**CoderAgent:** **Simulating** **Student** **Behavior** **for** **Personalized** **Programming**

**Learning** **with** **Large** **Language** **Models**

**Yi** **Zhan**1 , **Qi** **Liu**1,2 \* , **Weibo** **Gao**1 , **Zheng** **Zhang**1 , **Tianfu** **Wang**1 ,

**Shuanghong** **Shen**2 , **Junyu** **Lu**2 and **Zhenya** **Huang**1,2

1 State Key Laboratory of Cognitive Intelligence, University of Science and Technology of China

2Institute of Artificial Intelligence, Hefei Comprehensive National Science Center

{zy0119, weibogao, zhangzheng}@mail.ustc.edu.cn,{qiliuql, huangzhy}@ustc.edu.cn,

{ljunyu, shshen}@iai.ustc.edu.cn, tianfuwang.cs@gmail.com,

**Abstract**

[arX v:2505.20642v1 c s.AI 27 2025](https://arxiv.org/abs/2505.20642v1)i[]May

Personalized programming tutoring, such as exer- cise recommendation, can enhance learners’ effi- ciency, motivation, and outcomes, which is increas- ingly important in modern digital education. How- ever, the lack of sufficient and high-quality pro- gramming data, combined with the mismatch be- tween offline evaluation and real-world learning, hinders the practical deployment of such systems.

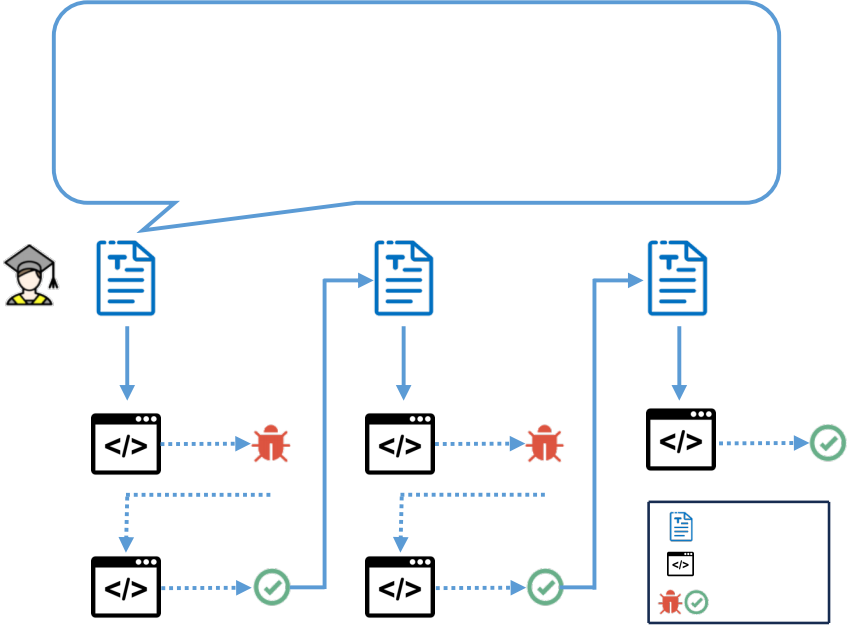
To address this challenge, many approaches at- tempt to simulate learner practice data, yet they often overlook the fine-grained, iterative nature of programming learning, resulting in a lack of inter- pretability and granularity. To fill this gap, we pro- pose a LLM-based agent, CoderAgent, to simulate students’ programming processes in a fine-grained manner without relying on real data. Specifically, we equip each human learner with an intelligent agent, the core of which lies in capturing the cog- nitive states of the human programming practice process. Inspired by ACT-R, a cognitive architec- ture framework, we design the structure of Coder- Agent to align with human cognitive architecture by focusing on the mastery of programming knowl- edge and the application of coding ability. Recog- nizing the inherent patterns in multi-layered cogni- tive reasoning, we introduce the Programming Tree of Thought (PTOT), which breaks down the pro- cess into four steps: why, how, where, and what.

This approach enables a detailed analysis of iter- ative problem-solving strategies. Finally, experi- mental evaluations on real-world datasets demon- strate that CoderAgent provides interpretable in- sights into learning trajectories and achieves accu- rate simulations, paving the way for personalized programming education.

**1** **Introduction**

In the digital age, programming has emerged as a critical skill, fueling the growth of online educational platforms like *Leet-* *Code.com*, which have democratized access to programming

\* Corresponding Author

q:**: Find the Maximum Sum Subarray**

Write a function that takes an array of integers as input and returns the maximum sum of any contiguous subarray. If all numbers are negative, return 0.

**Programming Knowledge:** Conditional Logic, Loops, Arrays

q3

q1

q2

Student

**Accepted**

**Compile Error**

**Wrong Answer**

Question Code

**Accepted**

**Accepted**

Feedback

Figure 1: The example of the programming practice process

education for learners worldwide [[Wang *et* *al.*, 2025](#bookmark2)]. By an- alyzing user programming data, these platforms deliver per- sonalized tutoring services, such as tailored exercise recom- mendations [[Zhao *et* *al.*, 2023](#bookmark3); [Gao *et* *al.*, 2025a](#bookmark4)], contextual programming hints [[Marwan *et* *al.*, 2019](#bookmark5)], and interactive ex- perimental simulations [[Gao and Zhu, 2023](#bookmark6)], all designed to optimize learning efficiency and enhance the overall educa- tional experience.

The effectiveness of these personalized tutoring services hinges on the availability of high-quality learner program- ming data, which is essential for training and evaluating adap- tive algorithms. However, the inherent complexity and time- intensive nature of programming often result in slow progress for many learners, leading to a scarcity of such data. Com- pounding this issue, stringent privacy regulations further re- strict the collection and utilization of learner data [[Reddy *et*](#bookmark7) [*al.*, 2022](#bookmark7)], exacerbating the data shortage. This scarcity, cou- pled with the disconnect between historical offline-collected data and contemporary online learning practices, creates a significant disparity between offline evaluation metrics and real-world online performance. As a result, translating re- search insights into practical programming applications re- mains a formidable challenge.

To bridge this gap, simulating learner programming exer- cise data has emerged as a promising solution [[Gao ,*etal.*](#bookmark8) [2025b](#bookmark8)]. Imagine an online programming platform equipped with a customizable coding simulation system that replicates

how human learners iteratively modify their code when tack- ling programming challenges. Such a system could generate valuable simulated data to train and evaluate personalized tu- toring algorithms. By analyzing these simulated interactions, algorithms could better adapt to diverse learner behaviors, narrowing the divide between simulated and real-world per- formance. Moreover, by anticipating potential code modifica- tions, the system could offer more targeted guidance, such as improving code quality and fostering a more adaptive learn- ing environment.

