scientific reports

Check for updates

OPEN

Difficulty aware programming knowledge tracing via large language models

Lina Yang, Xinjie Sun✉, Hui Li, Ran Xu & Xuqin Wei

Knowledge Tracing (KT) assesses students' mastery of specific knowledge concepts and predicts their problem-solving abilities by analyzing their interactions with intelligent tutoring systems. Although recent years have seen significant improvements in tracking accuracy with the introduction of deep learning and graph neural network techniques, existing research has not sufficiently focused on the impact of difficulty on knowledge state. The text understanding difficulty and knowledge concept difficulty of programming problems are crucial for students' responses; thus, accurately assessing these two types of difficulty and applying them to knowledge state prediction is a key challenge. To address this challenge, we propose a *Difficulty aware Programming Knowledge Tracing via Large Language Models*(DPKT) to extract the text understanding difficulty and knowledge concept difficulty of programming problems. Specifically, we analyze the relationship between knowledge concept difficulty and text understanding difficulty using an attention mechanism, allowing for dynamic updates to students' s. This model combines an update gate mechanism with a graph attention network, significantly improving the assessment accuracy of programming problem difficulty and the spatiotemporal reflection capability of knowledge state. Experimental results demonstrate that this model performs excellently across various language datasets, validating its application value in programming education. This model provides an innovative solution for programming knowledge tracing and offers educators a powerful tool to promote personalized learning.

Knowledge Tracing (KT) aims to model students' knowledge mastery states by analyzing their interaction data with Intelligent Tutoring Systems (ITS), automatically assessing their mastery of specific knowledge concepts, and predicting whether they can successfully solve related problems in their next attempts[1,2]. In programming education, knowledge tracing is particularly important; effective knowledge tracing not only helps educators accurately assess students' programming abilities but also provides data support for personalized learning[3,4]. Through dynamic monitoring of students' knowledge state, educators can develop more personalized teaching strategies to meet the diverse learning needs of different students[5].

In recent years, the field of knowledge tracing has developed various innovative models by introducing technologies such as deep learning, graph neural networks, and personalized learning[2,6]. These include deep learning models[1,7], memory network models[8,9], graph neural network models[10–12], and probabilistic models[13,14], which have significantly improved tracking accuracy and the effectiveness of personalized learning. In the field of programming education, knowledge tracing technology also plays a crucial role. For example, the PDKT model constructs a bipartite graph to embed programming problems and combines the improved pre-trained model PLCodeBERT for code embedding and tracking students' programming knowledge state[15]. The DSAKT model addresses sparse data issues through a self-attention mechanism, enhancing the accuracy of knowledge tracing[16]. The GPPKT model combines students' learning abilities with a Variational Autoencoder (VAE), improving model performance through the integration of personalized answer sequences and knowledge graphs[17]. With continuous technological advancements, the field of knowledge tracing will continue to evolve, bringing more possibilities for personalized education and improved learning outcomes.

Currently, research on programming knowledge tracing primarily focuses on predicting students' learning states based on programming problems and their submission records[18,19]. However, this approach often overlooks the significant impact of problem difficulty on answer correctness. Programming problems require students not only to accurately understand and analyze the problem text but also to apply knowledge of programming language syntax, algorithms, and data structures to solve real-world problems, as shown in Fig. 1, which illustrates the impact of text understanding difficulty and knowledge concept difficulty on predicting students' knowledge state. In this process, both the text understanding difficulty and knowledge concept difficulty of

School of Computer Science, Liupanshui Normal University, Liupanshui 553000, China. ✉email: xinjiesun@lpssy.edu.cn

Difficulty of Problem Text Understanding	1	2	2	3	4
Difficulty of Problem Knowledge Concepts	1	2	3	4	4
KCs	Add	Array	Graph	Multi-threading	Optimization

Fig. 1. A toy example demonstrating the impact of problem difficulty on programming knowledge tracing.

programming problems are crucial for students' success in answering these problems. Currently, the annotation of programming problem difficulty largely relies on manual efforts, making it challenging to accurately label both text understanding difficulty and knowledge concept difficulty simultaneously. Therefore, efficiently and accurately determining the text understanding difficulty and knowledge concept difficulty for each programming problem and utilizing this information to optimize the model's predictions of students' knowledge state has become a key challenge in the field of programming knowledge tracing. This paper aims to address this challenge by proposing an innovative method to enhance the efficiency and accuracy of matching programming problem difficulties. Through this approach, we can comprehensively and objectively assess students' programming abilities, thereby providing more targeted support and interventions for programming education.

To ensure that the difficulty of programming problems is reasonably matched to students' knowledge state, we face three major challenges: **first**, the diversity of programming problems complicates the assessment of semantic difficulty; **second**, the interaction between knowledge concept difficulty and text understanding difficulty is difficult to effectively distinguish; and **third**, the sequential modeling of knowledge concepts does not adequately consider the strong correlations between them. To address these challenges, we propose the following solutions: **first**, by introducing large language models to extract deep semantic features from programming problems, we enhance the precision of semantic difficulty assessment; **second**, we use an attention mechanism to dynamically analyze the relationship between knowledge concept difficulty and text understanding difficulty to ensure the model's detailed performance; **finally**, combining graph attention mechanisms with a carefully designed gating mechanism allows for better attention to the spatiotemporal relationships between knowledge state.

Based on these ideas, we propose difficulty-aware programming knowledge tracing with large language models (DPKT). This model dynamically adjusts students' mastery states of different difficulty knowledge concepts by combining an update gate mechanism with a graph attention network. The DPKT model inputs programming problems and corresponding knowledge concepts into a large language model to extract rich semantic features and uses an attention mechanism to differentiate between knowledge concept difficulty and text understanding difficulty, ensuring precise updates to knowledge state.

The main contributions of this model include:

- The DPKT model combines large language models with graph attention networks, improving the assessment accuracy of programming problem difficulty.
- The dynamic update mechanism enables the model to reflect changes in students' knowledge state in real time, enhancing sensitivity to students' learning progress.
- The model proposes new solutions for distinguishing between knowledge concept and text understanding difficulty, providing solid theoretical support for personalized learning.

By leveraging the powerful data processing capabilities of large language models, this model achieves significant improvements in the efficiency and accuracy of semantic information extraction. Additionally, the DPKT model integrates large language models, CodeBERT, and graph attention networks, addressing the issue of accurately predicting students' knowledge state in situations where knowledge concepts are the same but semantic difficulties differ, thus enhancing the model's performance and interpretability. In comparison with existing knowledge tracing models, the DPKT model demonstrates excellent performance across various language datasets, validating its value in practical applications.

In summary, the DPKT model not only provides an innovative solution for programming knowledge tracing but also offers educators a powerful tool, with the expectation of making a positive impact in the field of programming education and promoting further research development in this area.

Related works

In the context of the rapid development of modern educational technology, knowledge tracing models have gradually become important tools for personalized learning, helping educators monitor and assess students' learning progress more effectively. At the same time, the introduction of large language models (LLMs) has brought new possibilities to the field of knowledge tracing, driving innovation and transformation in educational practices[20]. The following will delve into the foundational models of knowledge tracing, the advancements in dynamic adaptive models, as well as the applications and challenges associated with Large Language Models (LLMs) in educational contexts.

Knowledge tracing

The foundational models of knowledge tracing lay the theoretical groundwork for subsequent research and applications. Bayesian Knowledge Tracing (BKT), as a classic model, utilizes Bayesian networks to predict students' knowledge state, emphasizing the importance of uncertainty and prior knowledge, thus providing a solid statistical foundation for knowledge tracing[21]. Additionally, the application of probabilistic models in knowledge tracing further supports the theoretical framework of the entire field[22]. By incorporating deep learning and recurrent neural network techniques, these models can effectively capture students' mastery levels of knowledge concepts, enabling tracking and prediction of personalized learning[1]. Researchers have enriched our understanding of the learning process by identifying key factors influencing student learning, allowing for assessments of knowledge mastery and providing feedback to educators to help them adjust teaching strategies[23,24]. Building on this, Memory-Aware Knowledge Tracing (MKT) considers the influence of students' memory factors, enhancing the accuracy of knowledge state predictions by simulating human memory processes[25]. The comprehensive application of these models allows for a more thorough understanding and improvement of students' learning processes.

With the advancement of knowledge tracing technology, dynamic and adaptive knowledge tracing models have emerged, better adapting to the changes in students' learning. These models not only enhance the flexibility of knowledge tracing but also improve responsiveness to individual differences among students[8,11,26]. For instance, models that incorporate dynamic student classification can better adjust to varying ability levels among students, thus enhancing the personalized learning experience[5]. Moreover, Knowledge Tracing Machines propose a knowledge tracing method based on factorization machines, focusing on accurately modeling students' knowledge state, providing a new perspective for the model[27]. By using convolutional neural networks to model personalized features in students' learning processes, the learning capabilities of the models are further enhanced[28]. The integration of these dynamic and adaptive models enables a more comprehensive response to students' learning needs and changes.

In recent years, significant progress has been made in the field of programming knowledge tracing, particularly in developing models that incorporate the difficulty of programming exercises and forgetting factors. For example, a deep knowledge tracing (KT) method integrates the difficulty of programming exercises and forgetting factors, using the GPT-3 algorithm to analyze problem difficulty and constructing knowledge structure graphs with graph neural networks, thereby improving the model's predictive performance and interpretability[29]. Additionally, research has explored how the difficulty and order of programming problems in online assessment systems affect user performance and cognitive load, revealing various influencing factors through statistical analysis[3]. Common challenges in programming education, such as compilation errors and skill misunderstandings, have also received considerable attention[4]. At the same time, a meta-analysis summarized effective interventions in programming education, highlighting the outstanding performance of visualization and tangible learning tools in improving learning outcomes[30].

The development of knowledge tracing models provides important theoretical support for educational practices, especially excelling in personalized learning and dynamic adaptability. However, research in the area of programming problems, particularly regarding problem difficulty, remains relatively weak, especially concerning the complexity of text understanding and the definition of specific knowledge concept difficulty, with limited relevant literature available. Therefore, in-depth exploration of these dimensions will not only enrich the application context of knowledge tracing models but also provide a more robust theoretical foundation for the deepening and optimization of programming education.

