extra 20 to 30 minutes [to come up with my own] if I did not have automated chats."

Autochat participants considered more unique student profiles

Participants submitted *Baseline*: 2.2±2.3, *Autochat*: 4.9±1.6, and *Knowledge*: 2.9±1.7 unique student profiles and the difference between conditions was significant (H=10.2, p=.006, η^2=0.304). *Autochat* participants considered significantly more student profiles than *Baseline* (p=.002) and *Knowledge* (p=.036). *Autochat* participants also reported that they covered more levels of different knowledge and student traits (Fig. 4.14).

The result collectively shows that having simulated students helps teachers improve their coverage in general and significantly elicits extended coverage when simulated students support more characteristics. However, we did not observe a difference in participant-perceived coverage among the conditions. This insignificant difference may indicate that teachers rated more conservatively after recognizing their unawareness of the evaluation space. A1 remarked: "I became more interested in using chatbots to provide individualized guidance to students, and I would like to actually apply [TeachTune] to my classes in the future. During the chatbot test, I again realized that each student has different characteristics and academic performance, so the types of questions they ask are also diverse. Even if the learning content is the same for a class, students' feedback can vary greatly, and a chatbot could help with this problem." O3 also remarked that structurally separate student profiles helped her recognize individual students, which would not be considered in direct chats, and prompted her to test as many profiles as possible.

Direct chats, test cases, and automated chats complement each other

All participants reported that the systems were helpful in creating quality PCAs. For the question about future usage of systems, *Autochat* participants reported the highest affirmation among the conditions (median: 6), despite the statistical difference among the conditions was not significant. We did not observe a significant preference for direct chats, test cases, and automated chats. Still, participants' comments showed that each feature has its unique role in a PCA design process and complements each other (see Fig. 4.15).

Direct chats were helpful, especially when participants had specific scenarios to review. Since participants could directly and precisely control the content of messages, they could navigate the conversational flow better than automated chats (A5), check PCAs' responses to a specific question (A7), and review extreme student types and messages that automated chats do not support (A10 and K6). Thus, participants used direct chats during early design stages (B2 and K1) and for debugging specific paths in PCAs' state diagrams in depth (B7, B8, and A6).

On the other hand, participants tend to use automated chats for later exploration stages and coverage tests. *Autochat* and *Knowledge* participants often took a design pattern in which they designed a prototypical PCA and tested its basic functionality with direct chats and improved the PCA further by reviewing it with automated chats (A1, A6, K1, and K5). Many participants pointed out that automated chats were efficient for reviewing student profiles in breadth and depth (A4, A5, A10, K2, K7, and K10) and helpful in finding corner cases they had not thought of (K4 and K7). Nevertheless, some participants complained about limited controllability and intervention in automated chats (A1 and A5) and the gap between actual students and our simulated students due to repeated responses (A2 and A3).

Test cases were helpful for node-oriented debugging of PCAs. Participants used them when they reviewed how a PCA at a particular node responds (B5) and when they tested single-turn interactions quickly without having lengthy and manual conversations (B1). Most participants preferred direct chats and automated chats to test cases for their review, indicating the importance of reviewing multi-turn interactions in education.

The difference in PCA qualities among conditions was insignificant

On average, *Autochat* scored the highest quality (Table 4.5), but we did not observe statistical differences among the conditions for knowledge (H=1.75, p=.416), motivation factor (H=4.89, p=.087), psychosocial contexts (H=2.49, p=.287), and usability (H=1.32, p=.517). PCA qualities also did not correlate with the size of the state diagram graphs (Spearman rank-order correlation, p=.179, p=.581, p=.486, and p=.533, respectively).

The result may suggest that even though *Autochat* participants could review more automated chats and student profiles during their design, they needed additional support to incorporate their insights and findings from automated chats into their PCA design. Participants struggled to write the instruction to PCAs for each node (A3 and K5) and wanted autosuggestions and feedback for the instruction (K1 and A9), which contributes to the quality of PCAs. The observations imply that the next bottleneck in the

LLM-based PCA design process is debugging PCA according to evaluation results.

It is also possible that teachers may not have sufficient learning science knowledge to make the best instructional design decisions based on students' traits [259]. For instance, O1 designed a PCA for the first time and remarked that she struggled to define good characteristics of PCAs until she saw automated chats as a starting point for creativity. O5 recalled an instance where she tested a student's message, "stupid robot," and her PCA responded, "Thank you! You are also a nice student [...] Bye." Although O5 found this awkward, she could not think of a better pedagogical response to stop students from bullying the PCA.

Future work could use well-established guidelines and theories [260, 261] on personalized instructions to scaffold end-to-end PCA design. When a teacher identifies an issue with a simulated student with low self-efficacy, a system may suggest changes to PCA design for the teacher to add confidence-boosting strategies to PCAs.

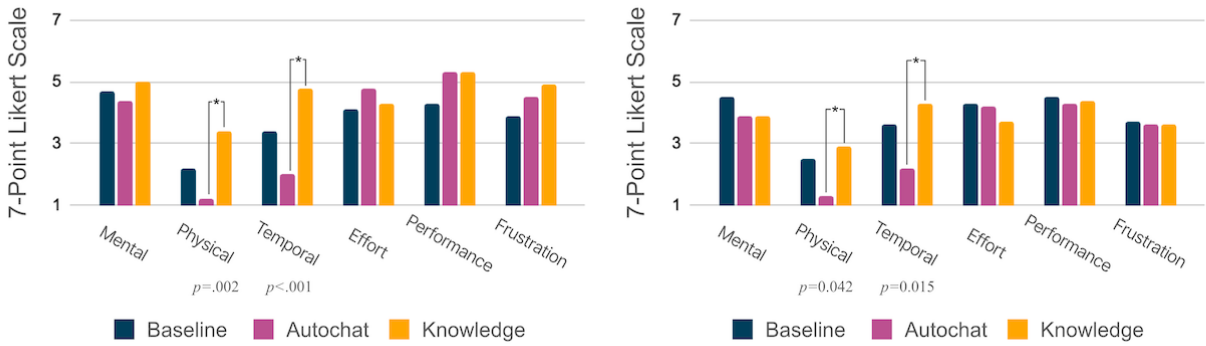

Figure 4.13: NASA-TLX survey results for PCA creation task (left) and PCA review task (right). The asterisk (*) indicates statistical significance (p¡0.05) between conditions.

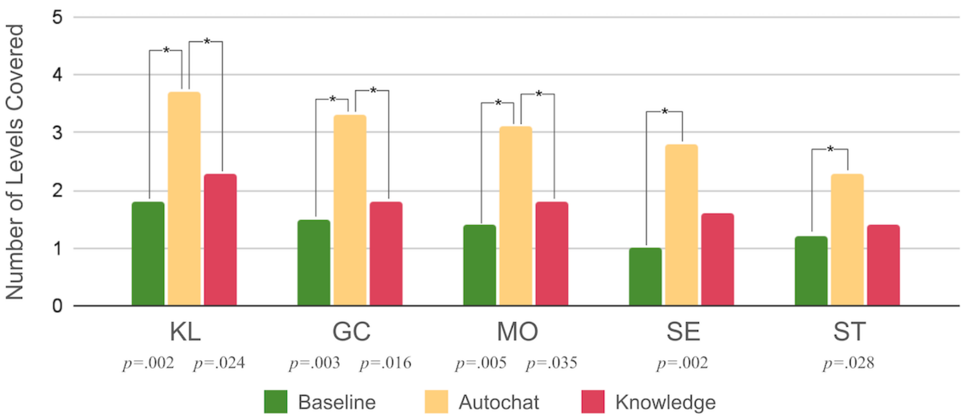

Figure 4.14: The number of levels covered in reported student profiles, in the order of knowledge level (KL), goal commitment (GC), motivation (MO), self-efficacy (SE), and stress (ST). The asterisk (*) indicates statistical significance (p¡0.05) between conditions.

4.6 Discussion

We revisit our research questions briefly and discuss how TeachTune contributes to augmenting the PCA design process.

Trait	Baseline	Autochat	Knowledge
Knowledge coverage	16.5±1.4	17.0±0.9	16.3±1.5
Motivation factor coverage	15.6±1.2	17.4±1.8	16.2±1.0
Psychosocial context coverage	15.4±0.6	16.3±1.5	15.4±0.7
Usability	16.0±0.9	16.2±1.3	16.0±1.1

Table 4.5: The average quality scores of PCAs from each condition. There was no statistical difference among the conditions.

4.6.1 Student Traits for Inclusive Education

Teachers expressed their need to review how PCAs adapt to students' diverse knowledge levels, motivation factors, and psychosocial contextual influence. Prior literature on student traits [197] provided us with extensive dimensions of student traits, and our interview complemented them with teachers' practical priority and concern among them. Our approach may highlight that we might need a more holistic understanding that spans theories, quantitative analysis, and teacher interviews to identify key challenges teachers face and derive effective design goals.

Moreover, although TeachTune satisfied the basic needs for simulating these student traits, teachers wanted additional characteristics to include more diverse student types and teaching scenarios in actual class settings (A5, A8, A10, and K7). These additional needs should not only include the 42 student traits [197] investigated in our formative interview but should also involve the traits of marginalized learners [262, 263]. For instance, students with cognitive disabilities need adaptive delivery of information, and immigrant learners would benefit from culturally friendly examples. Reviewing PCAs before deployment with simulated marginalized students will make classes inclusive and prevent technologies from widening skill gaps [264].

4.6.2 Tolerance for the Alignment Gap

We observed 5% and 10% median alignment gaps between our simulated students and teachers' perceptions (RQ2). This degree of gap could be bearable in the context of simulating conversations because simulated students are primarily designed for teachers to review interactions, not to replicate a particular student precisely, and real students also often show discrepancies in their knowledge states and behaviors by making mistakes and guess answers [265]. Recent research on knowledge tracing suggests that students make more than 10% of slips and guesses in a science examination, and the rate depends on students' proficiency [266]. The individualized rate of slips and guesses per student profile (e.g., increasing the frequency of guesses for a highly motivated simulated student) may improve the believability of simulated students. Teachers will also need interfaces that transparently reveal the state of simulated students (e.g., Fig. 4.4 C) to distinguish system errors from intended slips.