Existing methods for simulating learner exercises primar- ily focus on predicting whether a learner’s response to a given problem is correct [[Li *et* *al.*, 2022](#bookmark9); [Puri *et* *al.*, 2021](#bookmark10)]. Still, they fall short of capturing the nuanced, iterative process of coding and revision. This limitation undermines their abil- ity to provide meaningful insights for complex programming education. In reality, programming exercises involve a mul- tifaceted learning process. As depicted in Figure [1](#bookmark1), learners write code to solve a specific problem (*e.g.,* q1 ), receive sys- tem feedback (*e.g.,compilation* *errors* *or* *correctness* *confir-* *mation*), and iteratively refine their code based on this feed- back. This cycle continues until the learner moves on to the next problem. Such complexity demands a more so- phisticated simulator that goes beyond simplistic response prediction. Furthermore, traditional simulators are predom- inantly data-driven, relying on large volumes of program- ming response data for training. This makes them ill-suited for cold-start scenarios, where data availability is limited, a common challenge in real-world applications. Recent ad- vancements in large language models (LLMs) [[Li *et* *al.*, 2024](#bookmark11); [Yuan *et* *al.*, 2024](#bookmark12)] have demonstrated their ability to emulate human-like intelligence. By leveraging LLMs to construct intelligent agents, it becomes possible to simulate complex human practice processes [[Xu *et* *al.*, 2024](#bookmark13)]. Additionally, the in-context learning capabilities of LLMs enable them to perform cold-start simulations with minimal reliance on real- world data [[Huang *et* *al.*, 2023](#bookmark14)], offering a promising solu- tion to the limitations of current simulators. While some re- cent studies have begun exploring the use of LLMs to predict students’ next code submissions, they still fail to capture the detailed, iterative nature of code modification. Consequently, there is a pressing need to develop an advanced agent capable of simulating the intricate, fine-grained process of program- ming problem-solving and code revision.

In this paper, we leverage the intelligence of LLM agents to propose CoderAgent, a novel framework that can simulate students’ programming processes in a fine-grained manner without relying on real data. Specifically, it includes Mem- ory, Tools, Planning & Action, and Reflection modules. The **Memory** module is designed under the guidance of the ACT- R cognitive architecture [[Anderson and Lebiere, 2014](#bookmark15); [Gao *et*](#bookmark16) [*al.*, 2021](#bookmark16)]. This architecture posits that human programming behavior is primarily determined by programming-related cognitive factors and describes the cognitive structure of hu- man programming as consisting of two components: the mas- tery of programming knowledge and the application of cod- ing skills. As illustrated in Figure [1](#bookmark1), problem q1 involves the knowledge concept conditional logic, loops and arrays, and thus the performance in solving q1 is determined by the

level of mastery of the relevant knowledge and coding ability. Based on this, we partition the memory into two regions that store programming knowledge and coding skills, respectively [[Berges *et* *al.*, 2012](#bookmark17)], enabling CoderAgent to capture both the conceptual understanding and practical skills of learners. Additionally, by analyzing each student’s coding style and common errors from their past work, the framework gener- ates code that mirrors the student’s realistic behavior. When a student encounters challenges beyond their current abili- ties, the agent simulates plausible mistakes they might make in subsequent iterations, providing insights into their learn- ing process and informing the design of personalized pro- gramming exercises. As for the **Planning** **&** **Action** module, emerging from the inherent complexity of human learning [[Choi and Lee, 2009](#bookmark18)], where learners must engage in multi- layered cognitive reasoning, from identifying problems to im- plementing solutions, we introduce the Programming Tree of Thought (PTOT), a novel reasoning mechanism specifically designed for programming tasks. PTOT decomposes debug- ging into four steps: *why,* *how,* *where,* *and* *what*, pinpointing specific code segments for modification and leveraging the rich contextual information generated from **Tools** and pro- gramming process to enhance decision-making. By modeling the cognitive pathways students take when debugging, PTOT enables CoderAgent to achieve both high interpretability and fine-grained analysis of their behavior. Additionally, a Re- flection module ensures the accuracy of the generated code. If the **Reflection** module detects that a modification exceeds the student’s capabilities or deviates from their coding profile, it prompts the agent to refine the output, ensuring alignment with the student’s abilities and coding style.

Finally, we evaluate CoderAgent using GPT-4o and Gpt- 4o-mini APIs on several real-world programming datasets, demonstrating its ability to effectively simulate student cod- ing behavior. We further apply CoderAgent to real-world pro- gramming tasks: mistake-prone point analysis and test case generation, showing its utility in practical coding applica- tions. The performance ofCoderAgent underscores its poten- tial for improving programming education through personal- ized learning scenarios. Our main contributions are summa- rized as follows:

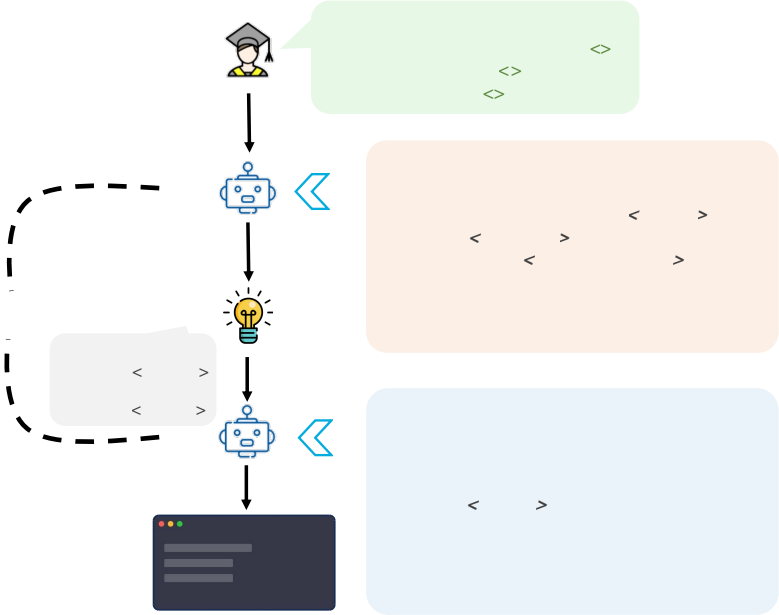
• We introduce a novel LLM agent framework for pro- grammer simulation that eliminates reliance on large- scale datasets, enabling effective performance even in data-scarce scenarios.

• We propose the Programming Tree of Thought (PTOT) to simulate realistic student code modifications, offering detailed insights into their learning process with inter- pretability and granularity.

• We validate our framework on real-world code submis- sion data, showcasing its utility in personalized pro- gramming education.