Development of large language models in knowledge tracing

Large language models (LLMs) have emerged as powerful interpreters in the field of knowledge tracing, showcasing significant application potential and introducing innovative methods and tools to the educational domain[31]. For instance, CoRE proposes using LLMs as interpreters across various programming forms, including natural language programming, pseudocode programming, and AI agent flow programming[32]. Leveraging LLMs can help students understand code, thereby enhancing the learning experience[33]. To make programming more intuitive, Low-code LLM has introduced a low-code approach for visual programming using large language models, providing a user-friendly programming environment for non-specialists[34]. In terms of assessing the effectiveness of technology, LLMs have been found to effectively address issues in example programming[35]. Additionally, the Copilot Evaluation Harness presents a method for assessing software programming guided by large language models, providing a framework for subsequent application evaluations[36]. These studies indicate that LLMs have a broad application prospect in knowledge tracing and programming education.

The introduction of LLMs has significantly enhanced the capabilities of knowledge tracing models, injecting new vitality into the development of educational technology. For example, methods that enhance code knowledge tracing with large language models have further propelled innovation in this field[37,38]. In teaching applications, CS1-LLM explores methods to integrate LLMs into introductory computer science courses, effectively improving instructional outcomes[39]. Concurrently, research has found that novices use LLM-based

code generators to tackle programming tasks in self-directed learning environments, providing valuable insights for course design[40]. Furthermore, Forgetful Large Language Models discusses lessons learned from using LLMs in robotic programming, reflecting their limitations and challenges[41]. These studies demonstrate that LLMs not only enhance the functionality of knowledge tracing models but also reveal their extensive potential in educational practice.

Despite the promise of LLMs in education and the insights they offer for course design, revealing the diversity of educational goals[2,42], several challenges remain to be addressed. In terms of technological integration and infrastructure, incorporating LLMs into existing educational technologies requires substantial resources and technical support, and deployment in different educational environments may necessitate customized solutions. Additionally, ethical and legal issues cannot be overlooked; using LLMs involves concerns about the use and ownership of student data, necessitating compliance with data protection regulations. To tackle these challenges, close collaboration among educators, technology developers, policymakers, and researchers is essential to ensure that the application of LLMs in education is responsible, effective, and capable of enhancing educational quality and equity. Through this research, we can gain deeper insights into the challenges of LLMs in education and provide valuable perspectives for future development[43,44].

Preliminary

In this section, we will provide a formal definition of the difficulty-aware programming knowledge tracing with large language models. Additionally, we will introduce some fundamental elements and important embeddings involved in our proposed DPKT framework. Table 1 summarizes the mathematical symbols used in this paper.

Problem definition

In the online programming assessment system, the set of students is represented as *S*, the set of programming problems as *P*, the set of knowledge concepts as *K*, and the set of student code submission responses as *R*. The student set $S = \{s_1, s_2, \ldots, s_n\}$ contains *n* different students. The problemming problem set $P = \{p_1, p_2, \ldots, p_m\}$ represents *m* different coding problems. The knowledge concepts set $K = \{k_1, k_2, \ldots, k_v\}$ contains v different knowledge concepts, generated based on LLMs in this paper. The set of text understanding difficulty levels $TD = \{1, 2, 3, 4\}$ include 4 difficulty levels. The set of knowledge concepts difficulty levels $KD = \{1, 2, 3, 4\}$ also includes 4 difficulty levels. The set of student submission responses $R = \{0, 1\}$ contains two elements, indicating whether the student's submitted code is deemed acceptable by the system. Here, 1 represents a successful code submission, while 0 indicates a failed submission. During the student's programming practice, this is represented as a programming event sequence $L = \{(p_1, k_1, td_1, kd_1, r_1), (p_2, k_2, td_2, kd_2, r_2), \ldots, (p_t, k_t, td_t, kd_t, r_t)\}$. In this sequence, p_t represents the programming problem answered by the student at time step *t*, k_t indicates the corresponding knowledge concept generated by the large model, td_t denotes the text understanding difficulty level of the problem, kd_t represents the knowledge concept difficulty level, and r_t indicates the student's submission status at time step *t*, By analyzing the student's historical practice records, we can trace their mastery of programming knowledge. The definition of knowledge tracing in programming practice is as follows:

Knowledge Tracing(KT): The historical practice records of students $L = \{(p_1, k_1, td_1, kd_1, r_1), (p_2, k_2, td_2, kd_2, r_2), \ldots, (p_t, k_t, td_t, kd_t, r_t)\}$, The task of programming knowledge tracing (KT) is a dynamic process. A student's programming skill level progresses and changes over time based on their performance in understanding knowledge concepts and the difficulty of problem text. The goal of this task is to assess their mastery of programming skills based on their historical code submissions, in conjunction with their performance in knowledge concept mastery and text understanding, and to further predict their performance in the next programming task.

Difficulty definition

To better track students' programming learning progress and performance, we introduce two important factors: knowledge concept difficulty(KD) and problem text understanding difficulty(TD)[45,46]. Knowledge concept difficulty reflects the complexity of the programming concepts or skills involved in a problem. It can be quantified based on multiple dimensions, such as the coverage of knowledge concepts and the frequency with which students master these concepts. The Dreyfus model divides the skill acquisition process into: the Novice stage, the Advanced Beginner stage, the Competent stage, the Proficient and Expert stage. Each

Symbol	Description
S	The set of students
P	The set of programming problems
R	The set of response
m, n, v	The number of problem, student and KCs
KC	The set of Knowledge concepts
KD	Knowledge Concepts difficulty
TD	Text understanding difficulty
d_i	Knowledge Concepts difficulty level
u_i	Problem text understanding difficulty level

Table 1. Mathematical symbol and descriptions.

stage represents a different level of understanding and application ability. Based on this model, we define the difficulty levels of programming problems into four tiers corresponding to the novice, advanced, proficient, and expert stages, thus providing a systematic difficulty assessment framework[47–49]. Specifically, this is defined as: $KD = \{d_1, d_2, d_3, d_4\}$, where each d_i represents a specific difficulty level, with higher values indicating greater complexity in syntax.

Problem text understanding difficulty refers to the complexity of students' reading and comprehension of programming problems. This difficulty not only depends on whether the problem description is clear and concise but also relates to the terminology used in the problem and the complexity of the problem background. It is defined as: $TD = \{u_1, u_2, u_3, u_4\}$, where each u_i represents different levels of text understanding difficulty, with higher values indicating that the problem description is more complex and harder for students to understand. We generate the difficulty levels for each knowledge concept and the text understanding difficulty levels of the problems using large language models. Table 2 summarizes the descriptions and definitions of knowledge concepts and problem text understanding difficulties.

Embeddings

Before delving into the assessment of programming problem difficulty and the generation of embeddings based on large language models, we need to briefly introduce some key fundamental elements that aid in a deeper understanding of the model's internal workings. In this model, five key elements are primarily involved: problems (P), knowledge concepts (KCs), knowledge concept difficulty (KD), text understanding difficulty (TD), and student code submission responses (R), which are represented through embedding matrices. The problem embedding matrix $\mathrm{V} \in \mathbb{R}^{m \times d_p}$, represents the embeddings generated by CodeBERT for programming problems, with dimensions $m \times d_p$, where m is the number of problems and d_p is the dimension size of the problem embeddings. The knowledge concept embedding matrix $\mathrm{K} \in \mathbb{R}^{v \times d_k}$, represents the embeddings generated by CodeBERT for knowledge concepts related to programming problems, with dimensions $v \times d_k$, where $v$is the number of knowledge concepts and d_k is the dimension size of the knowledge concept embeddings. The knowledge concept difficulty embedding matrix $\mathrm{KD} \in \mathbb{R}^{4 \times d_g}$, represents the embeddings generated by the large language model for knowledge concept difficulty in programming problems, with dimensions $4 \times d_g$, where 4 corresponds to the four levels of knowledge concept difficulty and d_g is the dimension size of the knowledge concept difficulty embeddings. The text understanding difficulty embedding matrix $\mathrm{TD} \in \mathbb{R}^{4 \times d_t}$ represents the embeddings generated by the large language model for text understanding difficulty in programming problems, with dimensions $4 \times d_t$, where 4 corresponds to the four levels of text understanding difficulty and d_t is the dimension size of the text understanding difficulty embeddings. The student code submission response is represented by the embedding matrix $\mathrm{R} \in \mathbb{R}^{2 \times d_r}$, with dimensions $2 \times d_r$, where 2 indicates whether the submitted code was accepted by the system (1 for accepted, 0 for rejected), and d_r is the dimension size of the response embeddings.

The DPKT framework

In this section, we will delve into the DPKT framework proposed in this paper, the structure of which is illustrated in Fig. 2. This framework is divided into four main parts. The first part is the difficulty assessment layer, which assesses the text understanding difficulty and knowledge concept difficulty of programming problems using a large language model, generating relevant difficulty level information. The second part is the semantic perception layer, which utilizes the large language model to extract semantic information about difficulty from the problems, constructing a relationship graph between problem difficulty and knowledge concepts. The third part is the knowledge update layer, which enhances and updates knowledge based on the information from the first two layers, thereby better reflecting the students' grasp of knowledge. The fourth part is the prediction layer, which uses the updated knowledge information to predict and assess students' knowledge state. Next, we will provide a more detailed discussion and explanation of these four modules.

Difficulty assessment layer

Traditional programming problem difficulty labeling primarily relies on the accuracy of each problem[50]. This method depends on a large amount of student submission data, leading to inaccurate labeling for new or infrequently used problems. Accuracy assessments often overlook the details and complexities of the problems and cannot reflect real-time changes in problem difficulty or student abilities[46]. Furthermore, traditional methods

Symbol	Description
d_1	The knowledge concept belongs to the beginner level of programming, such as basic syntax, variable declaration, simple control structures, etc.
d_2	Involves more complex logic, such as array manipulation, recursion, basic algorithms, etc.
d_3	Requires the application of advanced programming skills and algorithms, such as dynamic programming, graph algorithms, concurrent programming, etc.
d_4	Focused on advanced scenarios in algorithm competitions or real-world development, such as distributed systems, memory optimization, multithreading, etc.
u_1	The problem description is clear and concise, easy for students to understand, requiring no additional background knowledge.
u_2	The problem is slightly more complex, potentially involving multiple steps or requiring students to combine certain background knowledge.
u_3	The problem includes multi-layered logical descriptions or complex backgrounds, possibly necessitating strong reading comprehension skills from students.
u_4	The problem involves a large number of technical terms or implicit assumptions, requiring students to have a solid knowledge base and strong reasoning skills to understand.

Table 2. Difficulty level description.