4.6.3 Using Simulated Students for Analysis

Our user study showed that TeachTune helps teachers consider a broader range of students and can help them review their PCAs more robustly before deployment (RQ3). PCA design is an iterative process, and it continues after deploying PCAs to classes. Student profiles and simulated students can support teachers' post-deployment design process by leveraging students' conversation history with PCAs. For instance, teachers can group students by their predefined student profiles as a unit of analysis

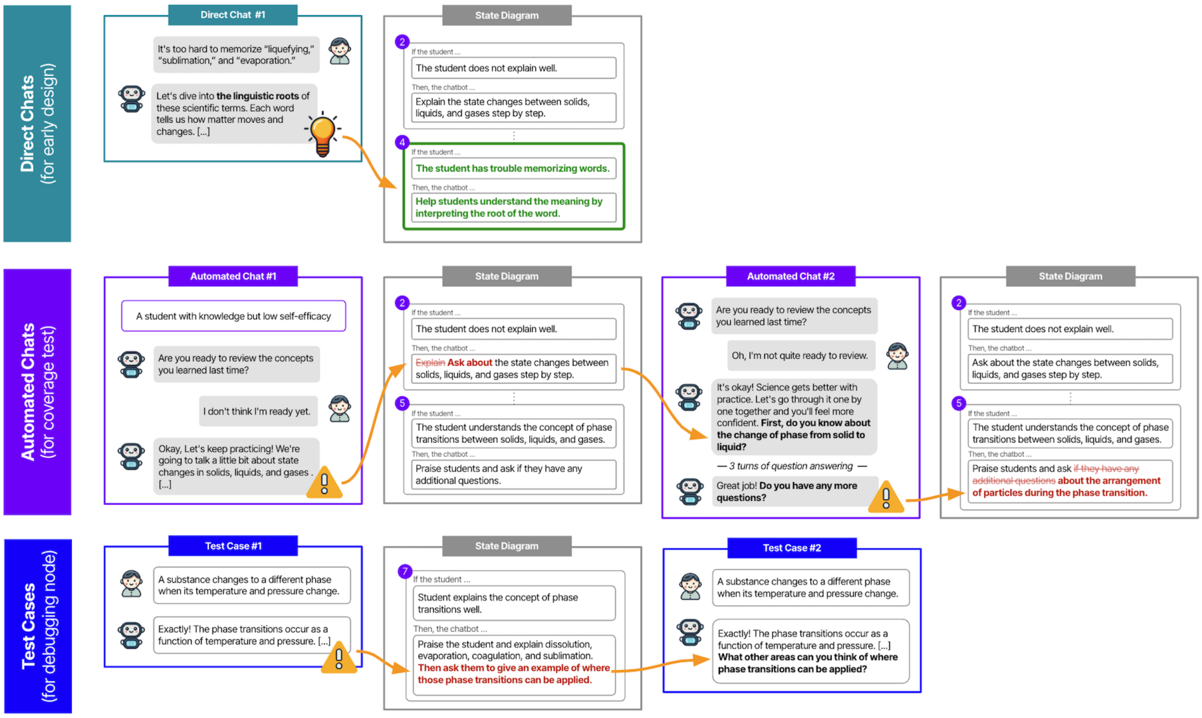

Figure 4.15: Examples of iterative PCA design using each feature. Direct chat: O1 tested a specific question and added new nodes inspired by the PCA's response. Automated chat: O5 identified problems and modified the state diagram. To provide adaptive pedagogy to a low efficacy knowledgeable student, O5 changed the instruction from giving explanations to asking questions. Test cases: O6 modified the specific node to include additional content and used the same test case for re-testing.

and compare learning gain among the groups to identify design issues in PCA. Simulated students can also serve as an interactive analysis tool. Teachers may fine-tune a simulated student with specific student-PCA conversation data and interactively replay (e.g., ask questions to gain deeper insight about the student) previous learning sessions with the simulated agent aligned with a particular student.

4.6.4 Profile-oriented Design Workflow

During formative interviews, we observed that teachers unfamiliar with reviewing PCAs often weave multiple student profiles into a single direct chat. To address the issue, TeachTune proposed a two-step profile-oriented workflow comprising steps for (1) organizing diverse student profiles defined by student traits and (2) observing the test messages generated from these profiles. Our user study showed that this profile-oriented review process could elicit diverse student profiles from teachers and help them explore extensive evaluation spaces. The effectiveness of this two-step workflow lies in its hierarchical structure, which first organizes the evaluation scope at the target user level and then branches into specific scenarios each user might encounter. Such a hierarchical approach can be particularly beneficial for laypeople who try making LLM-infused tools by themselves but are not familiar with reviewing them. For example, when a non-expert develops an LLM application, it will be easier to consider potential user groups than to think of corner cases immediately. The two-step workflow with simulated user groups can scaffold the creator to review the application step by step and generate user scenarios rapidly. We expect that the LLM-assisted profile-oriented design workflow is generalizable to diverse creative tasks, such as UX design [267], service design [268], and video creation [229], that require a profound and extensive understanding of target users.

4.6.5 Risks of Amplifying Stereotypes of Students

Our technical evaluation assumed teachers' expectations of student behaviors as ground truth, considering that simulated students are proxies for automating testing teachers intend. However, in practical classes, there are risks of teachers having stereotypes or TeachTune amplifying their bias toward students over time.

During the observational sessions, we asked teachers' perspectives, and teachers expressed varying levels of concern. O3 commented that private tutors would have limited opportunities to observe their students beyond lessons, making them dependent on simulated behaviors. Conversely, O1 was concerned about her possible stereotypes of student behaviors and relied on automated chat to confirm behaviors she expected. O4 stated that automated chats would not bias teachers as they know the chats are simulated and just a point of reference.

Teachers will need an additional feedback loop to close the gap between their expectations and actual students by deploying PCAs iteratively and monitoring student interaction logs as hypothesis testing. Future work may observe and support how teachers fill or widen the gap at a more longitudinal time scale (e.g., a semester with multiple lessons).

4.7 Limitations and Future Work

We outline the limitations of the work. First, we did not confirm the pedagogical effect of PCAs on students' learning gain and attitude, as we only evaluated the quality of PCAs with experts. We could run lab studies in which middle school students use the PCAs designed by our participants, and we measure their learning gain on phase transitions through a pre-and post-test. Student-involved studies could also reveal the gap between teachers' expectations and students' actual learning; even though a teacher tests a student profile and designs a PCA to cover it, a student of the profile may not find it helpful. Our research focused on investigating the gap between simulated students' behaviors and teachers' expectations. Future work can explore the alignment gap between simulated and actual students and develop interactions to guide teachers in debugging their PCAs and closing the gap. Our preliminary findings will act as a foundational step to move on to safer student-involved studies.

Second, our technical evaluation and user study are limited to a single subject (i.e., science) and learning topic (i.e., phase transitions). Under practical and temporal constraints, we evaluated how Personalized Reflect-Respond generalizes to diverse student profiles and how TeachTune works in a controlled setting as a case study. We expect that our findings will generalize to other STEM fields where knowledge components are well-defined. Still, humanities subjects may require additional support (e.g., simulating students' cultural backgrounds in literature classes). We plan to deploy TeachTune to a programming course at our university and a middle school second language writing class. In the deployment, we will ask the instructors to build PCAs for different roles and contexts, such as homework assistants, teaching assistants, and peer learners. These deployments will concretize our findings in diverse student ages, subjects, and pedagogies.

Lastly, we simulated a limited number of student traits only. Learning is a complex process with complex dynamics between knowledge states, learning traits, cognitive load, and emotion. Our Personalized Reflect-Respond introduced a multifaceted student simulation that involves both knowledge and student traits, but we acknowledge that more personal attributes of students are necessary for authentic simulated students. The attributes can also include interaction-level attributes like delayed responses and facial expressions. Moreover, we assumed student traits to be static throughout conversations, but actual students may change their attitudes with appropriate guidance, and thus, student traits should be as malleable as the knowledge state. We will explore and develop these different designs of student simulation in the future.

Chapter 5. General Discussion

This chapter revisits the fundamental components of the user-driven approach to creating adaptive learning tools and discusses the principles for successful learning tool development.

5.1 Guidance and Expressivity

Across the three systems, hierarchical representations played a central role in enabling collaboration and granting users fine-grained control during tool creation. These representations were effective because they segmented information into discrete, modular units, forming structured interaction patterns that guided users through complex authoring tasks. However, this discretization can pose challenges to user expressivity—particularly in creative processes where users may wish to define, modify, or combine features in ways not anticipated by the system. Rigid hierarchical structures, while supportive of scaffolding understanding and coordination, may inadvertently constrain the fluidity and individuality often needed in tool design.

To address this tension, each system integrated natural language into its hierarchical representations. By representing atomic components—such as subgoals, knowledge elements, or trait attributes—in natural language, users had the expressive flexibility to describe and customize the components in their terms while still operating within an underlying structural scaffold. In the future, we will see more of these hybrid approaches that combine discrete structures with continuous, language-based interactions, allowing systems to offer both guidance and expressivity: structured enough to guide yet open-ended enough to empower.

5.2 Motivating Learners and Teachers

For a user-initiated approach to succeed, it is essential to encourage learners and teachers to actively participate in creating the tools. Across the three systems, two primary motivational mechanisms consistently emerged: (1) reducing the cost and barriers to tool creation and (2) increasing the perceived pedagogical value of creating personalized tools.