**2** **Related** **Work**

**Learner** **Response** **Simulation.** Simulating human behav- ior has been a focal point in educational research [[Hillier](#bookmark19) [*et* *al.*, 2024](#bookmark19); [Chen and Yen, 2024](#bookmark20)], with early approaches relying on rule-based models and cognitive architectures,



Continuous Interactions Data H! at Timestep t

**Long-term** **Memory:**

My ProgramminKnowledge:

My Coding Ability :

My Coding Style:

***Update***

**Planning** **&** **Action**

**Context**: You are a students learning programming. You profile is ***Profile***

Question: ***Question***

Previous codes: ***Previous*** ***Codes***

**Instructions**: Use PTOT to generate the next step code based on previous your profile and code records.

**LLM**

愈

**(b)** **Dynamic** **Learner** **Long-term** **Memory**

**Reflection**

The previous

segment: #code0 The modified

segment: #code1

Programming Knowledge

**Reflection**

**Context**: You are an expert in coding. There is a coding agent. The agent can generate next modified code based on student’s ***Profile***

**Instructions**: Think about whether the

student has the ability to modify the code in this way and the modification aligns with the student's profile

**LLM**

Coding Ability

**Generated**

**Code**

**Segment**

 Coding Style

|  |  |  |
| --- | --- | --- |
|  | **Programming** **Interaction** **Process**  **(d)** **Planning** (Programming Tree of Thought)  1. Why to modify 2. How to modify  3. Where to modify 4. What to modify  **W**hy **H**ow **W**here **W**hat  Question Detail  **(c)** **Short-term** **Memory**  Debugging Logs  Revision History  ***Update***  **Action**   |  | | --- | | (e) Execution **Tools**    Compiler Information |   Result  Feedback  **(f)** **Reflection**  1. Whether the student has the ability to modify the code in this way  2. Whether the code modification aligns with the student's profile |

(a) The overall pipeline of CoderAgent

Figure 2: CoderAgent Framework. (a) The overall pipeline of CoderAgent; (b) Long-term memory, containing knowledge, ability and style;

(c) Short-term memory, consisting of question, history and debugging logs; (d) Four steps of PTOT in planning module; (e) Execution tools, returning compiler information; (f) Reflection module, checking whether the output code is reasonable.

such as ACT-R [[Anderson and Lebiere, 2014](#bookmark15)], to replicate students’ problem-solving processes. Knowledge Tracing (KT) [[Corbett and Anderson, 1994](#bookmark21)] and Cognitive Diagno- sis (CD) [[Zhang *et* *al.*, 2024](#bookmark22); [Zhang ,*etal.*](#bookmark23)] are founda- tional techniques in the field of educational data mining, focusing on modeling the evolution of a learner’s knowl- edge state over time and simulating the final response of stu- dents. Recent studies in KT have explored more sophisticated models, such as deep learning-based approaches like Deep Knowledge Tracing [[Piech *et* *al.*, 2015](#bookmark24); [Liu *et* *al.*, 2021a](#bookmark25); [Liu *et* *al.*, 2021b](#bookmark26)], which employ recurrent neural networks to capture complex temporal patterns in simulating behav- iors. In the context of programming education, researchers have adapted KT frameworks to simulate students’ mastery of programming concepts and skills [[Wang *et* *al.*, 2017](#bookmark27); [Zhu *et* *al.*, 2022](#bookmark28); [Li *et* *al.*, 2022](#bookmark9); [Liang *et* *al.*, 2022](#bookmark29)]. A few of works, such as OKT [[Liu *et* *al.*, 2022](#bookmark30)], aim to predict stu- dents’ code. Nonetheless, how to more precisely simulate how students modify their code remains an unresolved issue. In this study, we delve into capturing richer details about stu- dents’ problem-solving strategies, misconceptions, and itera- tive refinement processes.

**LLM-based** **Agents.** Recently, LLMs have presented new opportunities to enhance human simulation [[Aher ,*etal.*](#bookmark31) [2023](#bookmark31); [Park *et* *al.*, 2023](#bookmark32); [Zhao *et* *al.*, 2023](#bookmark3); [Hu *et* *al.*, 2024](#bookmark33)]. The in-context learning capabilities of LLMs have been used as agents in various domains, including recommendation [[Huang *et* *al.*, 2023](#bookmark14)] and education [[Park *et* *al.*, 2024](#bookmark34); [Qadir,](#bookmark35) [2023](#bookmark35); [Zhou *et* *al.*, 2024](#bookmark36)]. Edu4Agent [[Gao *et* *al.*, 2025b](#bookmark8)] rep- resents the latest framework employing LLM agents to simu- late student responses within the educational domain. How- ever, its design targets general scenarios, making it unsuit- able for simulating programming data. For programming ed- ucation, although recent work [[Liu *et* *al.*, 2022](#bookmark30)] has explored LLMs to predict students’ code more specifically, their gran- ularity is insufficient, and they heavily rely on data. Different from these approaches, to the best of our knowledge, we are the first to simulate the programming process through agents. By focusing on the incremental modifications students make to their code, we aim to replicate the cognitive processes un-

derlying their problem-solving strategies. This simulation framework provides a more accurate representation of how students tackle programming tasks. Such insights are critical for developing adaptive educational tools and interventions tailored to individual learning trajectories.

**3** **Methodology**

In this section, we present the proposed framework, named CoderAgent. Our approach enables LLM agents to simulate the programming process. The overall framework of our pro- posed CoderAgent is depicted in Figure [2](#bookmark37).

**3.1** **Preliminaries**

**Problem** **Definition.** In our study, we define the program- ming history of a student as a sequence of code submis- sions, denoted as a programming history sequence Hn = {h1 , h2,..., hn }, where each hi = (ei , ci , fi ) represents the code ci submitted by the student at step i for a specific exer- cise ei. Here, ci is the code submitted by the student, ei refers to the exercise associated with that submission, and fi is the correctness indicator of the submission. Specifically, fi = 1 if the submission passes all tests and meets the requirements of the exercise, and fi = 0 if the submission fails to meet the requirements. Each code submission represents an incremen- tal change in the student’s code, reflecting their attempts to solve the exercise and address previous errors. The primary goal of our study is to predict the student’s next code sub- mission as well as whether it will ultimately result in a cor- rect solution. Formally, given the historical sequence of code submissions Hn = {h1 , h2,..., hn }, our objective is to pre- dict the student’s next code submission cn+1 and determine whether it will ultimately be correct. While the intermediate submissions c1 , c2 , . . . , cn are part of the student’s process, the focus is on predicting the student’s next submission and determining if it leads to a correct final solution.