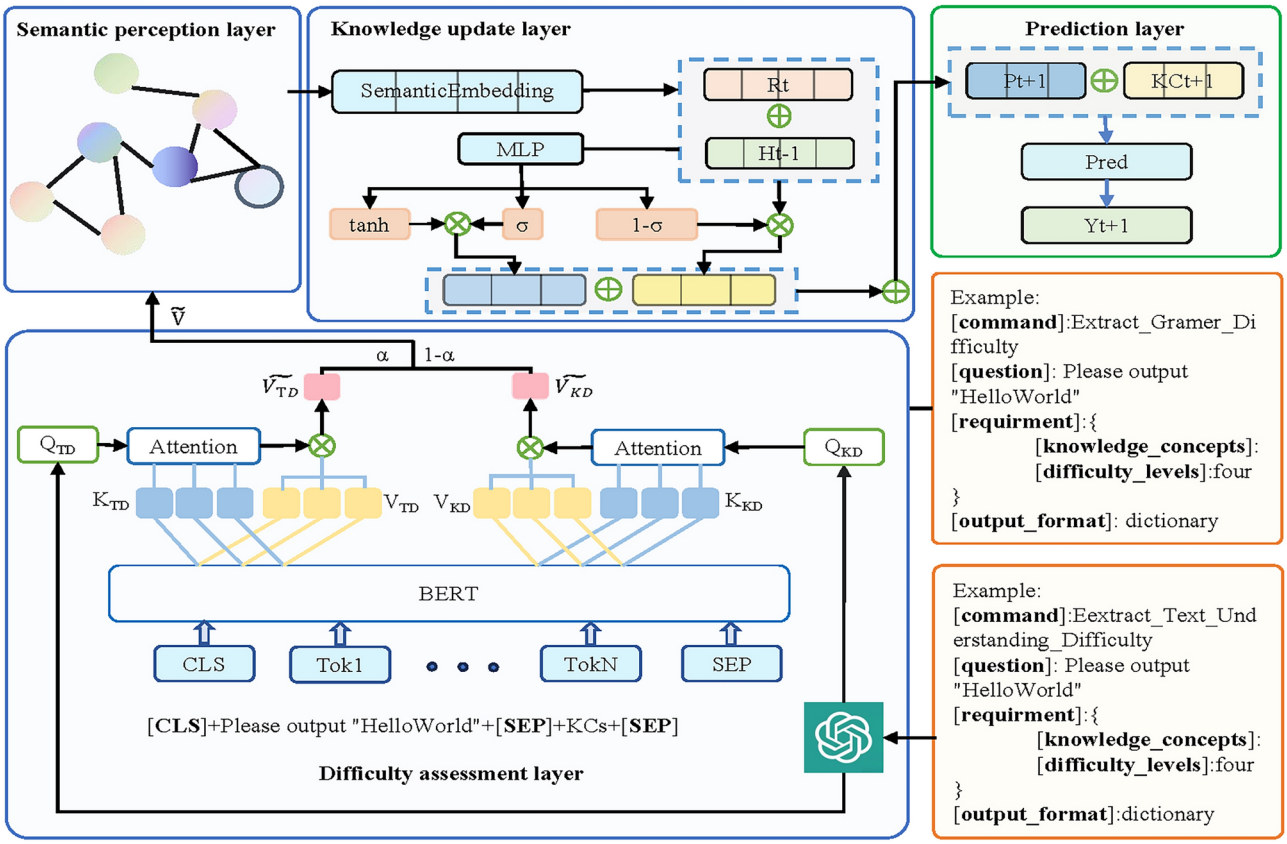

Fig. 2. The core architecture of the DPKT model involves using a large language model to annotate the text understanding difficulty and knowledge concepts difficulty of programming tasks. The task difficulty is then incorporated as one of the input conditions for predicting the student's knowledge state at the next time step in the knowledge tracing model.

are mainly based on historical performance, failing to effectively capture the deep structure of programming knowledge. The method proposed in this paper, which utilizes a large language model to label the knowledge concepts and text understanding difficulty of programming problems, is more resource-efficient and accurate. It fully leverages the powerful coding and comprehension capabilities of large language models, providing a comprehensive understanding of knowledge concepts and levels[36]. For evaluating the difficulty of knowledge concepts in programming problems, we adopted the following prompt approach:

$$A = \{problem\_id, text\_description, L_{TD}, L_{KD}, KCs\} \tag{1}$$

Where: A is the complete difficulty dictionary for the problems, $L_{KD} \in D$ represents the difficulty level of the knowledge concepts for the problem identified by $problem\_id$, $L_{TD} \in D$ indicates the text understanding difficulty level for the problem identified by $problem\_id$, KCs are the knowledge concepts involved in the problem.

To effectively integrate the information from the feature set A and provide a unified representation for subsequent model training and prediction, we convert the set A into an embedding matrix Q. This transformation not only captures the relationships among the various features but also represents rich semantic information through high-dimensional vectors, thereby enhancing the model's expressive and generalization capabilities.

Thus, the embedding matrix Q is constructed as:

$$Q = \begin{bmatrix} f_{one\_hot}(problem\_id) \\ f_{word2vec}(text\_description) \\ f_{integer}(L_{TD}) \\ f_{integer}(L_{KD}) \\ f_{one\_hot}(KCs) \end{bmatrix} \tag{2}$$

In this way, we can represent various aspects of the problems more comprehensively, improving the accuracy and effectiveness of subsequent analyses.

We use a large language model to pre-train the difficulty levels of knowledge concepts and text understanding difficulty for programming problems. The pre-training data is obtained through manual annotation. We invited ten experienced teachers in the field of computer education to assess the knowledge concept difficulty of

selected programming problems: $KD = \{d_1, d_2, d_3, d_4\}$ and the text understanding difficulty of the problems: $TD = \{u_1, u_2, u_3, u_4\}$. Each teacher provided ratings for different difficulty dimensions of each problem. Ultimately, we conducted a weighted average of all teachers' ratings to generate a comprehensive difficulty score for each problem. Let the score for the i problem be S_i, where S_i^j represents the rating given by the j teacher for the i problem (including both knowledge concept difficulty and text understanding difficulty). The final score for each problem can be expressed as:

$$S_i = \frac{1}{N} \sum_{j=1}^{N} w_j S_i^j \tag{3}$$

In the equations, N is the total number of teachers (in this case, 10), and w_j is the weight assigned to the j teacher, which is set based on the teacher's professional background, experience, and scoring consistency. Specifically, among the 10 teachers, there are 3 senior teachers, 3 intermediate teachers, and 4 junior teachers. Due to their extensive experience and professional level, senior teachers have a higher base weight, and after adjustment for consistency, their weight values are 0.144, 0.12, and 0.144, respectively. The base weight for intermediate teachers is slightly lower than that for senior teachers, with weight values of 0.10, 0.08, and 0.10, respectively. Junior teachers have a relatively lower base weight, with weight values of 0.064, 0.08, 0.096, and 0.064, respectively. This method of weight distribution takes into account both the professional level and scoring stability of the teachers, more accurately reflecting the reference value of each teacher's scores[51]. The calculations for knowledge concept difficulty scores and text understanding difficulty scores can be expressed as: $KD_i = \{d_1, d_2, d_3, d_4\}$, $TD_i = \{u_1, u_2, u_3, u_4\}$

Therefore, the final comprehensive difficulty scores can be expressed as:

$$KD_i^{\text{final}} = \frac{1}{N} \sum_{j=1}^{N} w_j KD_i^j \tag{4}$$

$$TD_i^{\text{final}} = \frac{1}{N} \sum_{j=1}^{N} w_j TD_i^j \tag{5}$$

Here, the value ranges for the knowledge concept difficulty score matrix KD and the text understanding difficulty score matrix TD are $KD_i^j, TD_i^j \in [1, 4]$, for each i (problem index) and j (the j-th teacher). By using a weighted average approach, we can effectively integrate the professional assessments from different teachers, thereby providing a more objective and accurate difficulty score for each programming problem.

Through the pre-training of knowledge concept difficulty and text understanding difficulty using the large language model, we obtain the embedding matrix Q. We further extract contextual information from programming problems and knowledge concepts using the self-attention mechanism of the CodeBERT model, generating high-quality embedding representations. These embeddings can capture the semantic relationships within the text, thus providing rich information for subsequent processing.

The K and V matrices output by CodeBERT are used for attention calculations, where K represents the keys (knowledge concepts and problem features), and V represents the values (information associated with K). The attention mechanism dynamically weights the importance of different information by calculating the relationships among Q,K and V. Q represents the query of the current input, K consists of the features to be referenced, while V contains the values associated with K.

Through attention calculations, the model can focus on the knowledge concepts or semantic information most relevant to the current input. The attention mechanism helps the model understand the relationships between different knowledge concepts and problems, making the final output more semantically relevant, thereby enhancing the assessment of students' knowledge state. The calculation process is as follows:

$$\tilde{v}_{\text{TD}} = \text{softmax}(F(K_{\text{TD}}, Q_{\text{TD}})) \cdot V_{\text{TD}} \tag{6}$$

$$\tilde{v}_{\text{KD}} = \text{softmax}(F(K_{\text{KD}}, Q_{\text{KD}})) \cdot V_{\text{KD}} \tag{7}$$

The generated V_{TD} and V_{KD} are fused, with the calculation formula as follows:

$$\tilde{V} = \alpha V_{\text{TD}} + (1 - \alpha) V_{\text{KD}} \tag{8}$$

Here, $F(K, Q)$ represents the similarity function in the self-attention mechanism, α is a trainable fusion weight, V_{TD} and V_{KD} are the attention results for knowledge concept difficulty and text understanding difficulty, respectively.

Semantic perception layer

Through the perception of problem text understanding difficulty and knowledge concept difficulty in the difficulty perception layer, we need to further understand the close relationship between the problem at time t and the problems solved by the student from time 1 to $t-1$. This forms a spatiotemporal representation of the student's knowledge state, providing a more accurate assessment of the student's knowledge state.

We treat knowledge concepts and programming problems as each node $\tilde{V}_{\mathrm{i}}$, aggregating the information of each node's neighboring nodes through the self-attention mechanism to generate new node feature representations. This helps capture the relationships and contextual information between nodes. By aggregating the information from neighboring nodes to update each node's state, we reflect the student's mastery of different knowledge concepts. The model allows for the dynamic adjustment of contributions from different neighboring nodes, thus determining how to aggregate information based on the actual situation.