In CodeTree, the act of generating subgoal hierarchies through self-explanation not only supported the creation of adaptive code explanations but also significantly improved learners' code-tracing skills. TeachYou leveraged natural language knowledge-state representations and large language models (LLMs) to lower the barrier of authoring teachable agents. TeachTune enabled teachers to create inclusive pedagogical chatbots and explore their effectiveness with simulated students, ultimately reducing teachers' workload while enhancing instructional coverage.

These findings highlight two promising directions for the learner- and teacher-driven creation of educational tools. First, tool creation itself can serve as a pedagogical activity, fostering a metacognitive understanding of learning content and student needs. Constructing tools requires reflection on the underlying cognitive processes they aim to support. Although teachers and learners may initially lack such metacognitive insights, AI scaffolding and collaborative authoring can guide and augment this process—echoing the principles of learnersourcing, where didactic experience is gained through constructive participation [31].

Second, enhancing accessibility in tool creation opens pathways for scalable and personalized learning. As generative technologies become more usable, non-technical educators and students increasingly engage with systems they once considered out of reach. The growing interest among teachers in adopting such systems, as observed in practice [269, 270], underscores this shift. While interactive learning is widely recognized as the most effective mode, its application has traditionally been limited to a few standardized subjects due to the high development cost of domain-specific software.

By empowering teachers and learners to create tools, we open up new opportunities for more metacognitively engaging and pedagogically inclusive learning experiences. Ultimately, I envision a future where educational technology is no longer solely built by experts but co-created by learners and teachers—making education more adaptable, participatory, and democratized.

Chapter 6. Conclusion

This thesis has proposed a user-driven approach to adaptive learning design as a scalable response to the growing diversity of learners and learning contexts in online education. Rather than relying solely on centralized system designers, the proposed approach empowers learners and teachers to actively participate in the creation and customization of adaptive learning tools that align with their specific needs and contexts.

At the core of this approach is the use of hierarchical representations—modular, compositional structures that enable users to articulate, organize, and refine learning content and adaptive behaviors. To explore this approach, three systems were developed: CodeTree, TeachYou, and TeachTune. Each system leverages authoring interfaces built upon hierarchical representations, enabling learners and teachers to contribute meaningfully to the authoring process without requiring extensive technical expertise. Through the use of subgoal hierarchies, knowledge states, and student profiles, these systems demonstrated how learners and teachers could collaboratively create adaptive explanations, configure teachable agents, and iteratively refine pedagogical chatbots.

Ultimately, this thesis argues that empowering learners and teachers as active co-designers of adaptive learning tools offers a promising path toward more personalized, responsive, and inclusive educational technologies—ones that better reflect the diverse and evolving realities of modern education.

Bibliography

[1] Michelene TH Chi and Ruth Wylie. The icap framework: Linking cognitive engagement to active learning outcomes. *Educational psychologist*, 49(4):219–243, 2014.

[2] Noboru Matsuda. 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):48–75, 2022.

[3] Hyoungwook Jin and Juho Kim. Codetree: A system for learnersourcing subgoal hierarchies in code examples. volume 8, New York, NY, USA, April 2024. Association for Computing Machinery.

[4] Yuhao Wu, Shaowei Wang, Cor-Paul Bezemer, and Katsuro Inoue. How do developers utilize source code from stack overflow? *Empirical Software Engineering*, 24(2):637–673, 2019.

[5] Graham Cooper and John Sweller. Effects of schema acquisition and rule automation on mathematical problem-solving transfer. *Journal of educational psychology*, 79(4):347, 1987.

[6] Bat-Sheva Eylon and F. Reif. Effects of knowledge organization on task performance. *Cognition and Instruction*, 1(1):5–44, 1984.

[7] Richard Catrambone. Improving examples to improve transfer to novel problems. *Memory & cognition*, 22(5):606–615, 1994.

[8] Richard Catrambone. Aiding subgoal learning: Effects on transfer. *Journal of educational psychology*, 87(1):5, 1995.

[9] Lauren E. Margulieux, Mark Guzdial, and Richard Catrambone. Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, ICER '12, page 71–78, New York, NY, USA, 2012. Association for Computing Machinery.

[10] Briana B. Morrison, Lauren E. Margulieux, and Mark Guzdial. Subgoals, context, and worked examples in learning computing problem solving. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ICER '15, page 21–29, New York, NY, USA, 2015. Association for Computing Machinery.

[11] Lauren E Margulieux and Richard Catrambone. Finding the best types of guidance for constructing self-explanations of subgoals in programming. *Journal of the Learning Sciences*, 28(1):108–151, 2019.

[12] Kabdo Choi, Hyungyu Shin, Meng Xia, and Juho Kim. Algosolve: Supporting subgoal learning in algorithmic problem-solving with learnersourced microtasks. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery.

[13] Hyoungwook Jin, Minsuk Chang, and Juho Kim. 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, CHI EA '19, page 1–6, New York, NY, USA, 2019. Association for Computing Machinery.

[14] Richard Catrambone. Task analysis by problem solving (taps): Uncovering expert knowledge to develop high-quality instructional materials and training. In *Learning and Technology Symposium, Columbus, GA*, 2011.

[15] Paulo Blikstein, Marcelo Worsley, Chris Piech, Mehran Sahami, Steven Cooper, and Daphne Koller. Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *Journal of the Learning Sciences*, 23(4):561–599, 2014.

[16] Chris Piech, Mehran Sahami, Jonathan Huang, and Leonidas Guibas. Autonomously generating hints by inferring problem solving policies. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*, L@S '15, page 195–204, New York, NY, USA, 2015. Association for Computing Machinery.

[17] Samiha Marwan, Yang Shi, Ian Menezes, Min Chi, Tiffany Barnes, and Thomas W Price. Just a few expert constraints can help: Humanizing data-driven subgoal detection for novice programming. *International Educational Data Mining Society*, 2021.

[18] Ohad Barzilay, Christoph Treude, and Alexey Zagalsky. Facilitating crowd sourced software engineering via stack overflow. In *Finding Source Code on the Web for Remix and Reuse*, pages 289–308. Springer, 2013.

[19] Sarah Weir, Juho Kim, Krzysztof Z. Gajos, and Robert C. Miller. Learnersourcing subgoal labels for how-to videos. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '15, page 405–416, New York, NY, USA, 2015. Association for Computing Machinery.

[20] Anjali Singh, Christopher Brooks, and Shayan Doroudi. Learnersourcing in theory and practice: Synthesizing the literature and charting the future. In *Proceedings of the Ninth ACM Conference on Learning @ Scale*, L@S '22, page 234–245, New York, NY, USA, 2022. Association for Computing Machinery.

[21] Richard Catrambone and Keith J Holyoak. Learning subgoals and methods for solving probability problems. *Memory & Cognition*, 18(6):593–603, 1990.

[22] Richard Catrambone. The subgoal learning model: Creating better examples so that students can solve novel problems. *Journal of experimental psychology: General*, 127(4):355, 1998.

[23] Richard Catrambone. Generalizing solution procedures learned from examples. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 22(4):1020, 1996.

[24] Lauren E Margulieux and Richard Catrambone. Improving problem solving with subgoal labels in expository text and worked examples. *Learning and Instruction*, 42:58–71, 2016.

[25] Lauren E Margulieux, Briana B Morrison, and Adrienne Decker. Reducing withdrawal and failure rates in introductory programming with subgoal labeled worked examples. *International Journal of STEM Education*, 7(1):1–16, 2020.

[26] Lauren E Margulieux, Briana B Morrison, Baker Franke, and Harivololona Ramilison. Effect of implementing subgoals in code. org's intro to programming unit in computer science principles. *ACM Transactions on Computing Education (TOCE)*, 20(4):1–24, 2020.

[27] Lauren Margulieux and Richard Catrambone. Using learners' self-explanations of subgoals to guide initial problem solving in app inventor. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ICER '17, page 21–29, New York, NY, USA, 2017. Association for Computing Machinery.

[28] Barbara J. Ericson, Lauren E. Margulieux, and Jochen Rick. Solving parsons problems versus fixing and writing code. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*, Koli Calling '17, page 20–29, New York, NY, USA, 2017. Association for Computing Machinery.

[29] Michelene TH Chi. Active-constructive-interactive: A conceptual framework for differentiating learning activities. *Topics in cognitive science*, 1(1):73–105, 2009.

[30] Briana B Morrison, Lauren E Margulieux, and Adrienne Decker. The curious case of loops. *Computer Science Education*, 30(2):127–154, 2020.

[31] Juho Kim et al. *Learnersourcing: improving learning with collective learner activity.* PhD thesis, Massachusetts Institute of Technology, 2015.

[32] Juho Kim, Phu Tran Nguyen, Sarah Weir, Philip J. Guo, Robert C. Miller, and Krzysztof Z. Gajos. Crowdsourcing step-by-step information extraction to enhance existing how-to videos. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, page 4017–4026, New York, NY, USA, 2014. Association for Computing Machinery.

[33] Solmaz Abdi, Hassan Khosravi, Shazia Sadiq, and Gianluca Demartini. Evaluating the quality of learning resources: A learnersourcing approach. *IEEE Transactions on Learning Technologies*, 14(1):81–92, 2021.

[34] Lin Ni, Qiming Bao, Xiaoxuan Li, Qianqian Qi, Paul Denny, Jim Warren, Michael Witbrock, and Jiamou Liu. Deepqr: Neural-based quality ratings for learnersourced multiple-choice questions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 12826–12834, 2022.

[35] Jonathan Huang, Chris Piech, Andy Nguyen, and Leonidas Guibas. Syntactic and functional variability of a million code submissions in a machine learning mooc. In *AIED 2013 Workshops Proceedings Volume*, volume 25. Citeseer, 2013.

[36] Juho Kim, Philip J. Guo, Carrie J. Cai, Shang-Wen (Daniel) Li, Krzysztof Z. Gajos, and Robert C. Miller. Data-driven interaction techniques for improving navigation of educational videos. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, page 563–572, New York, NY, USA, 2014. Association for Computing Machinery.