**Overview** **of** **CoderAgent.** The CoderAgent framework consists of five key modules: Memory, Tools, Planning, Ac- tion, and Reflection, each designed to simulate the iterative and dynamic coding process. Rooted in the ACT-R theory

from cognitive psychology, the Memory module categorizes a student’s programming proficiency into programming knowl- edge and coding skills, enabling the agent to store and re- call code snippets, mistakes, and adjustments for informed decision-making. The Planning module incorporates the Pro- gramming Tree of Thought (PTOT), an innovative structure that captures the iterative progression of a student’s thought process during coding, offering fine-grained interpretability and aligning with cognitive reasoning to enhance realism. Tools provide compilers for various programming languages, delivering feedback on compilation errors, while the Ac- tion module governs step-by-step coding strategies, includ- ing writing, testing, and refining code. Finally, the Reflec- tion module evaluates the agent’s output, ensuring alignment with the student’s abilities and refining problem-solving ap- proaches through iterative feedback. Together, these modules create a comprehensive and personalized simulation of the student coding process.

**3.2** **Memory** **Module**

The memory module is specifically designed to address the unique demands of code generation and iterative program- ming, setting it apart from memory systems in prior stud- ies. Unlike traditional agents that operate on broad problem- solving tasks, our framework must manage the intricacies of programming, such as handling syntax, debugging, and iter- ative refinements. To meet these challenges, we divide mem- ory into long-term memory and short-term memory, each tai- lored to the dynamics of the programming process.

**Long-term** **Memory.** The long-term memory component is designed to simulate the foundational knowledge and skills that students develop throughout their programming journey. Grounded in the ACT-R theory from cognitive psychology, the acquisition of expertise is conceptualized as comprising two primary elements: knowledge and ability. Applied to programming, these correspond to programming knowledge (*e.g.,* *understanding* *syntax,* *structures,* *and* *algorithms*) and coding ability (*e.g.,* *implementing,* *debugging,* *and* *optimiz-* *ing* *code*). Research [[McKeithen *et* *al.*, 1981](#bookmark38)] has shown that students at different skill levels demonstrate distinct hierar- chical structures in their programming knowledge. Expert- level students, for example, tend to organize programming constructs (such as keywords or syntax rules) into meaning- ful hierarchies, enabling more efficient recall and application. This hierarchical knowledge structure significantly enhances their coding ability, leading to more precise and effective pro- gramming. In our framework, the long-term memory mod- ule reflects this layered organization of programming knowl- edge and captures its influence on coding ability. Beyond these core factors, the module also records individual coding styles and common errors for each student. This personal- ized profiling ensures that the generated code closely aligns with the student’s natural tendencies, enabling a more realis- tic simulation of how students might iteratively modify code or make specific mistakes during programming. By contin- uously updating the stored programming knowledge, coding ability, style, and common errors using historical data from students’ prior coding tasks, we ensure that the memory mod- ule evolves to increasingly reflect the real-world behaviors of

individual students. During training, the framework updates

MT iteratively using historical task data H at time t:

MT (t + 1) = f(MT (t), H) (1)

where f is the update function that integrates new insights

from task data D to refine the student’s programming knowl-

edge, ability, style, and error profiles.

**Short-term** **Memory.** In contrast, short-term memory cap- tures transient information related to the current task, such as recent question details, written code snippets and debugging history. This allows the agent to maintain a clear and updated context throughout the iterative coding process. For example, when resolving a compilation error, the short-term memory retains the feedback from the compiler and integrates it into subsequent edits. Similarly, it stores intermediate results dur- ing debugging or optimization, enabling the agent to refine the code systematically without losing sight of prior changes. The adaptability of short-term memory makes it essential for managing the back-and-forth nature of programming tasks. This dynamic capability is key to creating realistic simula- tions of student coding behavior.

**3.3** **Programming** **Tools**

The tools module in CoderAgent is responsible for interfac- ing with external compilers to simulate the real-world feed- back loop students rely on during programming. This mod- ule enables the agent to compile code snippets, interpret error messages, and retrieve translation information from various programming languages. By leveraging these tools, the agent gains access to detailed feedback that informs the iterative re- finement process of the code. For example, when the agent submits code to a compiler, the Tools module processes the compilation results, including syntax errors, runtime issues, or warnings. These results are then fed back into the agent’s planning module, allowing it to adjust its strategy and im- prove the code accordingly. By simulating the practical util- ity of compilers, the tools module enhances the realism and functionality of the CoderAgent, enabling it to mimic how students utilize external feedback to iteratively refine their programming solutions.

**3.4** **Elaborate** **Planning** **&** **Action**

The planning & action module is the decision-making core of the CoderAgent. It is designed to replicate the strategic and procedural steps students take during programming. This module orchestrates the agent’s actions by breaking down complex programming tasks into manageable subtasks, se- lecting the appropriate steps, and executing them iteratively. **Planning.** Inspired by prior research on Chain-of-Thought (CoT) [[Wei *et* *al.*, 2022](#bookmark39)] reasoning, we propose a special- ized framework for the programming domain: **Programming** **Tree** **of** **Thought** **(PTOT)**. PTOT is designed to address the unique characteristics of the iterative nature of code devel- opment. Unlike other learning scenarios, the process in pro- gramming is often localized. When a student receives error feedback from a compiler, they typically only need to re- vise specific code snippets to resolve the issue. Even when facing semantic errors, adjustments are usually confined to

small segments of the code rather than requiring wholesale rewrites. Regardless of whether these adjustments succeed or introduce new errors, this localized iteration forms the core of a student’s problem-solving approach in programming. To replicate this iterative reasoning, PTOT breaks down the stu- dent’s thought process into four key steps:

1. **Why** **do** **I** **need** **to** **modify** **the** **code?** This step identifies the root cause of the issue based on feedback, such as errors flagged by the compiler. It involves understanding whether the error arises from syntax or semantics.

2. **How** **do** **I** **modify** **the** **code?** Here, the agent formu- lates a strategy for addressing the issue. For example, it decides whether to rewrite the logic, change variable declarations, or update function parameters.

3. **Where** **should** **I** **modify** **the** **code?** This step pinpoints the location that requires changes, guided by compiler messages or a structural understanding of the code.

4. **What** **should** **I** **modify** **in** **the** **code?** Finally, the agent determines the precise changes to be made, such as edit- ing lines or correcting incorrect data types.

To facilitate this process, the LLM agent integrates multi- ple sources of input: the latest version of the student’s code ct , exercise or problem description ei and feedback from the Tools module Ft. Using these inputs, the planning module formulates a structured roadmap that directs the subsequent action module. This roadmap ensures that each step is logi- cally grounded and consistent with the feedback received. By modeling this iterative and localized thought process, PTOT enables CoderAgent to effectively simulate the nuanced way students approach problem-solving in tasks.

**Action.** The action module is responsible for executing the specific steps outlined in the Planning module. This includes identifying the most likely code segment requiring modifica- tion and generating its replacement. Guided by the planning module’s structured roadmap and real-time feedback from the tools module, the action module ensures that the modi- fications align with the coding objective while addressing the highlighted issues. The action module identifies the code seg- ment st within ct that is most likely the source of the issue

and proposes a replacement s based on planned modifica-

tion strategy pt. Formally, the output of the action module can be represented as:

s = g (st , pt ) , (2)

where g is a transformation function that generates s by ap-

plying planned modifications pt to the identified code seg- ment st. where g applies planned modifications pt to code

segment st to generate s .