We construct the adjacency matrix B based on $\tilde{V}$. In this graph, the node $\tilde{V}_{\mathrm{i}}$ represents the features of each programming problem, and the relationship between two nodes indicates that the two programming problems are semantically similar or have a dependency relationship.

$$\mathrm{Sim}(\tilde{V}_{\mathrm{i}}, \tilde{V}_{\mathrm{j}}) = \frac{\tilde{V}_{\mathrm{i}} \cdot \tilde{V}_{\mathrm{j}}}{\|\tilde{V}_{\mathrm{i}}\| \|\tilde{V}_{\mathrm{j}}\|} \tag{9}$$

Among them, $\mathrm{Sim}(\tilde{V}_{\mathrm{i}}, \tilde{V}_{\mathrm{j}})$ represents the cosine similarity between node $\tilde{V}_{\mathrm{i}}$ and node $\tilde{V}_{\mathrm{j}}$, where $\tilde{V}_{\mathrm{i}}$ and $\tilde{V}_{\mathrm{j}}$ are the feature vectors of the two nodes, and $\|\cdot\|$ denotes the vector norm.

First, we construct the similarity matrix S, where $S[i][j]$ indicates the similarity between node i and node j :

$$S[i][j] = \mathrm{similarity}(\tilde{V}_{\mathrm{i}}, \tilde{V}_{\mathrm{j}}) \tag{10}$$

According to the similarity matrix S, a threshold λ is set. The threshold λ is a trainable key parameter that determines the connection between two nodes. The adjacency matrix B is constructed based on this threshold:

$$B[i][j] = \begin{cases} 1 & \text{if } S[i][j] \geq \lambda \\ 0 & \text{if } S[i][j] < \lambda \end{cases} \tag{11}$$

According to the adjacency matrix, for each pair of adjacent nodes i and j, we calculate the attention coefficient α_i^j:

$$\alpha_i^j = \mathrm{softmax}_j \left(\frac{\boldsymbol{a}^T [W\boldsymbol{v}_i \parallel W\boldsymbol{v}_j]}{\sqrt{d}} \right) \tag{12}$$

Here, $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ are the feature vectors of nodes i and j, W is a learnable weight matrix, $\boldsymbol{a}$ is the weight vector used for attention calculation, $\parallel$ denotes the concatenation operation, and d is the feature dimension.

The attention coefficient α_i^j is used to weight the features of adjacent nodes; for each node i, its updated feature representation $\boldsymbol{h}_i'$ is obtained by weighted aggregation of the information from adjacent nodes:

$$\boldsymbol{h}_i' = \sigma \left(\sum_{j \in N(i)} \alpha_{ij} W \boldsymbol{v}_j \right) \tag{13}$$

Here, $N(i)$ denotes the set of adjacent nodes connected to node i, σ is the ReLU non-linear activation function.

The adjacency matrix defines the connection relationships between nodes and guides the calculation of attention coefficients, influencing the node feature update process. Through Graph Attention Networks (GAT), the DPKT model we designed can effectively propagate and aggregate the features and contextual information of problems within the graph structure, aiding in a more accurate assessment of the student's learning state.

Finally, the new feature matrix of the nodes is output:

$$\boldsymbol{H}' = \mathrm{concat}(\boldsymbol{h}_1', \boldsymbol{h}_2', \ldots, \boldsymbol{h}_N') \tag{14}$$

Each node in the matrix $\boldsymbol{H}'$ represents the integration of information from that node and its adjacent nodes, reflecting the student's status in learning programming knowledge and skills, thus providing rich contextual information for subsequent knowledge tracing and state assessment.

Knowledge update layer

Different difficulty levels of exercises have varying impacts on students' knowledge state, and the influence of exercise difficulty on knowledge state is dynamic[48]. The knowledge update layer allows for the dynamic updating of students' mastery of exercises at different difficulty levels and their overall knowledge state. Specifically, we first concatenate the semantic features h_t' at time t with the student's code submission response R_t and the previous hidden state H_{t-1} to obtain a comprehensive information vector Z_t, The specific calculation formula is as follows:

$$Z_t = h_t' \oplus R_t \oplus H_{t-1} \tag{15}$$

In this context, $\oplus$ denotes the vector concatenation operation.

The students' knowledge state are updated by practicing exercises of varying difficulties, and the knowledge update gate G_t dynamically adjusts the students' mastery of knowledge at different difficulty levels, facilitating effective information integration. The calculation formula is as follows:

$$G_t = \sigma(W_g \cdot Z_t + b_g) \tag{16}$$

Where, W_g is the weight matrix for the update gate, b_g is the bias vector, and $\sigma(\cdot)$ is the Sigmoid activation function.

To ensure the model can dynamically update knowledge state, we integrate current input information with historical data to enhance and update the knowledge state. The calculation formula is as follows:

$$h_t = (1 - G_t) \cdot Z_t + G_t \cdot \tanh(W_h \cdot Z_t + b_h) \tag{17}$$

Where W_h is the weight matrix for knowledge state updates, b_h is the bias vector, and represents element-wise multiplication.

Through the knowledge update layer, the model integrates the latest and historical semantic information from exercises and students' knowledge state, allowing for dynamic updates during the students' learning process.

Output layer

The knowledge mastery state of the student at the current time h_t is obtained. To better predict the student's knowledge performance at the next time step, the next programming problem and relevant knowledge points are fused into the current knowledge state for output. The calculation formula is as follows:

$$y_t = h_t \oplus P_{t+1} \oplus KC_{t+1} \tag{18}$$

$$y_{t+1} = \sigma(W_y \cdot y_t + b_y) \tag{19}$$

Where, W_y is the learnable weight parameter, b_y is the bias term, and $\sigma(\cdot)$ is the Sigmoid activation function.

Objective function

During the model training process, we employed the Adam optimization algorithm to accelerate convergence and enhance robustness through adaptive learning rates. Since knowledge tracing can be regarded as a classification problem, we selected the binary cross-entropy loss function, which effectively measures the discrepancy between predictions and true labels. Specifically, the objective function is defined as:

$$L = -\sum_{t=1}^{T} [R_t \log y_t + (1 - R_t) \log(1 - y_t)] + \mu_\theta ||\theta||^2 \tag{20}$$

Among them, θ represents all trainable parameters in this model, including learning rate, batch size, number of training epochs, and regularization techniques. The specific settings are as follows: The initial learning rate is set to 0.001, and a learning rate decay strategy is adopted, where the learning rate is reduced to 0.1 of the original after a certain number of training epochs. Batch size: The batch size is set to 32, which is the optimal value found in experiments, ensuring model training efficiency while avoiding overfitting. Number of training epochs: The number of training epochs is set to 50, and the optimal number of training epochs is selected through cross-validation. Regularization: L2 regularization technology is used, with a regularization coefficient set to 0.01 to prevent model overfitting. With these settings, we can more effectively optimize model parameters and improve model performance and generalization ability. μ_θ is the regularization hyperparameter used to prevent model overfitting. The regularization term $\mu_\theta \|\theta\|^2$ penalizes the model's complexity by adding the L2 norm of the parameters to the loss function, thereby improving the model's generalization ability. To ensure the reproducibility and transparency of the model, we have made detailed adjustments and verifications of the regularization hyperparameter μ_θ in the experiment. Specifically, the value of μ_θ is the optimal value obtained by grid search within the interval $[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$. When μ_θ is set to 10^{-3}, the model achieves the best performance on the validation set, striking a good balance between predictive accuracy and generalization ability. This selection process ensures the model's stability and generalization ability across different datasets.

Experiments

In the experimental section, we first provide a detailed overview of the dataset used, including its source, structure, and statistical information. We describe the data preprocessing, feature extraction, and other processes to ensure that the model can efficiently utilize this data. Next, we compare the baseline model with our proposed knowledge tracing model based on large language models in terms of their ability to predict student performance, showcasing their performance across various evaluation metrics. Furthermore, we conduct a series of experiments to explore the different modules of the model and their impact on overall performance, revealing the advantages and potential of large language models in knowledge tracing tasks. The key research questions we aim to investigate are as follows:

- **RQ1**: Does our proposed knowledge tracing model based on large language models demonstrate superior performance in predicting students' future performance?
- **RQ2**: How does the difficulty of programming problems affect the modeling of students' knowledge state?
- **RQ3**: How much prior knowledge should be provided for large language models to achieve optimal results?
- **RQ4**: How does the difficulty level generated by large language models compare to the difficulty level annotated by humans?

- **RQ5**: What is the impact of different modules in the model (such as difficulty assessment and semantic awareness) on the knowledge tracing results?
- **RQ6**: What is the effect of problems at varying difficulty levels on the model's performance?
- **RQ7**: How does the knowledge tracing model based on large language models understand the code submitted by students?

Through the exploration of these questions, we not only validate the model's effectiveness across different tasks but also investigate its limitations and areas for optimization, aiming to provide guidance for future related research.

Datasets

To validate the effectiveness of DPKT, we selected the Project CodeNet dataset for testing. Project CodeNet is a large and diverse programming dataset that contains approximately 14 million code samples, covering over 50 programming languages. It is particularly well-suited for tasks such as programming problem difficulty modeling and student knowledge state prediction[49]. Table 3 presents specific statistical information filtered after data cleaning. A notable feature of Project CodeNet is that it provides detailed metadata for each programming problem, such as code execution time, memory usage, and submission status (whether it passed or failed). This rich metadata provides a solid foundation for our difficulty analysis based on large language models. In CodeNet, 94.6% of submissions are made in the following four programming languages: $C++$, *Python*, *Java*, and *C*. In this study, we focus on these four languages. To more accurately assess students' programming skills, this research utilizes programming problems combined with the execution results of the code (such as whether it compiled successfully and whether it was correct) to trace students' knowledge state. Additionally, to handle exercise sequences of varying lengths, we set the maximum sequence length for analysis to 50. For sequences exceeding this length, we split them into multiple parts, while shorter sequences are standardized through zero padding. This approach ensures that the model can effectively leverage the diversity of different problems and languages for difficulty perception and student state prediction.