[37] Xu Wang, Srinivasa Teja Talluri, Carolyn Rose, and Kenneth Koedinger. Upgrade: Sourcing student open-ended solutions to create scalable learning opportunities. In *Proceedings of the Sixth (2019) ACM Conference on Learning @ Scale*, L@S '19, New York, NY, USA, 2019. Association for Computing Machinery.

[38] Piotr Mitros. Learnersourcing of complex assessments. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*, L@S '15, page 317–320, New York, NY, USA, 2015. Association for Computing Machinery.

[39] Elena L. Glassman, Aaron Lin, Carrie J. Cai, and Robert C. Miller. Learnersourcing personalized hints. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, CSCW '16, page 1626–1636, New York, NY, USA, 2016. Association for Computing Machinery.

[40] Joseph Jay Williams, Juho Kim, Anna Rafferty, Samuel Maldonado, Krzysztof Z. Gajos, Walter S. Lasecki, and Neil Heffernan. Axis: Generating explanations at scale with learnersourcing and machine learning. In *Proceedings of the Third (2016) ACM Conference on Learning @ Scale*, L@S '16, page 379–388, New York, NY, USA, 2016. Association for Computing Machinery.

[41] Iman Yeckehzaare, Tirdad Barghi, and Paul Resnick. Qmaps: Engaging students in voluntary question generation and linking. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–14, New York, NY, USA, 2020. Association for Computing Machinery.

[42] Alexander Renkl. Learning from worked-out examples: A study on individual differences. *Cognitive science*, 21(1):1–29, 1997.

[43] Adrienne Decker, Lauren E. Margulieux, and Briana B. Morrison. Using the solo taxonomy to understand subgoal labels effect in cs1. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, ICER '19, page 209–217, New York, NY, USA, 2019. Association for Computing Machinery.

[44] Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell, and Katrina Panovich. Soylent: A word processor with a crowd inside. volume 58, page 85–94, New York, NY, USA, jul 2015. Association for Computing Machinery.

[45] Ching Liu, Juho Kim, and Hao-Chuan Wang. Conceptscape: Collaborative concept mapping for video learning. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–12, New York, NY, USA, 2018. Association for Computing Machinery.

[46] Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. Brat: A web-based tool for nlp-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '12, page 102–107, USA, 2012. Association for Computational Linguistics.

[47] Philipp Helfrich, Elias Rieb, Giuseppe Abrami, Andy Lücking, and Alexander Mehler. TreeAnnotator: Versatile visual annotation of hierarchical text relations. In Nicoletta Calzolari, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Koiti Hasida, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, Stelios Piperidis, and Takenobu Tokunaga, editors, *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA).

[48] Norman J Slamecka and Peter Graf. The generation effect: Delineation of a phenomenon. *Journal of experimental Psychology: Human learning and Memory*, 4(6):592, 1978.

[49] Yisong Yue, Josef Broder, Robert Kleinberg, and Thorsten Joachims. The k-armed dueling bandits problem. *Journal of Computer and System Sciences*, 78(5):1538–1556, 2012.

[50] Paul Ginns. Integrating information: A meta-analysis of the spatial contiguity and temporal contiguity effects. *Learning and instruction*, 16(6):511–525, 2006.

[51] Mareike Florax and Rolf Ploetzner. What contributes to the split-attention effect? the role of text segmentation, picture labelling, and spatial proximity. *Learning and instruction*, 20(3):216–224, 2010.

[52] Joon-Ho Lim, Yongjin Bae, Hyunki Kim, Yunjeong Kim, and Kyu-Chul Lee. Korean dependency guidelines for dependency parsing and exo-brain language analysis corpus. In *Annual Conference on Human and Language Technology*, pages 234–239. Human and Language Technology, 2015.

[53] Thorvald A Sorensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on danish commons. *Biol. Skar.*, 5:1–34, 1948.

[54] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[55] Stephen MacNeil, Andrew Tran, Dan Mogil, Seth Bernstein, Erin Ross, and Ziheng Huang. Generating diverse code explanations using the gpt-3 large language model. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 2*, ICER '22, page 37–39, New York, NY, USA, 2022. Association for Computing Machinery.

[56] Edward H Simpson. Measurement of diversity. *nature*, 163(4148):688–688, 1949.

[57] Briana B. Morrison, Brian Dorn, and Mark Guzdial. Measuring cognitive load in introductory cs: Adaptation of an instrument. In *Proceedings of the Tenth Annual Conference on International Computing Education Research*, ICER '14, page 131–138, New York, NY, USA, 2014. Association for Computing Machinery.

[58] Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the Fourth International Workshop on Computing Education Research*, ICER '08, page 101–112, New York, NY, USA, 2008. Association for Computing Machinery.

[59] Lydia B. Chilton, Greg Little, Darren Edge, Daniel S. Weld, and James A. Landay. Cascade: Crowdsourcing taxonomy creation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, page 1999–2008, New York, NY, USA, 2013. Association for Computing Machinery.

[60] Yuyin Sun, Adish Singla, Dieter Fox, and Andreas Krause. Building hierarchies of concepts via crowdsourcing. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, page 844–851. AAAI Press, 2015.

[61] Anjali Singh, Christopher Brooks, Yiwen Lin, and Warren Li. What's in it for the learners? evidence from a randomized field experiment on learnersourcing questions in a mooc. In *Proceedings*

of the Eighth ACM Conference on Learning @ Scale, L@S '21, page 221–233, New York, NY, USA, 2021. Association for Computing Machinery.

[62] Xingquan Zhu, Ahmed K Elmagarmid, Xiangyang Xue, Lide Wu, and Ann Christine Catlin. Insightvideo: toward hierarchical video content organization for efficient browsing, summarization and retrieval. *IEEE Transactions on Multimedia*, 7(4):648–666, 2005.

[63] Megha Nawhal, Jacqueline B. Lang, Greg Mori, and Parmit K. Chilana. Videowhiz: Non-linear interactive overviews for recipe videos. In *Proceedings of the 45th Graphics Interface Conference on Proceedings of Graphics Interface 2019*, GI'19, Waterloo, CAN, 2019. Canadian Human-Computer Communications Society.

[64] Anh Truong, Peggy Chi, David Salesin, Irfan Essa, and Maneesh Agrawala. Automatic generation of two-level hierarchical tutorials from instructional makeup videos. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[65] Paul Denny, Sami Sarsa, Arto Hellas, and Juho Leinonen. Robosourcing educational resources–leveraging large language models for learnersourcing. *arXiv preprint arXiv:2211.04715*, 2022.

[66] Maha Al-Yahya. Ontology-based multiple choice question generation. *The Scientific World Journal*, 2014, 2014.

[67] Ellampallil Venugopal Vinu et al. A novel approach to generate mcqs from domain ontology: Considering dl semantics and open-world assumption. *Journal of Web Semantics*, 34:40–54, 2015.

[68] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. What makes a good code example?: A study of programming q&a in stackoverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 25–34. IEEE, 2012.

[69] Lauren E. Margulieux, Briana B. Morrison, and Adrienne Decker. Design and pilot testing of subgoal labeled worked examples for five core concepts in cs1. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '19, page 548–554, New York, NY, USA, 2019. Association for Computing Machinery.

[70] Hyoungwook Jin, Seonghee Lee, Hyungyu Shin, and Juho Kim. 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, New York, NY, USA, 2024. Association for Computing Machinery.

[71] Muhsin Menekse, Glenda Stump, Stephen Krause, and Michelene T.H.Chi. Differentiated overt learning activities for effective instruction in engineering classrooms. *Journal of Engineering Education*, 102:346–374, 07 2013.

[72] Jonathan Tudge. *Vygotsky, the zone of proximal development, and peer collaboration: Implications for classroom practice*, page 155–172. Cambridge University Press, 1990.

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

[74] Pavel Smutný and Petra Schreiberova. Chatbots for learning: A review of educational chatbots for the facebook messenger. *Computers & Education*, 151:103862, 02 2020.

[75] Doris Chin, I.M. Dohmen, and D.L. Schwartz. Young children can learn scientific reasoning with teachable agents. *Learning Technologies, IEEE Transactions on*, 6:248–257, 07 2013.

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

[77] Sangho Suh and Pengcheng An. Leveraging generative conversational ai to develop a creative learning environment for computational thinking. In *27th International Conference on Intelligent User Interfaces*, IUI '22 Companion, page 73–76, New York, NY, USA, 2022. Association for Computing Machinery.

[78] Philip J. Guo. Online python tutor: Embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, page 579–584, New York, NY, USA, 2013. Association for Computing Machinery.

[79] Catherine C. Chase, Doris B. Chin, Marily A Oppezzo, and Daniel L. Schwartz. Teachable agents and the protégé effect: Increasing the effort towards learning. *Journal of Science Education and Technology*, 18:334–352, 2009.

[80] Amy Debbané, Ken Jen Lee, Jarvis Tse, and Edith Law. Learning by teaching: Key challenges and design implications. *Proc. ACM Hum.-Comput. Interact.*, 7(CSCW1), apr 2023.

[81] Tasmia Shahriar and Noboru Matsuda. "can you clarify what you said?": Studying the impact of tutee agents' follow-up questions on tutors' learning. In Ido Roll, Danielle McNamara, Sergey Sosnovsky, Rose Luckin, and Vania Dimitrova, editors, *Artificial Intelligence in Education*, pages 395–407, Cham, 2021. Springer International Publishing.

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

[83] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz. 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*, IUI '23, page 491–514, New York, NY, USA, 2023. Association for Computing Machinery.

[84] Julia M Markel, Steven G Opferman, James A Landay, and Chris Piech. Gpteach: Interactive ta training with gpt-based students, Feb 2023.

[85] Aobo Kong, Shiwan Zhao, Hao Chen, Qicheng Li, Yong Qin, Ruiqi Sun, and Xin Zhou. Better zero-shot reasoning with role-play prompting. *arXiv preprint arXiv:2308.07702*, 2023.