**3.5** **Ability** **Reflection**

Despite the advanced reasoning capabilities of LLMs, they can occasionally exhibit errors in judgment or decision- making. For instance, a LLM agent might suggest a code modification that appears syntactically correct but is beyond the user’s skill level or inconsistent with their typical cod- ing style. To address such issues, prior research has incorpo- rated self-reflection mechanisms into LLMs, enabling them

Table 1: The statistics of the datasets.

|  |  |  |
| --- | --- | --- |
| Dataset | CodeNet | CSEDM |
| # Students | 137,833 | 506 |
| # Exercises | 4,053 | 50 |
| # Solutions | 13,916,868 | 125,578 |
| Avg. exercise number of learner | 3,437.96 | 47.49 |

Table 2: Tasks that can be accomplished by different methods.

|  |  |
| --- | --- |
|  | Task 1 Task 2 Task 3 Task 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DKT  codeBERT PDKT  PST  Agent4Edu OKT  CoderAgent | -  -  -  -  -  √  √ | -  -  -  -  -  √  √ | -  -  -  -  -  √  √ | √  √  √  √  √  -  √ |

to identify and mitigate errors autonomously. Inspired by this approach, we have developed a Reflection module to enhance the realism and accuracy of the CoderAgent framework. The execution steps of the reflection module are listed as follows:

• **Whether** **the** **student** **has** **the** **ability** **to** **modify** **the** **code** **in** **this** **way.** This step assesses whether the pro- posed modification aligns with the student’s coding abil- ities, as captured in the memory module. If the modifi- cation requires advanced knowledge that exceed the stu- dent’s profiled skill level, the reflection module flags the change as inappropriate.

• **Whether** **the** **code** **modification** **aligns** **with** **the** **stu-** **dent’s** **profile.** Beyond correctness, the reflection mod- ule evaluates whether the modification adheres to the student’s typical patterns of problem-solving. By refer- encing long-term memory, the module ensures that the modifications remain consistent with their prior work.

Through this two-step evaluation, the reflection module iden- tifies potential mismatches between the agent’s outputs and the student’s profile. When discrepancies are detected, the module provides feedback to the planning module, prompt- ing the agent to revise its strategy. This iterative feedback loop reduces reasoning errors and improves the agent’s abil- ity to emulate the student’s thought process accurately.

**4** **Experiments**

**4.1** **Experimental** **Setup**

**Datasets.** Following prior research [[Li *et* *al.*, 2022](#bookmark9); [Liu *et*](#bookmark30) [*al.*, 2022](#bookmark30)], we utilized two datasets for experiments: CodeNet [[Puri *et* *al.*, 2021](#bookmark10)] and CSEDM. The basic data statistics are presented in Table [1](#bookmark40). CodeNet, provided by IBM, com- prises programming submissions sourced from two popular online coding platforms, AIZU.org and Atcoder.org. Given the diversity of programming languages in CodeNet, we re- stricted our analysis to the Python subset, ensuring a more focused and computationally efficient evaluation. CSEDM is a publicly available, college-level dataset containing real- world student programming submissions with Java language. To maintain data quality, we excluded students with an insuf- ficient number of submissions, concentrating on those with richer activity records. Furthermore, we sampled subsets

Table 3: Results of comparison methods on different task. Only OKT and CoderAgent can complete task 1, 2 and 3. The best results are bold, the second-best results are marked by an underline, and ↑ means the higher score the better performance.

 CSEDM  CodeNet

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Task 1 | Task 2 | Task 3 | Task 1 | Task 2 | Task 3 |
|  | ACC ↑ | ACC ↑ | CodeBLEU ↑ | ACC ↑ | ACC ↑ | CodeBLEU ↑ |
| OKT | 0.2531 | 0.4430 | 0.5115 | 0.2063 | 0.2222 | 0.1805 |
| CoderAgent(4o-mini)  CoderAgent(4o) | 0.3670  **0.3841** | 0.5229  **0.5324** | 0.7580  **0.7698** | **0.4464**  0.4010 | **0.5893**  0.4853 | **0.5946**  0.5900 |
|  |

Table 4: The AUC scores of comparison methods on task 4.

|  |  |  |
| --- | --- | --- |
|  | CSEDM | CodeNet |
| DKT | 0.5184 | 0.5239 |
| codeBERT | 0.5142 | 0.5255 |
| PDKT | 0.5091 | 0.5078 |
| PST | 0.5261 | 0.5272 |
| Agent4Edu | 0.5071 | 0.5116 |
| CoderAgent(4o-mini) | 0.5351 | 0.5386 |
| CoderAgent(4o) | **0.5493** | **0.5522** |

from both datasets for experiments due to the expensive cost of API calls associated with processing. Both datasets were split into 80% training, 10% validation, and 10% testing.

**Evaluation.** We define four tasks for the evaluation:

• **Task** **1:** Predict the next modification intention.

• **Task** **2:** Predict where will be modified next.

• **Task** **3:** Predict the student’s next code submission.

• **Task** **4:** Predict whether the next code submission will result in an AC (Accepted) outcome.

Task 1 and task 2 is utilized to evaluate the fine-grained pre- cision of code iteration, assessing whether the system can ac- curately simulate what students think and do. We leverage LLMs (GPT-4o-mini) to assess the accuracy (ACC) of these two tasks. For task 3, we use metrics that evaluate the frame- work’s ability to generate code that closely matches the actual student submission in the test set. Specifically, we employ CodeBLEU [[Ren *et* *al.*, 2020](#bookmark44)], an adaptation of the traditional BLEU metric tailored for code evaluation. CodeBLEU as- sesses the similarity between the predicted and actual code by considering both syntactic and semantic aspects, providing a comprehensive measure of prediction quality. For task 4, the performance of the framework on this task is measured using Area Under the Curve (AUC), which evaluates the model’s ability to distinguish between positive and negative outcomes across various thresholds.

**Methods.** The details of the above baselines are as follows:

• DKT [[Piech *et* *al.*, 2015](#bookmark24)] leverages RNN to assess knowledge mastery.

• codeBERT [[Feng *et* *al.*, 2020](#bookmark45)] is a model for code representation learning. To track programming skill progression, we leverage codeBERT to represent the source code, and incorporate code embeddings, exer- cise embeddings, and feedback embeddings to model the complete learning sequence of students using a LSTM [[Hochreiter, 1997](#bookmark46)] structure.

• PDKT [[Puri *et* *al.*, 2021](#bookmark10)] utilizes the exercise informa- tion and the source code to implement double-sequence modeling process to track programming knowledge.