Experimental setup

To comprehensively assess the performance of the DPKT model, we conducted an in-depth comparison with several mainstream baseline models. To ensure fairness and scientific rigor in the comparison, all baseline models were reproduced under the same experimental conditions, with precise tuning of their key parameters to ensure each model operated in its optimal state. These baseline models encompass existing mainstream knowledge tracing (KT) methods, representing various learning strategies and network architectures. For performance evaluation, we selected AUC (Area Under Curve) and ACC (Accuracy) as the primary metrics, which allow for a more comprehensive assessment of the overall performance of the models and directly measure their accuracy in actual prediction tasks, particularly in predicting accuracy under different student knowledge state. The experimental methods strictly followed the principle of reproducibility, with all models running on a Linux server equipped with a Tesla V100 GPU, and multiple repetitions were conducted to minimize the impact of random factors on the experiments, ensuring the reliability and robustness of the results. In this study, we employed 5-fold cross-validation to assess the model's performance, which ensures the stability of the evaluation results while reducing computational costs. During the experimental process, we thoroughly tested the model through cross-validation and extensively verified it with different types of datasets, such as real student learning datasets from educational platforms.The LLM model we utilized is GPT-4o, which is a multimodal large language model, and the BERT model is CodeBERT, an optimized BERT model for programming languages[52]. The selection and application of these models further enhance the depth and breadth of our research.

- **DKT**[1] proposes using recurrent neural networks (RNNs) to model students' knowledge state. By not relying on manual knowledge encoding, RNNs capture more complex student learning behaviors, enhancing the predictive performance of knowledge tracing tasks.
- **DKVMN**[8] is based on dynamic memory networks and introduces static key and dynamic value matrices to store knowledge concepts and the levels of knowledge mastered by students, respectively. This model automatically discovers and updates knowledge state, addressing the issue of manual concept annotation in traditional methods.
- **SAKT**[53] employs self-attention mechanisms to select relevant knowledge concepts from students' past activities, offering better handling of data sparsity issues. The model excels in sparse data environments, significantly enhancing predictive performance.
- **EKT**[7] uses self-attention mechanisms to select past activities related to specific knowledge concepts, mitigating the impact of data sparsity on the model. The model demonstrates outstanding performance on multiple real datasets, improving AUC and prediction outcomes.

Language type	Number of problems	Number of students	Number of submissions
C++	1218	678	35,245
Python	1097	689	29,330
Java	1052	487	22,650
C	963	512	19,400

Table 3. Statistics of Selected Languages from Project CodeNet Dataset.

- **Code-DKT**[19] is designed for programming tasks and extends the DKT model with attention mechanisms, automatically extracting code features relevant to problems and enhancing prediction performance in programming tasks through these features.
- **DKT+**[54] addresses issues present in DKT (such as input reconstruction failure and inconsistent knowledge level predictions) by introducing two regularization terms to enhance the model's predictive consistency and stability, thereby improving DKT's performance.
- **DIMKT**[50] introduces a problem difficulty matching mechanism to dynamically assess the relationship between students' knowledge state and problem difficulties, adjusting knowledge state predictions for problems of varying difficulty.
- **QIKT**[55] refines the modeling of students' knowledge state through problem-centric approaches and proposes a prediction layer based on item response theory, enhancing the model's interpretability and predictive accuracy.
- **TCKT**[56] incorporates time and causality enhancement mechanisms to improve the consistency and predictive accuracy of knowledge state through self-attention methods and knowledge forgetting and acquisition gates.
- **THKT**[57] introduces a hierarchical guidance mechanism to dynamically assess student feedback at different problem levels, optimizing predictions of students' knowledge mastery and enhancing the model's interpretability and hierarchical recognition capabilities.

Performance prediction (RQ1)

To validate the effectiveness of our proposed DPKT model in tracking students' knowledge state and predicting future performance, we conducted a comparative analysis with various classic baseline models. In the experiment, we used AUC (Area Under the Curve) and ACC (Accuracy) as the main evaluation metrics. AUC measures the model's ability to distinguish between positive and negative samples, while ACC represents the overall predictive accuracy of the model. Table 4 shows the average performance of different models on the AUC and ACC metrics. The choice to report the average values rather than the standard deviation values is based on the experimental design, where the consistency of experimental conditions leads to minimal variation in results. Specifically, by repeating the experiments multiple times, we ensured that the data division and experimental conditions were the same each time, thus making the average values of AUC and ACC highly stable. Compared to traditional knowledge tracing models such as DKT and DKVMN, the DPKT model outperformed across all programming language datasets. As show in Table 4, this indicates that the DPKT model is better at capturing changes in knowledge concepts and difficulty levels within programming problems. Models like EKT and THKT, which encode problems, performed well on certain datasets; for instance, THKT achieved an AUC of 75.2% on the C dataset, but their overall performance was still inferior to that of the DPKT model. By introducing graph attention mechanisms, DPKT enables finer semantic integration and knowledge state updates, significantly enhancing prediction accuracy. A core advantage of the DPKT model lies in its automated annotation of knowledge points and difficulty levels for programming problems using large language models. This automation improves the model's generalization ability, allowing it to capture students' learning trajectories more precisely across different difficulty levels, leading to more accurate predictions of their knowledge state. This design enables DPKT to flexibly adjust its predictions based on the content and difficulty of programming problems, significantly enhancing the model's predictive performance.

Through comparative experiments with baseline models, our DPKT model demonstrated strong performance across multiple programming language datasets, especially excelling in AUC and ACC metrics on C++ and C datasets. Additionally, the DPKT model effectively enhanced the dynamic tracking of students' knowledge state through automated annotation and multi-level attention mechanisms. The experimental results confirm the advantages of the DPKT model in programming knowledge tracing and predicting student future performance, providing new research directions for personalized teaching and knowledge assessment.

Methods	C++_AUC	C++_ACC	C_AUC	C_ACC	Java_AUC	Java_ACC	Python_AUC	Python_ACC
DKT[1]	70.9	69.7	71.7	66.5	71.8	69.7	74.0	68.0
DKVMN[8]	71.5	69.9	66.9	64.2	71.8	70.3	73.8	69.0
SAKT[53]	71.9	70.6	72.5	67.0	73.0	69.9	74.5	68.8
EKT[7]	72.0	70.0	73.2	69.8	75.0	71.7	73.3	67.0
Code-DKT[19]	71.2	69.9	71.8	67.4	71.8	67.9	72.5	69.1
DKT+[54]	73.2	71.0	72.9	69.7	74.9	72.6	75.5	68.5
DIMKT[50]	72.5	71.7	75.1	68.7	73.5	71.8	75.9	68.9
QIKT[55]	73.4	71.6	72.4	67.3	72.2	70.8	73.8	69.5
TCKT[56]	73.6	70.9	73.1	68.4	72.6	71.8	76.0	67.7
THKT[57]	73.8	71.8	75.2	70.0	74.5	72.5	74.9	70.0
DPKT	**76.2**	**74.5**	**75.7**	**71.5**	**75.4**	**72.9**	**76.8**	**68.1**

Table 4. Comparison of the AUC and ACC performance (%) of various knowledge tracing methods on the C++, C, Java, and Python datasets. The best results are highlighted in **bold**, and the second-best results are underlined.

Knowledge state modeling (RQ2)

In programming learning, assessing students' knowledge state is crucial for effective knowledge tracing. Programming problems often involve multiple knowledge points with varying levels of difficulty and complexity. Capturing students' knowledge state in programming tasks through the DPKT model becomes key to evaluating their programming abilities. As shown in Fig. 3, the DPKT model illustrates changes in students' knowledge state across multiple knowledge points, especially as the difficulty and complexity of problems gradually increase, reflecting dynamic variations in their mastery of these points.

First, overall, students' knowledge state change with the increase in practice involving different difficulty levels. Second, we find that problems of varying difficulty impact students' knowledge state differently. For instance, the knowledge state growth for problems 9 and 10 differs significantly from that for problems 18 and 19. Medium-difficulty problems (like 9 and 10) provide solid consolidation opportunities, ensuring students steadily improve their mastery while gradually adapting to more complex tasks. High-difficulty problems (like 18 and 19), while challenging, can greatly enhance students' knowledge state, particularly when related to prerequisite knowledge; the benefits of practicing high-difficulty problems are even more pronounced.

Third, the DPKT model can also explain the impact of problem difficulty on knowledge state by analyzing the relationships between knowledge concepts. In the experiment, we found that the prerequisite relationships between knowledge concepts affected the degree to which students mastered the relevant knowledge concepts, with high-difficulty problems having a greater impact on the mastery of these related knowledge concepts. This demonstrates that the DPKT model effectively captures the dynamic changes in students' knowledge state across different knowledge concepts and problem difficulties, and it uses difficulty analysis to help explain variations in students' knowledge mastery. The DPKT model provides strong data support for knowledge tracing in programming learning. By integrating and analyzing multidimensional features such as problem difficulty, knowledge concepts, and student responses, the model reveals patterns of how students' knowledge state change with increasing difficulty. This model offers a solid basis for optimizing personalized learning pathways.

The impact of data proportion fed into LLM on performance (RQ3)

To verify the impact of prior knowledge from different data volumes on model performance, we designed an experiment to segment datasets from four programming languages (C++, C, Python, Java) and input them into the model for training and evaluation. The experimental results, as shown in Fig. 4, reveal the following trends:

First, the volume of data from different programming languages affects model performance (AUC and ACC) differently. As the training data volume increases, the model's AUC and ACC metrics generally show an upward trend, but this growth is not linear. For example, in the Python dataset, the AUC significantly improves with initial increases in data volume, but the growth rate gradually slows when the volume reaches 3/4 or more. A similar phenomenon occurs in the C++, C, and Java datasets, where the increases in AUC and ACC tend to plateau as data volume grows.

Second, specifically, C++ exhibits the fastest growth rate when the data volume increases from 1/4 to 1/2, reflecting its sensitivity to data volume. As a powerful programming language, C++ has high flexibility and complexity, making it suitable for handling large amounts of complex algorithms and data structures, which may lead to significant AUC growth in the early stages. In contrast, Java's AUC growth is relatively slow, particularly during the 1/4 to 1/2 data volume stage, indicating weaker model adaptability to this language. Java is mainly used for enterprise-level applications and large-scale systems, and its strict syntax rules and relatively steep learning curve may limit the model's effectiveness with smaller data volumes. Comparatively, C and Python perform at a moderate level across different data volume stages, with Python starting at a higher point and overall performing better. The simplicity and widespread applicability of Python may enhance its learnability in

Fig. 3. The learning sequences and knowledge state evolution of two students across knowledge components (KCs) with different levels of difficulty. The difficulty levels are divided into four grades, with higher numerical values representing greater difficulty.