[86] 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. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

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

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

[89] Krittaya Leelawong and Gautam Biswas. Designing learning by teaching agents: The betty's brain system. *Int. J. Artif. Intell. Ed.*, 18(3):181–208, aug 2008.

[90] Noboru Matsuda, Victoria Keiser, Rohan Raizada, Arthur Tu, Gabriel Stylianides, William W. Cohen, and Kenneth R. Koedinger. Learning by teaching simstudent: Technical accomplishments and an initial use with students. In Vincent Aleven, Judy Kay, and Jack Mostow, editors, *Intelligent Tutoring Systems*, pages 317–326, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[91] Lena Pareto, Tobias Arvemo, Ylva Dahl, Magnus Haake, and Agneta Gulz. A teachable-agent arithmetic game's effects on mathematics understanding, attitude and self-efficacy. In Gautam Biswas, Susan Bull, Judy Kay, and Antonija Mitrovic, editors, *Artificial Intelligence in Education*, pages 247–255, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

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

[93] Fu-Yun Yu. 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*, page 564–571, NLD, 2005. IOS Press.

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

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

[96] R.D. Roscoe and M.T.H. Chi. Understanding tutor learning: Knowledge building and knowledge telling in peer tutors' explanations and questions. *Review of Educational Research*, 77:534–574, 2007.

[97] David Duran. Learning-by-teaching. evidence and implications as a pedagogical mechanism. *Innovations in Education and Teaching International*, 54(5):476–484, 2017.

[98] Marlene Scardamalia and Carl Bereiter. *Knowledge building: Theory, pedagogy, and technology*, pages 97–. 01 2006.

[99] Noreen M. Webb, Philip Ender, and Scott Lewis. Problem-solving strategies and group processes in small groups learning computer programming. *American Educational Research Journal*, 23(2):243–261, 1986.

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

[101] A. King, A. Staffieri, and A. Adelgais. Mutual peer tutoring: Effects of structuring tutorial interaction to scaffold peer learning. *Journal of Educational Psychology*, 90(1):134–152, 1998.

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

[103] Noboru Matsuda, Vishnu Priya Chandra Sekar, and Natalie Wall. Metacognitive scaffolding amplifies the effect of learning by teaching a teachable agent. In Carolyn Penstein Rosé, Roberto Martínez-Maldonado, H. Ulrich Hoppe, Rose Luckin, Manolis Mavrikis, Kaska Porayska-Pomsta, Bruce McLaren, and Benedict du Boulay, editors, *Artificial Intelligence in Education*, pages 311–323, Cham, 2018. Springer International Publishing.

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

[105] Arthur Graesser, Shulan Lu, G. Jackson, Heather Mitchell, Mathew Ventura, Andrew Olney, and Max Louwerse. Autotutor: a tutor with dialogue in natural language. *Behavior Research Methods*, 36:180–192, 06 2004.

[106] Noboru Matsuda, Evelyn Yarzebinski, Victoria Keiser, Rohan Raizada, Gabriel J. Stylianides, William W. Cohen, and Kenneth R. Koedinger. 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*, AIED'11, page 213–221, Berlin, Heidelberg, 2011. Springer-Verlag.

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

[108] Bert Bredeweg, Anders Bouwer, Jelmer Jellema, Dirk Bertels, Floris Floris Linnebank, and Jochem Liem. Garp3: a new workbench for qualitative reasoning and modelling. In *Proceedings of the 4th International Conference on Knowledge Capture*, K-CAP '07, page 183–184, New York, NY, USA, 2007. Association for Computing Machinery.

[109] Gautam Biswas, Krittaya Leelawong, Daniel Schwartz, Nancy Vye, and The Teachable Agents Group at Vanderbilt. Learning by teaching: A new agent paradigm for educational software. *Applied Artificial Intelligence*, 19(3-4):363–392, 2005.

[110] Noboru Matsuda. 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.

[111] Harri Ketamo. Semantic networks -based teachable agents in an educational game. *W. Trans. on Comp.*, 8(4):641–650, apr 2009.

[112] Noboru Matsuda, William W. Cohen, Kenneth R. Koedinger, Victoria Keiser, Rohan Raizada, Evelyn Yarzebinski, Shayna P. Watson, and Gabriel Stylianides. 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*, pages 25–32, 2012.

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

[114] Rose E. Wang and Dorottya Demszky. Is chatgpt a good teacher coach? measuring zero-shot performance for scoring and providing actionable insights on classroom instruction, 2023.

[115] Santiago Ojeda-Ramirez, Sina Rismanchian, and Shayan Doroudi. Learning about ai to learn about learning: Artificial intelligence as a tool for metacognitive reflection. 2023.

[116] Wangchunshu Zhou, Yuchen Eleanor Jiang, Peng Cui, Tiannan Wang, Zhenxin Xiao, Yifan Hou, Ryan Cotterell, and Mrinmaya Sachan. Recurrentgpt: Interactive generation of (arbitrarily) long text. *arXiv preprint arXiv:2305.13304*, 2023.

[117] Charles Packer, Vivian Fang, Shishir G Patil, Kevin Lin, Sarah Wooders, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

[118] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, New York, NY, USA, 2023. Association for Computing Machinery.

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

[120] Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

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

[122] Zhipeng Chen, Kun Zhou, Beichen Zhang, Zheng Gong, Xin Zhao, and Ji-Rong Wen. ChatCoT: Tool-augmented chain-of-thought reasoning on chat-based large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14777–14790, Singapore, December 2023. Association for Computational Linguistics.

[123] Paul Denny, Andrew Luxton-Reilly, and Beth Simon. Evaluating a new exam question: Parsons problems. In *Proceedings of the Fourth International Workshop on Computing Education Research*, ICER '08, page 113–124, New York, NY, USA, 2008. Association for Computing Machinery.

[124] OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

[125] Erin Walker, Nikol Rummel, and Kenneth R. Koedinger. To tutor the tutor: Adaptive domain support for peer tutoring. In Beverley P. Woolf, Esma Aïmeur, Roger Nkambou, and Susanne Lajoie, editors, *Intelligent Tutoring Systems*, pages 626–635, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

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

[127] Changyoon Lee, Yeon Seonwoo, and Alice Oh. CS1QA: A dataset for assisting code-based question answering in an introductory programming course. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz, editors, *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2026–2040, Seattle, United States, July 2022. Association for Computational Linguistics.

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

[129] Saelyne Yang, Sangkyung Kwak, Juhoon Lee, and Juho Kim. Beyond instructions: A taxonomy of information types in how-to videos. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery.

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

[131] Benjamin S Bloom. Learning for mastery. instruction and curriculum. regional education laboratory for the carolinas and virginia, topical papers and reprints, number 1. *Evaluation comment*, 1(2):n2, 1968.

[132] David R. Krathwohl. A revision of bloom's taxonomy: An overview. *Theory Into Practice*, 41(4):212–218, 2002.

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

[134] Tongshuang Wu, Michael Terry, and Carrie Jun Cai. 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*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery.

[135] 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. Llama: Open and efficient foundation language models. *ArXiv*, abs/2302.13971, 2023.

[136] Li Zhang, Liam Dugan, Hainiu Xu, and Chris Callison-burch. Exploring the curious case of code prompts. In Bhavana Dalvi Mishra, Greg Durrett, Peter Jansen, Danilo Neves Ribeiro, and Jason Wei, editors, *Proceedings of the 1st Workshop on Natural Language Reasoning and Structured Explanations (NLRSE)*, pages 9–17, Toronto, Canada, June 2023. Association for Computational Linguistics.

[137] Leo Gao. Shapley Value Attribution in Chain of Thought, 2023.

[138] Jieyi Long. Large language model guided tree-of-thought, 2023.

[139] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *ArXiv*, abs/2305.10601, 2023.

[140] Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. The lru-k page replacement algorithm for database disk buffering. *SIGMOD Rec.*, 22(2):297–306, jun 1993.

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

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

[143] Alison King. Ask to think-tel why: A model of transactive peer tutoring for scaffolding higher level complex learning. *Educational Psychologist*, 32(4):221–235, 1997.

[144] Alison King. Guiding knowledge construction in the classroom: Effects of teaching children how to question and how to explain. *American Educational Research Journal*, 31(2):338–368, 1994.

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

[146] Joshua Robinson, Christopher Michael Rytting, and David Wingate. Leveraging large language models for multiple choice question answering. *ArXiv*, abs/2210.12353, 2022.

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

[148] 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. Webgpt: Browser-assisted question-answering with human feedback, 2022.

[149] Hyoungwook Jin, Minsuk Chang, and Juho Kim. 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*, CHI EA '19, page 1–6, New York, NY, USA, 2019. Association for Computing Machinery.

[150] Elena L. Glassman, Aaron Lin, Carrie J. Cai, and Robert C. Miller. Learnersourcing personalized hints. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, CSCW '16, page 1626–1636, New York, NY, USA, 2016. Association for Computing Machinery.

[151] Roi Cohen, Eden Biran, Ori Yoran, Amir Globerson, and Mor Geva. Evaluating the ripple effects of knowledge editing in language models, 2023.

[152] Xiaopeng Li, Shasha Li, Shezheng Song, Jing Yang, Jun Ma, and Jie Yu. Pmet:, 2023.

[153] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. Fast model editing at scale, 2022.

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

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

[156] Tam Nguyen Thanh, Michael Morgan, Matthew Butler, and Kim Marriott. Perfect match: Facilitating study partner matching. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 1102–1108, New York, NY, USA, 2019. Association for Computing Machinery.

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

[158] Kristen N. Moreno, Bianca Klettke, Kiran Nibbaragandla, and Arthur C. Graesser. Perceived characteristics and pedagogical efficacy of animated conversational agents. In Stefano A. Cerri, Guy Gouardères, and Fàbio Paraguaçu, editors, *Intelligent Tutoring Systems*, pages 963–971, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[159] Abdulhadi Shoufan. Exploring students' perceptions of chatgpt: Thematic analysis and follow-up survey. *IEEE Access*, 11:38805–38818, 2023.