• PST [[Li *et* *al.*, 2022](#bookmark9)] divided programming skill into programming knowledge and coding ability to get more finegrained assessment.

• Agent4Edu [[Gao *et* *al.*, 2025b](#bookmark8)] is the only prior work to use LLM agents to simulate student’s response. In our experiments, we did not use the cognitive diagnosis tools employed in the original paper, and we plan to use the full version of Agent4Edu in future research.

• OKT [[Liu *et* *al.*, 2022](#bookmark30)] is the only framework which uti- lizes LLMs to track programming knowledge.

We use GPT-4o(2024-11-20) and GPT-4o-mini(2024-07-18) as the CoderAgent core. Our code and the simulated data is partially available at [https://github.com/USTChandsomeboy/](https://github.com/USTChandsomeboy/CoderAgent) [CoderAgent](https://github.com/USTChandsomeboy/CoderAgent).

**4.2** **Experimental** **Results**

To evaluate the unity of simulation, we designed four tasks to compare different methods. The experimental results are pre- sented in Table [3](#bookmark42) and Table [4](#bookmark43), revealing several noteworthy findings. First, our proposed method exhibits a robust ability to adapt to diverse task requirements, as shown in Table [2](#bookmark41). In contrast, traditional KT models, which serve as baselines, are only capable of addressing Task 4, while the OKT model is restricted to Tasks 1, 2, and 3. These limitations underscore the broader applicability and versatility of the CoderAgent framework in handling a wider range of tasks.

For Tasks 1, 2, and 3, CoderAgent consistently outper- forms OKT. This superior performance highlights CoderA- gent’s ability to extract fine-grained insights from students’ learning processes. By simulating the iterative nature of code modification, CoderAgent can effectively predict students’ next modification intentions and accurately identify specific areas of the code requiring adjustment. These strengths demonstrate the model’s capacity to provide a detailed and precise representation of students’ problem-solving strate- gies, offering a significant improvement over prior methods. In Task 4, which involves predicting whether the next code submission will result in an Accepted outcome, traditional KT models perform poorly, yielding AUC scores near 0.5. This underperformance is largely attributed to the unique characteristics of programming tasks, where students often require multiple attempts to achieve a correct solution. The resulting dataset imbalance, characterized by a significantly higher proportion of incorrect instances compared to cor- rect instances, leads these models to predominantly predict

Table 5: The specific iterative process of CoderAgent modifying the code

|  |  |  |  |
| --- | --- | --- | --- |
| **Submisson** **1** | **Submisson** **2** | **Submisson** **3** | **Submisson** **4** |
|  |  |  |  |

incorrect outcomes. In contrast, CoderAgent achieves no- table improvements by leveraging its agent-based framework. Rather than relying on straightforward label fitting, it assesses whether a code submission has the potential to satisfy accep- tance criteria. This approach enables more accurate predic- tions and a balanced performance in Task 4.

Overall, these results demonstrate that CoderAgent outper- forms baseline models across all tasks. It provides a nuanced and interpretive understanding of students’ learning behav- iors, thereby offering significant advantages over traditional methodologies in the domain of programming education.

**4.3** **Case** **Study**

**Motivation:** To validate the interpretability of CoderAgent in simulating student’s code modifications during the itera- tive process, we choose a question as a case study: *Consider* *a* *series* *of* *numbers* *starting* *from* *a* *given* *value* *start* *and* *run-* *ning* *up* *to* *but* *not* *including* *end.* *The* *goal* *is* *to* *return* *a* *new* *String[]* *array* *containing* *the* *string* *form* *of* *these* *numbers,* *with* *specific* *rules:* *for* *multiples* *of* *3,* *use* *“Fizz”* *instead* *of* *the* *number;for* *multiples* *of* *5,* *use* *”Buzz”;* *and* *for* *multiples* *of* *both* *3* *and* *5,* *use* *“FizzBuzz”.* *In* *Java,* *String.valueOf(xxx)* *will* *convert* *an* *integer* *(or* *other* *types)* *to* *a* *string.*

We selected a CoderAgent simulation process, summarized in Table [5](#bookmark47), omitting code sections that were correct and un- changed. In the student’s initial submission, three errors were identified: an undefined variable n, a misspelling of valueOf as valueof, and an incorrect loop starting value. In the first iteration, CoderAgent used compiler feedback to identify and fix the undefined variable. In the second iteration, it cor- rected the misspelled function, recognizing String.valueOf() as the intended method based on task hints. Once the syn- tax errors were resolved, CoderAgent focused on the seman- tic issue within the loop—the starting value was incorrect for the intended sequence. The agent, simulating the student’s knowledge level, accurately identified the semantic error in the loop and modified it to produce the correct result.

The entire process is as same as the student’s actual modi- fication path: first resolving syntax errors, then addressing se- mantic issues. This case study highlights CoderAgent’s abil- ity to model a student’s reasoning in real-time, ensuring both precision in detecting errors and interpretability.

**4.4** **Simulations** **in** **Programming** **Applications**

**Motivation:** Designing experiments that are suitable for stu- dents is essential for classroom instruction. To assess the ap- plicability and practicality of CoderAgent, we applied it in simulating a course-based programming experiment.

|  |  |  |
| --- | --- | --- |
| **Number of Mistake-Prone Points** | With CoderAgent  **15**  **Test Sample Count**  Without CoderAgent  15  **9**  10  **8 8 7**  **4**  5  0  Question 1 Question 2 Question 3  **(a) Simulation in mistake-prone point analysis** | **16**  **14**  15  **13**  10  **9**  **6**  **6**  5  0  Question 1 Question 2 Question 3  **(b) Simulation in test case generation** |

Figure 3: Simulation results in programming applications

The experiment involved three programming questions, us- ing data from 20 randomly selected students. For each task, we examined common errors under two conditions: one with- out CoderAgent and the other with simulated submissions generated by CoderAgent. The results, shown in Figure [3](#bookmark48) (a), reveal that using simulated data improved the identifica- tion of error-prone areas. With CoderAgent, a wider range of common errors was detected, providing deeper insights into the challenges students may encounter. This suggests that the simulated results produced can significantly support the design of course experiments. By identifying a broader spec- trum of errors, educators can create assignments that better align with students’ learning trajectories, addressing poten- tial difficulties more effectively. Additionally, the increased variety of identified errors enables the development of more robust test cases. Building on this, we conducted a second simulated experiment, where test cases for the problem were generated under both conditions. The experimental results, presented in Figure [3](#bookmark48) (b), show that CoderAgent generated more comprehensive test cases, ensuring that assignments re- inforce students’ understanding of key concepts. This high- lights the potential of CoderAgent to enhance the quality and effectiveness of programming education.