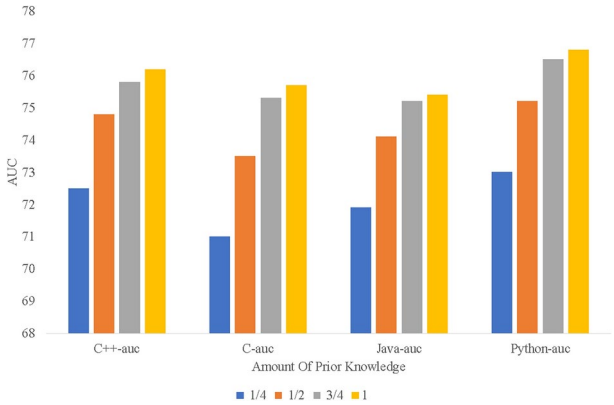

Fig. 4. By inputting different proportions of raw data into the LLM to generate knowledge concepts, we can observe its impact on the model's AUC performance.

projects, especially in data science and artificial intelligence, further facilitating the model's rapid understanding of its characteristics.

Based on the results of this experiment, we draw preliminary conclusions regarding the optimal amount of prior knowledge for large language models: data volume, the characteristics of programming languages, and the amount of prior knowledge collectively influence the model's performance. Future research could further explore how to optimize the allocation of prior knowledge according to different programming languages and datasets to enhance model performance and achieve broader applicability.

Comparison of expert annotations of problem difficulty with results generated by the large language model (RQ4)

To evaluate the differences and consistency between difficulty annotations generated by large language models and those made by human annotators, we designed a comparative experiment involving ten experienced computer education teachers. These teachers manually annotated the text understanding difficulty and knowledge concepts difficulty of problems in four programming languages. To ensure the comparability of the results, all annotations used a uniform scale of 1 to 4, representing low, low-medium, medium-high, and high difficulty (with 1 being the lowest and 4 the highest).

We matched the annotations generated by the large language model with those of the teachers and assessed the consistency between the two using both quantitative and qualitative analysis methods. The matching approach primarily involved calculating the difference in difficulty levels for each question, categorizing the consistency into three levels: fully consistent, acceptable deviation, and inconsistent. Subsequently, we quantified the consistency scores and measured the overall performance of the model using the average consistency score. In addition, we performed qualitative analysis of difficulty trends based on the data and compared the time costs involved, providing a comprehensive assessment of the large model's potential in large-scale annotation tasks. The specific methods are as follows:

1. Matching Algorithm: For each programming problem i, the teacher's annotated text understanding difficulty is T_i^{text} and the knowledge concepts difficulty is T_i^{syntax}, while the corresponding difficulties generated by the large language model are M_i^{text} and M_i^{syntax}. The difference in text understanding difficulty and knowledge concepts difficulty for each problem is calculated as:

$$\Delta_i^{\text{text}} = \left| T_i^{\text{text}} - M_i^{\text{text}} \right| \tag{21}$$

$$\Delta_i^{\text{syntax}} = \left| T_i^{\text{syntax}} - M_i^{\text{syntax}} \right| \tag{22}$$

Based on the size of the difference Δ_i,the matching situations are categorized into three types[58,59]: Perfect Match: $\Delta_i^{\text{text}} \leq 0.5$ and $\Delta_i^{\text{syntax}} \leq 0.5$. Acceptable Deviation: $0.5 < \Delta_i^{\text{text}} \leq 1.0$ or $0.5 < \Delta_i^{\text{syntax}} \leq 1.0$. Inconsistent: $\Delta_i^{\text{text}} > 1.0$ or $\Delta_i^{\text{syntax}} > 1.0$.

2. Consistency Score: To quantify the consistency of each problem in terms of text understanding and knowledge concepts difficulty, we introduce the consistency score S_i:

$$S_i = 1 - \left(\frac{\Delta_i^{\text{text}} + \Delta_i^{\text{syntax}}}{2} \right) \tag{23}$$

Here, the range of S_i is from 0 to 1, where 1 indicates perfect consistency and 0 indicates inconsistency.

3. Average Consistency Score: We calculate the average consistency score $\bar{S}$ for all problems to measure overall consistency:

$$\bar{S} = \frac{1}{N}\sum_{i=1}^{N} S_i \tag{24}$$

Here, N is the total number of problems. The results are shown in the table. The calculation formula for the Kappa value is:

$$Kappa = \frac{P_o - P_e}{1 - P_e} \tag{25}$$

Where P_o is the observed consistency, which is the proportion of problems annotated consistently by both the experts and the large model; P_e is the expected consistency, which is the consistency under random assignment. The Kappa value ranges from [−1, 1][60,61], where:

1 : Perfect consistency.
0 : Consistency equivalent to random selection.
Negative values : Consistency below random levels.
Specifically,
0.01–0.20 : Almost no consistency.
0.21–0.40 : Weak consistency.
0.41–0.60 : Moderate consistency.
0.61–0.80 : Strong consistency.
0.81–1.00 : Almost complete consistency.

The results, as shown in Fig. 5, indicate that when analyzing the difficulty annotations for different programming languages, C++ and Python perform better than C and Java. Specifically, C++ has 75 problems matched for text understanding difficulty, with a Kappa value of 0.57; for knowledge concepts difficulty, the number of matched problems is 80, with a Kappa value reaching 0.64, demonstrating high consistency. Python has 80 matched problems for text understanding difficulty, with a Kappa value of 0.65, and 85 matched problems for knowledge concepts difficulty, achieving a Kappa value of 0.73, indicating that the model can effectively capture the difficulty characteristics of Python. In contrast, C has 70 matched problems for text understanding difficulty, with a Kappa value of 0.50, and 65 matched problems for knowledge concepts difficulty, with a Kappa value of only 0.44, showing lower consistency. Additionally, Java's Kappa values for text understanding and knowledge concepts difficulties are 0.53 and 0.47, respectively, overall performing worse than C++ and Python. These data indicate that the large language model generally exhibits good annotation consistency when labeling the difficulties of different programming languages, but there are also certain discrepancies. These differences reflect that the model's annotation effectiveness is influenced by the characteristics of the programming languages, laying the groundwork for future work.

Model component effectiveness evaluation (RQ5)

The DPKT model utilizes a large language model to accurately annotate the knowledge concepts and difficulty levels of programming problems, integrating this difficulty-related semantic information into the process of updating students' knowledge state. To validate the key roles of the large language model, BERT model, and graph attention network in modeling student knowledge state within DPKT, we conducted experimental analyses based on datasets from four programming languages (C++, C, Java, Python) to assess the positive contributions of each component to the prediction of students' knowledge state, as shown in Table 5.

The large language model plays a crucial role in updating student knowledge state, especially when handling complex programming language datasets (such as Python), where its understanding of problem and knowledge

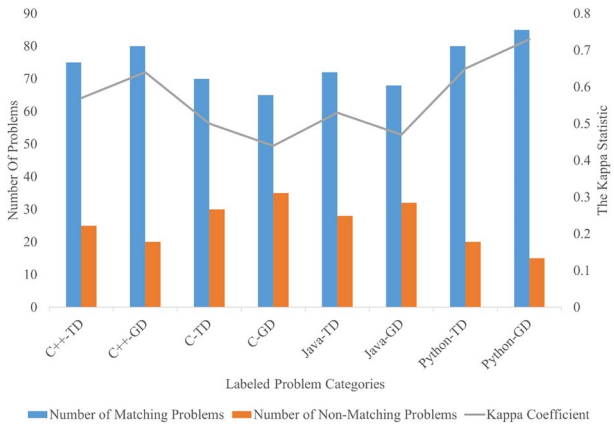

Fig. 5. Comparison of the Consistency between Human Annotations and LLMs Annotations for text Understanding Difficulty and Knowledge Concepts Difficulty of 100 Problems in Four Programming Languages.

Methods	C++_AUC	C++_ACC	C_AUC	C_ACC	Java_AUC	Java_ACC	Python_AUC	Python_ACC
DPKT w/o LLM	74.8	73.2	74.3	70.1	74.0	71.4	75.5	66.9
DPKT w/o BERT	73.5	72.1	73.1	69.0	73.0	70.5	74.1	65.4
DPKT w/o GAT	75.2	73.8	74.5	70.8	74.7	71.9	75.9	67.2

Table 5. Results of ablation studies on different programming language datasets.

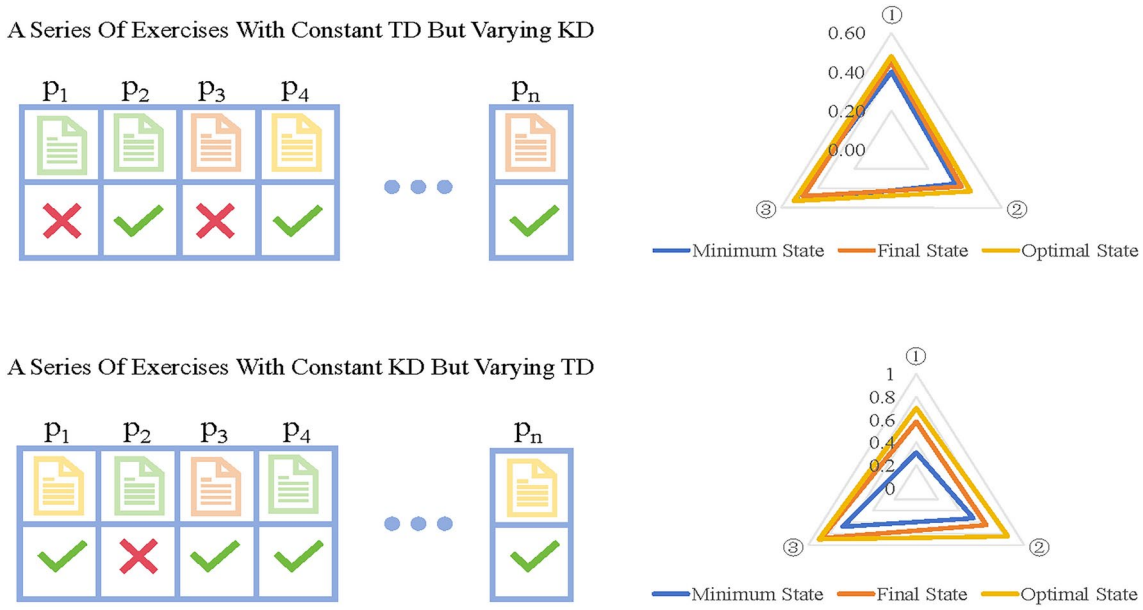

Fig. 6. After practicing a series of programming problems with varying levels of difficulty, the radar chart of changes in students' knowledge state.

concepts difficulty significantly enhances model performance. Meanwhile, the CodeBERT model is essential for capturing the semantic relationships between knowledge concepts and problems; its removal leads to a notable decline in model performance, underscoring its indispensability in modeling interactive relationships. Although the performance drop from removing the graph attention network is less pronounced than for the first two modules, it still plays a positive role in integrating multidimensional information and handling complex problem difficulties, indicating the collaborative effect of these modules in enhancing the overall capabilities of the model.