[160] 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. Guidelines for human-ai interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery.

[161] Ewa Luger and Abigail Sellen. "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*, CHI '16, page 5286–5297, New York, NY, USA, 2016. Association for Computing Machinery.

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

[163] Sasha Volodin and Sara Moussawi. 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*, SIGMIS-CPR'20, page 60, New York, NY, USA, 2020. Association for Computing Machinery.

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

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

[166] Sherry Ruan, Liwei Jiang, Justin Xu, Bryce Joe-Kun Tham, Zhengneng Qiu, Yeshuang Zhu, Elizabeth L. Murnane, Emma Brunskill, and James A. Landay. Quizbot: A dialogue-based adaptive learning system for factual knowledge. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery.

[167] Hyoungwook Jin, Minju Yoo, Jeongeon Park, Yokyung Lee, Xu Wang, and Juho Kim. Teachtune: Reviewing pedagogical agents against diverse student profiles with simulated students. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25, New York, NY, USA, 2025. Association for Computing Machinery.

[168] Florian Weber, Thiemo Wambsganss, Dominic Rüttimann, and Matthias Söllner. Pedagogical agents for interactive learning: A taxonomy of conversational agents in education. In *ICIS*, Austin, Texas, USA, 2021. International Conference on Information Systems.

[169] Arthur C Graesser, Shulan Lu, George Tanner Jackson, Heather Hite Mitchell, Mathew Ventura, Andrew Olney, and Max M Louwerse. Autotutor: A tutor with dialogue in natural language. *Behavior Research Methods, Instruments, & Computers*, 36:180–192, 2004.

[170] Noboru Matsuda, William W. Cohen, Kenneth R. Koedinger, Victoria Keiser, Rohan Raizada, Evelyn Yarzebinski, Shayna P. Watson, and Gabriel Stylianides. 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*, pages 25–32, 2012.

[171] Mehdi Alaimi, Edith Law, Kevin Daniel Pantasdo, Pierre-Yves Oudeyer, and Hélène Sauzeon. Pedagogical agents for fostering question-asking skills in children. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–13, New York, NY, USA, 2020. Association for Computing Machinery.

[172] Justin Cranshaw, Emad Elwany, Todd Newman, Rafal Kocielnik, Bowen Yu, Sandeep Soni, Jaime Teevan, and Andrés Monroy-Hernández. Calendar.help: Designing a workflow-based scheduling agent with humans in the loop. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, page 2382–2393, New York, NY, USA, 2017. Association for Computing Machinery.

[173] Yoonseo Choi, Toni-Jan Keith Palma Monserrat, Jeongeon Park, Hyungyu Shin, Nyoungwoo Lee, and Juho Kim. Protochat: Supporting the conversation design process with crowd feedback. *Proc. ACM Hum.-Comput. Interact.*, 4(CSCW3), jan 2021.

[174] Sumit Kumar Dam, Choong Seon Hong, Yu Qiao, and Chaoning Zhang. A complete survey on llm-based ai chatbots, 2024.

[175] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[176] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate problem solving with large language models, 2023.

[177] Ian Arawjo, Chelse Swoopes, Priyan Vaithilingam, Martin Wattenberg, and Elena L. Glassman. Chainforge: A visual toolkit for prompt engineering and llm hypothesis testing. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery.

[178] Andreas Martin, Charuta Pande, Hans Friedrich Witschel, and Judith Mathez. Chedbot: Designing a domain-specific conversational agent in a simulational learning environment using llms. 3(1):180–187, 2024.

[179] Alexander J. Fiannaca, Chinmay Kulkarni, Carrie J Cai, and Michael Terry. Programming without a programming language: Challenges and opportunities for designing developer tools for prompt programming. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI EA '23, New York, NY, USA, 2023. Association for Computing Machinery.

[180] Tae Soo Kim, Yoonjoo Lee, Minsuk Chang, and Juho Kim. Cells, generators, and lenses: Design framework for object-oriented interaction with large language models. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, New York, NY, USA, 2023. Association for Computing Machinery.

[181] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

[182] Irina Jurenka, Markus Kunesch, Kevin R McKee, Daniel Gillick, Shaojian Zhu, Sara Wiltberger, Shubham Milind Phal, Katherine Hermann, Daniel Kasenberg, Avishkar Bhoopchand, et al. Towards responsible development of generative ai for education: An evaluation-driven approach. *arXiv preprint arXiv:2407.12687*, 2024.

[183] J.D. Zamfirescu-Pereira, Laryn Qi, Bjorn Hartmann, John Denero, and Narges Norouzi. Conversational programming with llm-powered interactive support in an introductory computer science

course. In *Proceedings of the Workshop on Generative AI for Education (GAIED) at NeurIPS 2023*, New Orleans, Louisiana, USA, 2023.

[184] Jieun Han, Haneul Yoo, Yoonsu Kim, Junho Myung, Minsun Kim, Hyunseung Lim, Juho Kim, Tak Yeon Lee, Hwajung Hong, So-Yeon Ahn, and Alice Oh. Recipe: How to integrate chatgpt into efl writing education. In *Proceedings of the Tenth ACM Conference on Learning @ Scale*, L@S '23, page 416–420, New York, NY, USA, 2023. Association for Computing Machinery.

[185] Robin Schmucker, Meng Xia, Amos Azaria, and Tom Mitchell. Ruffle&riley: From lesson text to conversational tutoring. In *Proceedings of the Eleventh ACM Conference on Learning @ Scale*, L@S '24, page 547–549, New York, NY, USA, 2024. Association for Computing Machinery.

[186] Jiawen Liu, Yuanyuan Yao, Pengcheng An, and Qi Wang. Peergpt: Probing the roles of llm-based peer agents as team moderators and participants in children's collaborative learning. In *Extended Abstracts of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI EA '24, New York, NY, USA, 2024. Association for Computing Machinery.

[187] Rongxuan Wei, Kangkang Li, and Jiaming Lan. Improving collaborative learning performance based on llm virtual assistant. In *2024 13th International Conference on Educational and Information Technology (ICEIT)*, pages 1–6, 2024.

[188] Ha Nguyen, Victoria Nguyen, Saríah López-Fierro, Sara Ludovise, and Rossella Santagata. Simulating climate change discussion with large language models: Considerations for science communication at scale. In *Proceedings of the Eleventh ACM Conference on Learning @ Scale*, L@S '24, page 28–38, New York, NY, USA, 2024. Association for Computing Machinery.

[189] Arianna Johnson. Chatgpt in schools: Here's where it's banned—and how it could potentially help students. *Forbes*, 2023.

[190] Lingyao Li, Zihui Ma, Lizhou Fan, Sanggyu Lee, Huizi Yu, and Libby Hemphill. Chatgpt in education: A discourse analysis of worries and concerns on social media. *Education and Information Technologies*, 29(9):10729–10762, 2024.

[191] Abdulhadi Shoufan. Exploring students' perceptions of chatgpt: Thematic analysis and follow-up survey. *IEEE Access*, 11:38805–38818, 2023.

[192] Hyanghee Park and Daehwan Ahn. The promise and peril of chatgpt in higher education: Opportunities, challenges, and design implications. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery.

[193] Nicole Jakubczyk Oster Melissa Warr and Roger Isaac. Implicit bias in large language models: Experimental proof and implications for education. *Journal of Research on Technology in Education*, 0(0):1–24, 2024.

[194] Thiemo Wambsganss, Xiaotian Su, Vinitra Swamy, Seyed Neshaei, Roman Rietsche, and Tanja Käser. Unraveling downstream gender bias from large language models: A study on AI educational writing assistance. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10275–10288, Singapore, December 2023. Association for Computational Linguistics.

[195] Yogesh K Dwivedi, Nir Kshetri, Laurie Hughes, Emma Louise Slade, Anand Jeyaraj, Arpan Kumar Kar, Abdullah M Baabdullah, Alex Koohang, Vishnupriya Raghavan, Manju Ahuja, et al. Opinion paper: "so what if chatgpt wrote it?" multidisciplinary perspectives on opportunities, challenges and implications of generative conversational ai for research, practice and policy. *International Journal of Information Management*, 71:102642, 2023.

[196] Fatemeh Mosaiyebzadeh, Seyedamin Pouriyeh, Reza Parizi, Nasrin Dehbozorgi, Mohsen Dorodchi, and Daniel Macêdo Batista. Exploring the role of chatgpt in education: Applications and challenges. In *Proceedings of the 24th Annual Conference on Information Technology Education*, SIGITE '23, page 84–89, New York, NY, USA, 2023. Association for Computing Machinery.

[197] Michelle Richardson, Charles Abraham, and Rod Bond. Psychological correlates of university students' academic performance: a systematic review and meta-analysis. *Psychological bulletin*, 138(2):353, 2012.

[198] Benjamin S Bloom. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational researcher*, 13(6):4–16, 1984.

[199] Xiaoyi Tian, Zak Risha, Ishrat Ahmed, Arun Balajiee Lekshmi Narayanan, and Jacob Biehl. Let's talk it out: A chatbot for effective study habit behavioral change. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW1):1–32, 2021.

[200] Juanan Pereira. Leveraging chatbots to improve self-guided learning through conversational quizzes. In *Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality*, TEEM '16, page 911–918, New York, NY, USA, 2016. Association for Computing Machinery.

[201] Juanan Pereira and Óscar Díaz. Struggling to keep tabs on capstone projects: a chatbot to tackle student procrastination. *ACM Transactions on Computing Education (TOCE)*, 22(1):1–22, 2021.

[202] Bahar Memarian and Tenzin Doleck. Fairness, accountability, transparency, and ethics (fate) in artificial intelligence (ai), and higher education: A systematic review. *Computers and Education: Artificial Intelligence*, page 100152, 2023.