**5** **Conclusion**

In this paper, we presented CoderAgent, a LLM-based agent to simulate students’ iterative coding processes with in- terpretability and granularity, without relying on extensive datasets. By drawing on the ACT-R theory, we defined pro- gramming proficiency as a combination of knowledge and coding skills. Central to our approach is the Programming Tree of Thought, which deconstructs the planning phase into four distinct steps, enabling CoderAgent to effectively model students’ modification intentions and strategies. Experimen- tal evaluations on real-world datasets and programming sim- ulations demonstrated the framework’s effectiveness in cap- turing and analyzing student coding behaviors.

**Acknowledgements**

This research was supported by grants from the National Key Research and Development Program of China (Grant No. 2024YFC3308200), the National Natural Science Foundation of China (62337001), the Key Technologies R & D Program of Anhui Province (No. 202423k09020039) and the Funda- mental Research Funds for the Central Universities.

**References**

[Aher *et* *al.*, 2023] Gati V Aher, Rosa I Arriaga, and Adam Tauman Kalai. Using large language models to sim- ulate multiple humans and replicate human subject studies. In *International* *Conference* *on* *Machine* *Learning*, pages 337–371. PMLR, 2023.

[Anderson and Lebiere, 2014] John R Anderson and Chris- tian J Lebiere. *The* *atomic* *components* *of* *thought*. Psy- chology Press, 2014.

[Berges *et* *al.*, 2012] Marc Berges, Andreas Mhling, and

Peter Hubwieser. The gap between knowledge and abil- ity. In *Proceedings* *of* *the* *12th* *Koli* *Calling* *international* *conference* *on* *computing* *education* *research*, pages 126– 134, 2012.

[Chen and Yen, 2024] Mao-Siang Chen and An-Zi Yen. E- qgen: Educational lecture abstract-based question genera- tion system. In *Proceedings* *of* *the* *Thirty-Third* *Interna-* *tional* *Joint* *Conference* *on* *Artificial* *Intelligence,* *IJCAI-* *24*, pages 8631–8634. International Joint Conferences on Artificial Intelligence Organization, 8 2024. Demo Track.

[Choi and Lee, 2009] Insu Choi and Kyounghee Lee. De- signing and implementing a case - based learning envi- ronment for enhancing ill - structured problem solving: classroom management problems for prospective teach- ers. *Educational* *Technology* *Research* *and* *Development*, 57(1):99–129, 2009.

[Corbett and Anderson, 1994] Albert T Corbett and John R Anderson. Knowledge tracing: Modeling the acquisi- tion of procedural knowledge. *User* *modeling* *and* *user-* *adapted* *interaction*, 4:253–278, 1994.

[Feng *et* *al.*, 2020] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv* *preprint* *arXiv:2002.08155*, 2020.

[Gao and Zhu, 2023] Yazhuo Gao and Xuehua Zhu. Re- search on the learning experience of virtual simulation class experimental teaching and learning based on the per- spective of nursing students. *BMC* *nursing*, 22(1):367, 2023.

[Gao *et* *al.*, 2021] Weibo Gao, Qi Liu, Zhenya Huang, Yu Yin, Haoyang Bi, Mu-Chun Wang, Jianhui Ma, Shi- jin Wang, and Yu Su. Rcd: Relation map driven cognitive diagnosis for intelligent education systems. In *Proceed-* *ings* *of* *the* *44th* *international* *ACM* *SIGIR* *conference* *on* *research* *and* *development* *in* *information* *retrieval*, pages 501–510, 2021.

[Gao *et* *al.*, 2025a] Weibo Gao, Qi Liu, Rui Li, Yuze Zhao, Hao Wang, Linan Yue, Fangzhou Yao, and Zheng Zhang. Denoising programming knowledge tracing with a code graph-based tuning adaptor. In *Proceedings* *of* *the* *31st* *ACM* *SIGKDD* *Conference* *on* *Knowledge* *Discovery* *and* *Data* *Mining* *V.* *1*, pages 354–365, 2025.

[Gao *et* *al.*, 2025b] Weibo Gao, Qi Liu, Linan Yue, Fangzhou Yao, Rui Lv, Zheng Zhang, Hao Wang, and Zhenya Huang. Agent4edu: Generating learner re- sponse data by generative agents for intelligent education systems. *arXiv* *preprint* *arXiv:2501.10332*, 2025.

[Hillier *et* *al.*, 2024] Dylan Hillier, Cheston Tan, and Jing Jiang. Social learning through interactions with other agents: A survey. In *Proceedings* *of* *the* *Thirty-Third* *International* *Joint* *Conference* *on* *Artificial* *Intelligence,* *IJCAI-24*, pages 8067–8076. International Joint Confer- ences on Artificial Intelligence Organization, 8 2024. Sur- vey Track.

[Hochreiter, 1997] S Hochreiter. Long short-term memory. *Neural* *Computation* *MIT-Press*, 1997.

[Hu *et* *al.*, 2024] Yuxuan Hu, Gemju Sherpa, Lan Zhang, WeihuaLi, Quan Bai, Yijun Wang, and Xiaodan Wang. An llm-enhanced agent-based simulation tool for information propagation. In *Proceedings* *of* *the* *Thirty-Third* *Interna-* *tional* *Joint* *Conference* *on* *Artificial* *Intelligence,* *IJCAI-* *24*, pages 8679–8682. International Joint Conferences on Artificial Intelligence Organization, 8 2024. Demo Track.

[Huang *et* *al.*, 2023] Xu Huang, Jianxun Lian, Yuxuan Lei, Jing Yao, Defu Lian, and Xing Xie. Recommender ai agent: Integrating large language models for interac- tive recommendations. *arXiv* *preprint* *arXiv:2308.16505*, 2023.

[Li *et* *al.*, 2022] Ruixin Li, Yu Yin, Le Dai, Shuanghong Shen, Xin Lin, Yu Su, and Enhong Chen. Pst: measuring skill proficiency in programming exercise process via pro- gramming skill tracing. In *Proceedings* *of* *the* *45th* *Inter-* *national* *ACM* *SIGIR* *Conference* *on* *Research* *and* *Devel-* *opment* *in* *Information* *Retrieval*, pages 2601–2606, 2022.

[Li *et* *al.*, 2024] Qian Li, Zhuo Chen, Cheng Ji, Shiqi Jiang, and Jianxin Li. Llm-based multi-level knowledge gen- eration for few-shot knowledge graph completion. In *Proceedings* *of* *the* *Thirty-Third* *International* *Joint* *Con-* *ference* *on* *Artificial* *Intelligence,* *IJCAI-24*, pages 2135– 2143. International Joint Conferences on Artificial Intelli- gence Organization, 8 2024. Main Track.

[Liang *et* *al.*, 2022] Y. Liang, T. Peng, Y. Pu, et al. Help- dkt: an interpretable cognitive model of how students learn programming based on deep knowledge tracing. *Scientific* *Reports*, 12:4012, 2022.