The difficulty information processed by the large language model provides more precise and detailed annotations of problem and knowledge concepts difficulties, laying a solid foundation for subsequent research. This accurate difficulty information not only aids in optimizing the design of personalized learning paths for students but also improves the accuracy of knowledge state updates, enhancing the efficiency of educational technology applications. Furthermore, the deep modeling of the semantic relationships between knowledge concepts and problems through the CodeBERT module further enhances the model's understanding of student learning states, increasing the overall reliability of predictions. The significance of the graph attention network in integrating multidimensional information enables effective consolidation of difficulty information from various sources, thereby improving the model's ability to handle complex problem difficulties. Therefore, the comprehensive research based on the large language model, CodeBERT, and graph attention network provides new insights and directions for future model optimization, personalized learning path design, and educational research, promoting the further development of educational technology.

Analysis of the role of difficulty factors in knowledge tracing models (RQ6)

Accurately assessing and tracking students' knowledge state is crucial for improving learning efficiency and developing personalized learning plans[1]. However, variations in text understanding difficulty and knowledge concepts difficulty across different problems can significantly affect students' performance and the predictive effectiveness of models. In this experiment, we analyzed the impact of text understanding difficulty and knowledge concepts difficulty on student knowledge state updates, leading to several important conclusions. We had students practice a series of programming problems with the same knowledge concepts difficulty but varying understanding difficulties, as well as problems with the same text understanding difficulty but differing knowledge concepts difficulties. Their minimum knowledge state, final knowledge state, and optimal knowledge state are illustrated in the radar chart in Fig. 6.

When knowledge concepts difficulty is the same, and students practice programming problems with different understanding difficulties, their knowledge state do not change significantly. Moderate text understanding

KCs: Loop	Predicted
1.Write a program that uses a for loop to print all even numbers from 1 to 10.	
Text Understanding Difficulty: 1 Knowledge Concepts Difficulty: 2	✔
2. Write a function that takes a list of integers as a parameter, uses a for loop to iterate through the list, and outputs all the even numbers greater than 5.	
Text Understanding Difficulty: 3 Knowledge Concepts Difficulty: 2	✔
3. Write a program that uses a for loop to iterate through a list containing multiple sublists. For each number in the sublists, if the number is even, add it to a cumulative total, and finally return the sum of all even numbers.	
Text Understanding Difficulty: 4 Knowledge Concepts Difficulty: 2	✖

Fig. 7. The impact of text understanding difficulty in textual problems with varying difficulty levels for the same knowledge concept on model prediction results.

difficulty helps students better understand the problems, enhancing their performance and facilitating updates to their knowledge state. However, if text understanding difficulty is too high, students need to invest more effort to understand the problems, which can negatively affect the performance of some students. As a result, the model may incorrectly assume that these students have not mastered the relevant knowledge, preventing a positive update to their knowledge state. Conversely, when text understanding difficulty is the same, and students practice programming problems with different knowledge concepts difficulties, their knowledge state show significant improvement. After correctly analyzing the problem requirements, students practicing with problems of lower knowledge concepts difficulty can boost their confidence, foster their interest in learning, and promote further in-depth study. However, the improvement in knowledge state may be limited. In contrast, problems with higher knowledge concepts difficulty effectively differentiate students of varying levels, prompting greater cognitive enhancement, but may also demotivate some students and reduce their interest in learning. Therefore, moderate understanding and knowledge concepts difficulties can maintain a balance between student performance and knowledge state updates while ensuring high predictive accuracy for the model.

Both text understanding difficulty and knowledge concepts difficulty play vital roles in the knowledge tracing process, influencing not only student performance but also the model's accuracy in predicting students' knowledge state. Optimal levels of understanding and knowledge concepts difficulty can enhance students' learning outcomes and enable the model to more accurately track updates in their knowledge state. In research, setting appropriate difficulty levels can help students gain a better learning experience while improving the model's predictive accuracy. Future studies could further explore the best application scenarios for different difficulty combinations in knowledge state updates, providing richer references for optimizing educational models.

Case study (RQ7)

Comprehension difficulty of problems is a key factor affecting the performance of student knowledge state prediction models[50]. Through the analysis of problems of varying difficulty, we found that both excessively high and low comprehension difficulty can significantly impact the accuracy of model predictions. When comprehension difficulty is not taken into account, the model may rely on relatively simple knowledge concepts difficulty to predict student performance. However, even with lower knowledge concepts difficulty, students may fail to complete problems due to high comprehension difficulty, leading to biased predictions. For example, a student encountered a series of problems based on "loop structures" where the comprehension difficulty was rated as 1 and the knowledge concepts difficulty as 2. The first problem had low comprehension difficulty, and the student was able to complete it successfully, with the model accurately predicting that the student had mastered the relevant knowledge. In contrast, the third problem had the same knowledge concepts difficulty as the previous one, but due to an increase in comprehension difficulty, the student faced challenges. However, due to the complexity of the problem's wording, the model failed to recognize this, incorrectly predicting that the student had mastered the knowledge concepts.

As shown in Fig. 7, when comprehension difficulty is too high, students often struggle to accurately understand the problem, failing to adequately demonstrate their knowledge mastery. In this case, even though the model predicts that the student can correctly complete the problem based on lower knowledge concepts difficulty, the understanding barrier leads to the student not completing the task as expected, resulting in the model incorrectly updating their knowledge state to "not mastered." This bias introduced by high comprehension difficulty demonstrates that relying solely on knowledge concepts difficulty for predictions is one-sided. In contrast, in problems with moderate comprehension difficulty, students can successfully understand the problem and respond correctly, allowing the model to accurately capture the student's knowledge state and update it to "mastered."

On the other hand, while problems with low comprehension difficulty can help build student confidence, if the problems are too simple, students may easily complete them but fail to effectively enhance their knowledge level. As a result, the model might misjudge the student's true learning state, leading to updates in knowledge state that lack reference value.

Therefore, considering comprehension difficulty plays an important corrective role in student knowledge state prediction models. By integrating comprehension difficulty with knowledge concepts difficulty, the model

can more comprehensively evaluate student learning performance. For instance, when the model predicts a problem with low knowledge concepts difficulty that the student should be able to complete, incorporating comprehension difficulty can correct potential biases. In situations where comprehension difficulty is high, even if the grammar is simple, the model can reasonably anticipate that the student might perform poorly due to unclear problem wording, thus avoiding erroneous updates to their knowledge state. Comprehension difficulty should not be overlooked in knowledge state prediction models. By introducing comprehension difficulty, the model can more accurately capture the student's true knowledge level, avoiding performance declines caused by imbalances in problem features. This not only improves the accuracy of predictions but also provides stronger support for designing personalized learning paths. Future research should further explore how to dynamically adjust the difficulty of different problems to better help students master knowledge while enhancing the performance of knowledge tracing models.

Conclusions and limitations

This paper proposes a difficulty-aware programming knowledge tracing with large language models. The model inputs programming problems and knowledge concepts into the CodeBERT pre-trained model, leveraging difficulty annotations generated by the large language model, combined with attention mechanisms and graph attention mechanisms, to create a semantic matrix of difficulty. Through the knowledge update layer, the model can dynamically adjust the student's knowledge state and predict their future mastery of knowledge. By combining the comprehension difficulty of problems with the difficulty of knowledge concepts, the model provides a more comprehensive and accurate tracking of the student's learning state.

The model not only evaluates the student's grasp of knowledge concepts but also captures the challenges students face in problem comprehension, thereby enhancing prediction accuracy. With the help of attention mechanisms and graph attention networks, the model effectively integrates complex problem information and updates the student's knowledge state. As students complete problems of varying difficulty, the model dynamically adjusts their knowledge mastery state, reflecting their learning progress in real time.

Although the model demonstrates good performance, there are still areas for further improvement and optimization. For instance, future research could consider integrating more data sources generated during the learning process, such as code debugging information, time management, and even emotional data. By incorporating multimodal information, the model would gain a more comprehensive understanding of student learning behaviors, thus improving the accuracy of predictions and knowledge tracing. Current models primarily focus on short-term knowledge state updates; in the future, a long-term prediction module could be introduced to combine students' historical performance and long-term learning paths, predicting changes in knowledge mastery over extended periods. This would facilitate better adjustments to curricula and teaching strategies. With these improvements, future knowledge tracing models will be more personalized and intelligent, helping students master knowledge more efficiently in programming learning, while also providing educators with more precise teaching strategy support.

Data availability

The dataset analyzed during the current research period can be found in the Project Code net repository at https://github.com/IBM/Project_CodeNet.git. The code for this study can be found at the following GitHub repository: https://github.com/ELLE157/DPKT.

Received: 24 October 2024; Accepted: 28 March 2025
Published online: 03 April 2025

References

1. Piech, C. *et al.* Deep knowledge tracing. *Advances in neural information processing systems* **28** (2015).
2. Shen, S. et al. A survey of knowledge tracing: Models, variants, and applications. *IEEE Trans. Learn. Technol.* https://doi.org/10.1109/TLT.2024.3383325 (2024).
3. Wang, J. et al. How problem difficulty and order influence programming education outcomes in online judge systems. *Heliyon* **9**(11), e20947 (2023).
4. Piwek, P. & Savage, S. Challenges with learning to program and problem solve: An analysis of student online discussions. in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 494–499 (2020).
5. Minn, S., Yu, Y., Desmarais, M. C. *et al.* Deep knowledge tracing and dynamic student classification for knowledge tracing. in *2018 IEEE International Conference on Data Mining (ICDM)*, 1182–1187 (IEEE, 2018).
6. Li, S. et al. Sthkt: Spatiotemporal knowledge tracing with topological Hawkes process. *Expert Syst. Appl.* **259**, 125248 (2025).
7. Liu, Q. et al. Ekt: Exercise-aware knowledge tracing for student performance prediction. *IEEE Trans. Knowl. Data Eng.* **33**, 100–115 (2019).
8. Zhang, J., Shi, X., King, I. & Yeung, D.-Y. Dynamic key-value memory networks for knowledge tracing. in *Proceedings of the 26th International Conference on World Wide Web*, 765–774 (2017).
9. Ghosh, A., Heffernan, N. & Lan, A. S. Context-aware attentive knowledge tracing. in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2330–2339 (2020).
10. Tong, S. *et al.* Structure-based knowledge tracing: An influence propagation view. in *2020 IEEE international conference on data mining (ICDM)*, 541–550 (IEEE, 2020).
11. Nakagawa, H., Iwasawa, Y. & Matsuo, Y. Graph-based knowledge tracing: modeling student proficiency using graph neural network. in *IEEE/WIC/ACM International Conference on Web Intelligence*, 156–163 (2019).
12. Sun, X. et al. Daskt: A dynamic affect simulation method for knowledge tracing. *IEEE Trans. Knowl. Data Eng.* https://doi.org/10.1109/TKDE.2025.3526584 (2025).
13. Passalis, N., Tzelepi, M. & Tefas, A. Probabilistic knowledge transfer for lightweight deep representation learning. *IEEE Trans. Neural Netw. Learn. Syst.* **32**, 2030–2039 (2020).
14. Qiu, Y., Qi, Y., Lu, H., Pardos, Z. A. & Heffernan, N. T. Does time matter? Modeling the effect of time with bayesian knowledge tracing. in *Educational Data Mining*, 139–148 (2011).