[203] Savvas Petridis, Benjamin D Wedin, James Wexler, Mahima Pushkarna, Aaron Donsbach, Nitesh Goyal, Carrie J Cai, and Michael Terry. Constitutionmaker: Interactively critiquing large language models by converting feedback into principles. In *Proceedings of the 29th International Conference on Intelligent User Interfaces*, IUI '24, page 853–868, New York, NY, USA, 2024. Association for Computing Machinery.

[204] Michael A. Hedderich, Natalie N. Bazarova, Wenting Zou, Ryun Shim, Xinda Ma, and Qian Yang. A piece of theatre: Investigating how teachers design llm chatbots to assist adolescent cyberbullying education. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery.

[205] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with gpt-4, 2023.

[206] Marco Tulio Ribeiro and Scott Lundberg. Adaptive testing and debugging of NLP models. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3253–3267, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[207] Sherry Wu, Hua Shen, Daniel S Weld, Jeffrey Heer, and Marco Tulio Ribeiro. Scattershot: Interactive in-context example curation for text transformation. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*, IUI '23, page 353–367, New York, NY, USA, 2023. Association for Computing Machinery.

[208] J.D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. Why johnny can't prompt: How non-ai experts try (and fail) to design llm prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery.

[209] Ángel Alexander Cabrera, Erica Fu, Donald Bertucci, Kenneth Holstein, Ameet Talwalkar, Jason I. Hong, and Adam Perer. Zeno: An interactive framework for behavioral evaluation of machine learning. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery.

[210] Tae Soo Kim, Yoonjoo Lee, Jamin Shin, Young-Ho Kim, and Juho Kim. Evallm: Interactive evaluation of large language model prompts on user-defined criteria. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery.

[211] Yoonjoo Lee, Tae Soo Kim, Sungdong Kim, Yohan Yun, and Juho Kim. Dapie: Interactive step-by-step explanatory dialogues to answer children's why and how questions. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery.

[212] Tasmia Shahriar and Noboru Matsuda. "can you clarify what you said?": Studying the impact of tutee agents' follow-up questions on tutors' learning. In *Artificial Intelligence in Education: 22nd International Conference, AIED 2021, Utrecht, The Netherlands, June 14–18, 2021, Proceedings, Part I 22*, pages 395–407, Cham, 2021. Springer International Publishing.

[213] Jiale Li, Jiayang Li, Jiahao Chen, Yifan Li, Shijie Wang, Hugo Zhou, Minjun Ye, and Yunsheng Su. Evolving agents: Interactive simulation of dynamic and diverse human personalities, 2024.

[214] Xinyi Lu and Xu Wang. Generative students: Using llm-simulated student profiles to support question item evaluation. In *Proceedings of the Eleventh ACM Conference on Learning @ Scale*, L@S '24, page 16–27, New York, NY, USA, 2024. Association for Computing Machinery.

[215] Scott R. Klemmer, Anoop K. Sinha, Jack Chen, James A. Landay, Nadeem Aboobaker, and Annie Wang. Suede: a wizard of oz prototyping tool for speech user interfaces. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, UIST '00, page 1–10, New York, NY, USA, 2000. Association for Computing Machinery.

[216] Zhifa Chen, Yichen Lu, Mika P. Nieminen, and Andrés Lucero. Creating a chatbot for and with migrants: Chatbot personality drives co-design activities. In *Proceedings of the 2020 ACM Designing*

Interactive Systems Conference, DIS '20, page 219–230, New York, NY, USA, 2020. Association for Computing Machinery.

[217] Eva Durall Gazulla, Ludmila Martins, and Maite Fernández-Ferrer. Designing learning technology collaboratively: Analysis of a chatbot co-design. *Education and Information Technologies*, 28(1):109–134, 2023.

[218] Courtney Potts, Edel Ennis, RB Bond, MD Mulvenna, Michael F McTear, Kyle Boyd, Thomas Broderick, Martin Malcolm, Lauri Kuosmanen, Heidi Nieminen, et al. Chatbots to support mental wellbeing of people living in rural areas: can user groups contribute to co-design? *Journal of Technology in Behavioral Science*, 6:652–665, 2021.

[219] Heloisa Candello, Claudio Pinhanez, Michael Muller, and Mairieli Wessel. Unveiling practices of customer service content curators of conversational agents. *Proc. ACM Hum.-Comput. Interact.*, 6(CSCW2), nov 2022.

[220] Ana Paula Chaves and Marco Aurelio Gerosa. How should my chatbot interact? a survey on social characteristics in human–chatbot interaction design. *International Journal of Human–Computer Interaction*, 37(8):729–758, 2021.

[221] Jan de Wit. Leveraging large language models as simulated users for initial, low-cost evaluations of designed conversations. In *International Workshop on Chatbot Research and Design*, pages 77–93, Cham, 2023. Springer Nature Switzerland.

[222] Jingchao Fang, Nikos Arechiga, Keiichi Namaoshi, Nayeli Bravo, Candice Hogan, and David A. Shamma. On llm wizards: Identifying large language models' behaviors for wizard of oz experiments, 2024.

[223] Tommaso Calo and Christopher Maclellan. Towards educator-driven tutor authoring: Generative ai approaches for creating intelligent tutor interfaces. In *Proceedings of the Eleventh ACM Conference on Learning @ Scale*, L@S '24, page 305–309, New York, NY, USA, 2024. Association for Computing Machinery.

[224] Julian Coda-Forno, Marcel Binz, Jane X. Wang, and Eric Schulz. Cogbench: a large language model walks into a psychology lab, 2024.

[225] Keiichi Namikoshi, Alex Filipowicz, David A. Shamma, Rumen Iliev, Candice L. Hogan, and Nikos Arechiga. Using llms to model the beliefs and preferences of targeted populations, 2024.

[226] Hang Jiang, Xiajie Zhang, Xubo Cao, Cynthia Breazeal, Deb Roy, and Jad Kabbara. Personallm: Investigating the ability of large language models to express personality traits, 2024.

[227] Yunfan Shao, Linyang Li, Junqi Dai, and Xipeng Qiu. Character-llm: A trainable agent for role-playing. *arXiv preprint arXiv:2310.10158*, 2023.

[228] Yun-Shiuan Chuang, Agam Goyal, Nikunj Harlalka, Siddharth Suresh, Robert Hawkins, Sijia Yang, Dhavan Shah, Junjie Hu, and Timothy T. Rogers. Simulating opinion dynamics with networks of llm-based agents, 2024.

[229] Yoonseo Choi, Eun Jeong Kang, Seulgi Choi, Min Kyung Lee, and Juho Kim. Proxona: Leveraging LLM-Driven Personas to Enhance Creators' Understanding of Their Audience, August 2024. arXiv:2408.10937 [cs].

[230] Alexander Sasha Vezhnevets, John P Agapiou, Avia Aharon, Ron Ziv, Jayd Matyas, Edgar A Duéñez-Guzmán, William A Cunningham, Simon Osindero, Danny Karmon, and Joel Z Leibo. Generative agent-based modeling with actions grounded in physical, social, or digital space using concordia. *arXiv preprint arXiv:2312.03664*, 2023.

[231] Nian Li, Chen Gao, Mingyu Li, Yong Li, and Qingmin Liao. Econagent: Large language model-empowered agents for simulating macroeconomic activities, 2024.

[232] Omar Shaikh, Valentino Emil Chai, Michele Gelfand, Diyi Yang, and Michael S. Bernstein. Rehearsal: Simulating conflict to teach conflict resolution. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery.

[233] Joy He-Yueya, Noah D Goodman, and Emma Brunskill. Evaluating and optimizing educational content with large language model judgments. *arXiv preprint arXiv:2403.02795*, 2024.

[234] Songlin Xu, Xinyu Zhang, and Lianhui Qin. Eduagent: Generative student agents in learning, 2024.

[235] Zhengyuan Liu, Stella Xin Yin, Geyu Lin, and Nancy F. Chen. Personality-aware student simulation for conversational intelligent tutoring systems, 2024.

[236] Julia M. Markel, Steven G. Opferman, James A. Landay, and Chris Piech. Gpteach: Interactive ta training with gpt-based students. In *Proceedings of the Tenth ACM Conference on Learning @ Scale*, L@S '23, page 226–236, New York, NY, USA, 2023. Association for Computing Machinery.

[237] Zheyuan Zhang, Daniel Zhang-Li, Jifan Yu, Linlu Gong, Jinchang Zhou, Zhiyuan Liu, Lei Hou, and Juanzi Li. Simulating classroom education with llm-empowered agents, 2024.

[238] Manh Hung Nguyen, Sebastian Tschiatschek, and Adish Singla. Large language models for in-context student modeling: Synthesizing student's behavior in visual programming, 2024.

[239] Bahar Radmehr, Adish Singla, and Tanja Käser. Towards generalizable agents in text-based educational environments: A study of integrating rl with llms, 2024.

[240] Chieh-Yang Huang, Jing Wei, and Ting-Hao 'Kenneth' Huang. Generating educational materials with different levels of readability using llms, 2024.

[241] Ruiyi Wang, Stephanie Milani, Jamie C Chiu, Shaun M Eack, Travis Labrum, Samuel M Murphy, Nev Jones, Kate Hardy, Hong Shen, Fei Fang, et al. Patient-{\Psi}: Using large language models to simulate patients for training mental health professionals. *arXiv preprint arXiv:2405.19660*, 2024.

[242] Mary Besterfield-Sacre, Cynthia J Atman, and Larry J Shuman. Characteristics of freshman engineering students: Models for determining student attrition in engineering. *Journal of Engineering Education*, 86(2):139–149, 1997.

[243] Alexander W Astin and Helen S Astin. Undergraduate science education: the impact of different college environments on the educational pipeline in the sciences. final report., 1992.

[244] Sherry Y Chen and Robert D Macredie. Cognitive modeling of student learning in web-based instructional programs. *International Journal of Human-Computer Interaction*, 17(3):375–402, 2004.