[Liu *et* *al.*, 2021a] Qi Liu, Zhenya Huang, Yu Yin, En- hong Chen, Hui Xiong, Yu Su, and Guoping Hu. Ekt: Exercise-aware knowledge tracing for student perfor- mance prediction. *IEEE* *Trans.* *on* *Knowl.* *and* *Data* *Eng.*, 33(1):100–115, 2021.

[Liu *et* *al.*, 2021b] Qi Liu, Shuanghong Shen, Zhenya Huang, Enhong Chen, and Yonghe Zheng. A survey of knowledge tracing. *CoRR*, abs/2105.15106, 2021.

[Liu *et* *al.*, 2022] Naiming Liu, Zichao Wang, Richard Bara- niuk, and Andrew Lan. Open-ended knowledge tracing for computer science education. In *Proceedings* *of* *the* *2022* *Conference* *on* *Empirical* *Methods* *in* *Natural* *Language* *Processing*, 2022.

[Marwan *et* *al.*, 2019] Samiha Marwan, Joseph Jay Williams, and Thomas Price. An evaluation of the impact of automated programming hints on per- formance and learning. In *Proceedings* *of* *the* *2019* *ACM* *Conference* *on* *International* *Computing* *Education* *Research*, pages 61–70, 2019.

[McKeithen *et* *al.*, 1981] Katherine B McKeithen, Judith S Reitman, Henry H Rueter, and Stephen C Hirtle. Knowl- edge organization and skill differences in computer pro- grammers. *Cognitive* *Psychology*, 13(3):307–325, 1981.

[Park *et* *al.*, 2023] Joon Sung Park, Joseph O’Brien, Car- rie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive sim- ulacra of human behavior. In *Proceedings* *of* *the* *36th* *an-* *nual* *acm* *symposium* *on* *user* *interface* *software* *and* *tech-* *nology*, pages 1–22, 2023.

[Park *et* *al.*, 2024] Minju Park, Sojung Kim, Seunghyun Lee, Soonwoo Kwon, and Kyuseok Kim. Empowering per- sonalized learning through a conversation-based tutoring system with student modeling. In *Extended* *Abstracts* *of* *the* *CHI* *Conference* *on* *Human* *Factors* *in* *Computing* *Sys-* *tems*, pages 1–10, 2024.

[Piech *et* *al.*, 2015] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. Deep knowledge trac- ing. *Advances* *in* *neural* *information* *processing* *systems*, 28, 2015.

[Puri *et* *al.*, 2021] Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Ju- lian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, et al. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. *arXiv* *preprint* *arXiv:2105.12655*, 2021.

[Qadir, 2023] Junaid Qadir. Engineering education in the era of chatgpt: Promise and pitfalls of generative ai for edu- cation. In *2023* *IEEE* *Global* *Engineering* *Education* *Con-* *ference* *(EDUCON)*, pages 1–9. IEEE, 2023.

[Reddy *et* *al.*, 2022] Surendranadha Reddy Byrapu Reddy, Prabu Ravichandran, Srihari Maruthi, Mohan Raparthi, Praveen Thunki, and Sarath Babu Dodda. Ethical con- siderations in ai and data science-addressing bias, privacy, and fairness. *Australian* *Journal* *of* *Machine* *Learning* *Re-* *search* *&* *Applications*, 2(1):1–12, 2022.

[Ren *et* *al.*, 2020] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *arXiv* *preprint* *arXiv:2009.10297*, 2020.

[Wang *et* *al.*, 2017] Lisa Wang, Angela Sy, Larry Liu, and Chris Piech. Learning to represent student knowledge on programming exercises using deep learning. *International* *Educational* *Data* *Mining* *Society*, 2017.

[Wang *et* *al.*, 2025] Tianfu Wang, Yi Zhan, Jianxun Lian, Zhengyu Hu, Nicholas Jing Yuan, Qi Zhang, Xing Xie, and Hui Xiong. Llm-powered multi-agent framework for goal-oriented learning in intelligent tutoring system. In *ACM* *Web* *Conference*, 2025.

[Wei *et* *al.*, 2022] Jason Wei, Xuezhi Wang, Dale Schuur- mans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances* *in* *neural* *information* *processing* *systems*, 35:24824–24837, 2022.

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

*arXiv* *preprint* *arXiv:2404.07963*, 2024.

[Yuan *et* *al.*, 2024] Yu Yuan, Lili Zhao, Kai Zhang, Guangt- ing Zheng, and Qi Liu. Do llms overcome shortcut learn- ing? an evaluation of shortcut challenges in large language models. In *Proceedings* *of* *the* *2024* *Conference* *on* *Em-* *pirical* *Methods* *in* *Natural* *Language* *Processing*, pages 12188–12200, 2024.

[Zhang *et* *al.*, ] Zheng Zhang, Wei Song, Qi Liu, Qingyang Mao, Yiyan Wang, Weibo Gao, Zhenya Huang, Shijin Wang, and Enhong Chen. Towards accurate and fair cog- nitive diagnosis via monotonic data augmentation. In *The* *Thirty-eighth* *Annual* *Conference* *on* *Neural* *Information* *Processing* *Systems*.

[Zhang *et* *al.*, 2024] Zheng Zhang, Le Wu, Qi Liu, Jiayu Liu, Zhenya Huang, Yu Yin, Yan Zhuang, Weibo Gao, and Enhong Chen. Understanding and improving fairness in cognitive diagnosis. *Science* *China* *Information* *Sciences*, 67(5):152106, 2024.

[Zhao *et* *al.*, 2023] Guanhao Zhao, Zhenya Huang, Yan Zhuang, Jiayu Liu, Qi Liu, Zhiding Liu, Jinze Wu, and En- hong Chen. Simulating student interactions with two-stage imitation learning for intelligent educational systems. In *Proceedings* *of* *the* *32nd* *ACM* *International* *Conference* *on* *Information* *and* *Knowledge* *Management*, pages 3423– 3432, 2023.

[Zhou *et* *al.*, 2024] Zihao Zhou, Bin Hu, Chenyang Zhao, Pu Zhang, and Bin Liu. Large language model as a pol- icy teacher for training reinforcement learning agents. In *Proceedings* *of* *the* *Thirty-Third* *International* *Joint* *Con-* *ference* *on* *Artificial* *Intelligence,* *IJCAI-24*, pages 5671– 5679. International Joint Conferences on Artificial Intelli- gence Organization, 8 2024. Main Track.

[Zhu *et* *al.*, 2022] Renyu Zhu, Dongxiang Zhang, Chengcheng Han, Ming Gaol, Xuesong Lu, Weining Qian, and Aoying Zhou. Programming knowledge tracing: A comprehensive dataset and a new model. In *2022* *IEEE* *International* *Conference* *on* *Data* *Mining* *Workshops* *(ICDMW)*, pages 298–307. IEEE, 2022.