15. Zhu, R. *et al.* Programming knowledge tracing: A comprehensive dataset and a new model. in *2022 IEEE International Conference on Data Mining Workshops (ICDMW)* 298–307 (IEEE, 2022).
16. Qiu, Y., Qi, Y., Lu, H., Pardos, Z. A. & Heffernan, N. T. Application of deep self-attention in knowledge tracing. arXiv preprint arXiv:2105.07909 (2021).
17. Pan, J., Dong, Z., Yan, L. & Cai, X. Knowledge graph and personalized answer sequences for programming knowledge tracing. *Appl. Sci.* **14**, 7952 (2024).
18. Sun, X. et al. Lgs-kt: Integrating logical and grammatical skills for effective programming knowledge tracing. *Neural Netw.* **185**, 107164 (2025).
19. Shi, Y., Chi, M., Barnes, T. & Price, T. Code-dkt: A code-based knowledge tracing model for programming tasks. arXiv preprint arXiv:2206.03545 (2022).
20. Jeon, J. & Lee, S. Large language models in education: A focus on the complementary relationship between human teachers and chatgpt. *Educ. Inf. Technol.* **28**, 15873–15892 (2023).
21. Liu, F. et al. Fuzzy Bayesian knowledge tracing. *IEEE Trans. Fuzzy Syst.* **30**, 2412–2425 (2021).
22. Pelanek, R. Bayesian knowledge tracing, logistic models, and beyond: An overview of learner modeling techniques. *User Model. User-Adap. Inter.* **27**, 313–350 (2017).
23. Cen, H., Koedinger, K. & Junker, B. Learning factors analysis–a general method for cognitive model evaluation and improvement. in *International Conference on Intelligent Tutoring Systems*, 164–175 (Springer, 2006).
24. Pavlik, P. I., Cen, H. & Koedinger, K. R. Performance factors analysis–a new alternative to knowledge tracing. in *Artificial intelligence in education*, 531–538 (Ios Press, 2009).
25. Li, J., Deng, Y., Mao, S. *et al.* Knowledge-associated embedding for memory-aware knowledge tracing. in *IEEE Transactions onComputational Social Systems* (2023).
26. Cheung, L. P. & Yang, H. Heterogeneous features integration in deep knowledge tracing. in *International Conference on Neural Information Processing*, 653–662 (Springer, 2017).
27. Vie, J. J. & Kashima, H. Knowledge tracing machines: Factorization machines for knowledge tracing. in *Proceedings of the AAAI conference on artificial intelligence* **33**, 750–757 (2019).
28. Shen, S. *et al.* Convolutional knowledge tracing: Modeling individualization in student learning process. in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1857–1860 (2020).
29. Wang, D., Zhang, L., Zhao, Y. *et al.* Deep knowledge tracking integrating programming exercise difficulty and forgetting factors. in *International Conference on Intelligent Computing*, 192–203 (Springer Nature Singapore, 2024).
30. Scherer, R., Siddiq, F. & Viveros, B. S. A. A meta-analysis of teaching and learning computer programming: Effective instructional approaches and conditions. *Comput. Hum. Behav.* **109**, 106349 (2020).
31. Kazemitabaar, M. *et al.* Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. in *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 1–20 (2024).
32. Xu, S., Li, Z., Mei, K. & Zhang, Y. Core: Llm as interpreter for natural language programming, pseudo-code programming, and flow programming of AI agents. arXiv preprint arXiv:2405.06907 (2024).
33. Nam, D., Macvean, A., Hellendoorn, V., Vasilescu, B. & Myers, B. Using an llm to help with code understanding. in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 1–13 (2024).
34. Cai, Y. *et al.* Low-code llm: Visual programming over llms. arXiv preprint arXiv:2304.08103**2** (2023).
35. Li, W.-D. & Ellis, K. Is programming by example solved by llms? arXiv preprint arXiv:2406.08316 (2024).
36. Agarwal, A. *et al.* Copilot evaluation harness: Evaluating llm-guided software programming. arXiv preprint arXiv:2402.14261 (2024).
37. Yu, Y. *et al.* Eckt: Enhancing code knowledge tracing via large language models. in *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 46 (2024).
38. Fu, L. *et al.* Sinkt: A structure-aware inductive knowledge tracing model with large language model. arXiv preprint arXiv:2407.01245 (2024).
39. Vadaparty, A. *et al.* Cs1-llm: Integrating llms into cs1 instruction. in *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* 297–303 (2024).
40. Kazemitabaar, M. *et al.* How novices use llm-based code generators to solve cs1 coding tasks in a self-paced learning environment. in *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research* 1–12 (2023).
41. Chen, J.-T. & Huang, C.-M. Forgetful large language models: Lessons learned from using llms in robot programming. in *Proceedings of the AAAI Symposium Series* **2**, 508–513 (2023).
42. Ashurova, M. & Ashurov, M. The role and significance of the concepts of hard skill and soft skill in teaching it and programming languages. *J. Pedagog. Invent Pract.* **18**, 68–70 (2023).
43. Wang, S., Xu, T., Li, H. *et al.* Large language models for education: A survey and outlook. arXiv preprint arXiv:2403.18105 (2024).
44. Alhafni, B., Vajjala, S., Bannò, S. *et al.* Llms in education: Novel perspectives, challenges, and opportunities. arXiv preprint arXiv:2409.11917 (2024).
45. Sun, X. et al. Harnessing domain insights: A prompt knowledge tuning method for aspect-based sentiment analysis[J]. *Knowl. Based Syst.* **298**, 111975 (2024).
46. Hwang, G.-J. & Chen, C.-H. Evaluating the difficulty of programming tasks using a hybrid approach. *Computers & Education* (2017).
47. Dreyfus, S. E. & Dreyfus, H. L. A five-stage model of the mental activities involved in directed skill acquisition. in *Human Simulation for Training and Education* (1980).
48. Huang, H. & Lin, T. (A framework for classification. Educational Technology & Society, Assessing programming task difficulty, 2018).
49. Puri, R. et al. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks[J]. arXiv preprint arXiv:2105.12655 (2021).
50. Shen, S. *et al.* Assessing student's dynamic knowledge state by exploring the question difficulty effect. in *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 427–437 (2022).
51. Hooper, M., Broer, M., Yarnell, L. M. & Holmes, J. Talking about teachers would sampling weight adjustments allow for teacher centric inferences in future timss assessments. *Stud. Educ. Eval.* **73**, 101148 (2022).
52. Feng, Z. *et al.* Codebert: A pre-trained model for programming and natural languages. arXiv preprint arXiv:2002.08155 (2020).
53. Pandey, S. & Karypis, G. A self-attentive model for knowledge tracing. arXiv preprint arXiv:1907.06837 (2019).
54. Yeung, C. K. & Yeung, D. Y. Addressing two problems in deep knowledge tracing via prediction-consistent regularization. in *Proceedings of the fifth annual ACM conference on learning at scale* 1–10 (2018).
55. Chen, J., Liu, Z., Huang, S., Liu, Q. & Luo, W. Improving interpretability of deep sequential knowledge tracing models with question-centric cognitive representations. arXiv preprint arXiv:2302.06885 (2023).
56. Huang, C. et al. Learning consistent representations with temporal and causal enhancement for knowledge tracing. *Expert Syst. Appl.* **245**, 123128 (2024).
57. Sun, X. et al. Target hierarchy-guided knowledge tracing: Fine-grained knowledge state modeling. *Expert Syst. Appl.* **251**, 123898 (2024).
58. Baker, R. S., Martin, T. & Rossi, L. M. Educational data mining and learning analytics. *The Wiley handbook of cognition and assessment: Frameworks, methodologies, and applications* 379–396 (2016).

59. Doshi-Velez, F. & Kim, B. Towards a rigorous science of interpretable machine learning. arXiv preprint arXiv:1702.08608 (2017).
60. McHugh, M. L. Interrater reliability: The kappa statistic. *Biochemia medica* **22**, 276–282 (2012).
61. Viera, A. J. & Garrett, J. M. Understanding interobserver agreement: The kappa statistic. *Fam. Med.* **37**, 360–363 (2005).

Acknowledgements
This work was supported by Guizhou Provincial Higher Education Undergraduate Teaching Content and Curriculum System Reform Project under Grants GZJG2024331 and GZJG2024323; Guizhou Provincial Science and Technology Projects (QKHJC[2024]youth012); Guizhou Province First-Class Undergraduate Course under Grant GZSy1kc202229; Humanities and Social Sciences Research Project of Guizhou Provincial Universities under Grant 2024RW171; Guizhou Provincial Philosophy and Social Science Research Special Fund (Research on the Spirit of the National Education Conference); Liupanshui Normal University Teaching Research and Reform Project under Grant 2024-07-02; Liupanshui Normal University Institutional Course Ideological and Political Education Reform Project, Grant 2024-08-024; Ministry of Education Industry-Academia Cooperation and Collaborative Talent Cultivation Project, Grant 241202802121104.

Author contributions
Lina Yang and Xinjie Sun proposed the ideas and methodologies, designed the experimental scheme, and wrote the main manuscript text. Hui Li and Ran Xu assisted in the experiment and prepared the figures. Xuqin Wei reviewed and modified the manuscript.

Declarations

Competing interests
The authors declare no competing interests.

Additional information
Correspondence and requests for materials should be addressed to X.S.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.