[245] Konstantina Chrysafiadi and Maria Virvou. Student modeling approaches: A literature review for the last decade. *Expert Systems with Applications*, 40(11):4715–4729, 2013.

[246] Jiandong Sun, Michael P Dunne, Xiang-Yu Hou, and Ai-qiang Xu. Educational stress scale for adolescents: development, validity, and reliability with chinese students. *Journal of psychoeducational assessment*, 29(6):534–546, 2011.

[247] Howard J Klein, Michael J Wesson, John R Hollenbeck, Patrick M Wright, and Richard P DeShon. The assessment of goal commitment: A measurement model meta-analysis. *Organizational behavior and human decision processes*, 85(1):32–55, 2001.

[248] Adele E Gottfried. Academic intrinsic motivation in elementary and junior high school students. *Journal of educational psychology*, 77(6):631, 1985.

[249] Diana Kathleen May. *Mathematics self-efficacy and anxiety questionnaire*. PhD thesis, University of Georgia Athens, GA, USA, 2009.

[250] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain–of–thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[251] Manie Tadayon and Gregory J Pottie. Predicting student performance in an educational game using a hidden markov model. *IEEE Transactions on Education*, 63(4):299–304, 2020.

[252] Rabia Maqsood, Paolo Ceravolo, Cristóbal Romero, and Sebastián Ventura. Modeling and predicting students' engagement behaviors using mixture markov models. *Knowledge and Information Systems*, 64(5):1349–1384, 2022.

[253] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: deep hierarchical feature learning on point sets in a metric space. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5105–5114, Red Hook, NY, USA, 2017. Curran Associates Inc.

[254] Christina Sue-Chan and Mark Ong. Goal assignment and performance: Assessing the mediating roles of goal commitment and self-efficacy and the moderating role of power distance. *Organizational Behavior and Human Decision Processes*, 89(2):1140–1161, 2002.

[255] Yuka Mikami. Relationships between goal setting, intrinsic motivation, and self-efficacy in extensive reading. *Jacet journal*, 61:41–56, 2017.

[256] Sandra G Hart and Lowell E Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in psychology*, volume 52, pages 139–183. Elsevier, 1988.

[257] Raina Langevin, Ross J Lordon, Thi Avrahami, Benjamin R. Cowan, Tad Hirsch, and Gary Hsieh. Heuristic evaluation of conversational agents. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[258] Jacob Cohen, Patricia Cohen, Stephen G West, and Leona S Aiken. *Applied multiple regression/-correlation analysis for the behavioral sciences.* Routledge, New York, NY, USA, 2013.

[259] Wynne Harlen and Colin Holroyd. Primary teachers' understanding of concepts of science: Impact on confidence and teaching. *International journal of science education*, 19(1):93–105, 1997.

[260] Kenneth R Koedinger, Albert T Corbett, and Charles Perfetti. The knowledge-learning-instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive science*, 36(5):757–798, 2012.

[261] Daniel L Schwartz, Jessica M Tsang, and Kristen P Blair. *The ABCs of how we learn: 26 scientifically proven approaches, how they work, and when to use them.* WW Norton & Company, 2016.

[262] Danielle R Thomas, Erin Gatz, Shivang Gupta, Vincent Aleven, and Kenneth R Koedinger. The neglected 15%: Positive effects of hybrid human-ai tutoring among students with disabilities. In *International Conference on Artificial Intelligence in Education*, pages 409–423, Cham, 2024. Springer Nature Switzerland.

[263] Eleanor Manspile, Matthew N Atwell, and John M Bridgeland. Immigrant students and english learners: Challenges faced in high school and postsecondary education., 2021.

[264] Matt Beane. *The Skill Code: How to Save Human Ability in an Age of Intelligent Machines.* HarperCollins, 2024.

[265] Ryan SJ d Baker, Albert T Corbett, and Vincent Aleven. More accurate student modeling through contextual estimation of slip and guess probabilities in bayesian knowledge tracing. In *Intelligent Tutoring Systems: 9th International Conference, ITS 2008, Montreal, Canada, June 23-27, 2008 Proceedings 9*, pages 406–415, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[266] Xiangyi Liao and Daniel M Bolt. Guesses and slips as proficiency-related phenomena and impacts on parameter invariance. *Educational Measurement: Issues and Practice*, 00(0):1–9, 2024.

[267] Dan Wolff and Ahmed Seffah. Ux modeler: a persona-based tool for capturing and modeling user experience in service design. In *IFIP WG 13.2 Workshop at INTERACT 2011*, pages 7–16, 2011.

[268] Djilali Idoughi, Ahmed Seffah, and Christophe Kolski. Adding user experience into the interactive service design loop: a persona-based approach. *Behaviour & Information Technology*, 31(3):287–303, 2012.

[269] Minju Yoo, Hyoungwook Jin, and Juho Kim. How do teachers create pedagogical chatbots?: Current practices and challenges. *arXiv preprint arXiv:2503.00967*, 2025.

[270] Reuben Thiessen. Build-a-bot workshop series, 2024.

Acknowledgments in Korean

2017년 봄에 인간-컴퓨터 상호작용 개론 (CS374) 수업을 들으면서, 코딩이 마냥 재밌고 수업에서 다룬 CAPTCHA의 교묘한 크라우드소싱 기법이 너무 재미있어 그해 겨울 연구 인턴을 시작했습니다. 학부 때부터 하나의 연구실에서 오래 있으면서 많은 분을 만나고, 영감을 얻고, 같이 고민하며, 많은 배움을 얻어갑니다.

- 항상 저를 격려하고 걱정하며 제가 하고 싶은 것을 하도록 모든 것을 지원해 주신 부모님과 가족에게 감사합니다.
- 인간-컴퓨터 상호작용 분야를 알려주시고, 연구자로서 성품과 역량을 알려주신 김주호 교수님에게 감사합니다.
- 재미있는 연구를 할 수 있도록 같이 생각을 보태고, 실행하고, 끝까지 함께한 공동 저자 모두에게 감사합니다.
- 제 석사 논문을 검토하여 주시고, 건설적인 의견을 주셨던 논문 심사 위원 이의진, 이탁연 교수님에게 감사합니다.
- 연구 토론을 함께하고 제가 더 높은 기준을 바라보도록 영향을 준 이전·현재 KIXLAB 구성원들에게 감사합니다.
- 생각이 복잡한 날, 연구와 일에서 벗어나도록 제 이야기를 들어주고 어울려 주었던 대학 친구들에게 감사합니다.

Curriculum Vitae in Korean

이　　　름: 진 형 욱

생 년 월 일: 1997년 04월 12일

학　　력

2012. 9. – 2015. 5.	Yew Chung International School of Beijing
2015. 9. – 2023. 2.	한국과학기술원 전산학부 (학사)
2023. 3. – 2025. 8.	한국과학기술원 전산학부 (석사)

경　　력

2024. 3. – 2024. 6.	한국과학기술원 전산학부 조교 (Introduction to Algorithms)
2023. 9. – 2023. 12.	한국과학기술원 전산학부 조교 (Introduction to Programming)
2017. 12. – 2023. 2.	한국과학기술원 KIXLAB 연구 인턴
2022. 6. – 2022. 9.	University of California San Diego DesignLab 연구 인턴
2019. 8. – 2021. 10.	VCNC 소프트웨어 엔지니어

연 구 업 적

1. **Jin, H.**, Yoo, M., Park, J., Lee, Y., Wang, X., & Kim, J. (2025, May). "TeachTune: Reviewing Pedagogical Agents Against Diverse Student Profiles with Simulated Students." In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems* (to be published).
2. **Jin, H.**, Lee, S., Shin, H., & Kim, J. (2024, May). "Teach AI How to Code: Using large language models as teachable agents for programming education." In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (pp. 1-28).
3. **Jin, H.**, Chang, M., & Kim, J. (2019, May). "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* (pp. 1-6).
4. **Jin, H.**, & Kim, J. (2024). "CodeTree: A System for Learnersourcing Subgoal Hierarchies in Code Examples." *Proceedings of the ACM on Human-Computer Interaction, 8(CSCW1)*, 1-37.
5. Lee, C., Han, D., **Jin, H.**, & Oh, A. (2019, June). "AutomaTA: Human-machine interaction for answering context-specific questions." In *Proceedings of the Sixth (2019) ACM Conference on Learning@ Scale* (pp. 1-4).
6. **Jin, H.**, Kim, Y., Park, Y. S., Tilekbay, B., Son, J., & Kim, J. (2024, July). "Using Large Language Models To Diagnose Math Problem-solving Skills At Scale." In *Proceedings of the Eleventh ACM Conference on Learning@ Scale* (pp. 471-475).

7. Yen, Y. C. G., Pan, I. Y., Lin, G., Li, M., **Jin, H.**, Chen, M., ... & Dow, S. P. (2024). "When to Give Feedback: Exploring Tradeoffs in the Timing of Design Feedback." In *Proceedings of the 16th Conference on Creativity & Cognition (C&C '24)*. (pp. 292–310).

8. Yen, Y. C. G., E, J. L., **Jin, H.**, Li, M., Lin, G., Pan, I. Y., & Dow, S. P. (2024). "ProcessGallery: Contrasting Early and Late Iterations for Design Principle Learning." *Proceedings of the ACM on Human-Computer Interaction, 8(CSCW1)*, (pp. 1-35).

9. **Jin, H.**, & Kim, J. (2022). "Learnersourcing Subgoal Hierarchies of Code Examples." In *LSGCS@ L@S* (pp. 35-39).

10. Moore, S., Singh, A., Lu, X., **Jin, H.**, Khosravi, H., Denny, P., ... & Stamper, J. (2024, July). "Learnersourcing: Student-generated Content@ Scale: 2nd Annual Workshop." In *Proceedings of the Eleventh ACM Conference on Learning@ Scale* (pp. 559-562).