ProgMate: An Intelligent Programming Assistant based on LLM

Ying Li[1] Runze Yang[1] Xiantao Zhang[1] Peng Shi[2] Da Yang[2] Xuefei Huang[2,3]
liying@buaa.edu.cn da.yang@buaa.edu.cn huangxuefei@buaa.edu.cn
[1]School of Computer Science and Engineering, Beihang University, Beijing, China
[2]Key Laboratory of Data Science and Intelligent Computing, International Innovation Institute, Beihang University, Hangzhou, China
[3]Corresponding author

***Abstract*—This study addresses the challenges faced in personalized tutoring within large-scale programming courses, such as significant ability gaps among students, limited available resources, among others. For these reasons, we proposed an intelligent programming assistant, ProgMate, based on large language models (LLMs). Benefitting from the robust understanding and learning capabilities of LLM, ProgMate not only comprehensively monitors the learning process, but can also surpass human teaching in aspects such as intelligent assignment grading, identification of knowledge gaps, and assessment of learning abilities. ProgMate embodies a new 4-A digital teaching paradigm, characterized by its ability to provide precise guidance with anything, to anyone, anywhere, at any time, with features like "omnipresence", "adaptive guidance" and "customization". It facilitates the organic integration of collective teaching and individualized guidance, continuous learning, and long-term development, as well as resource efficiency and precision nurturing, offering a viable path and empirical support for the digital transformation of future education.**

Keywords—Large Language Model, Enhanced Knowledge Graph, Weak Knowledge Point Identification, Personalized Learning Pathway

I. INTRODUCTION

Personalized learning has always been one of the popular research directions in the education industry. The difficulty lies in how to reflect students' learning status in real time and accurately, so as to better teach students in accordance with their aptitude and promote the development of individual learners. The journal Spring also emphasized: personalized learning models seek to adapt to the pace of learning and the instructional strategies, content and activities being used to fit best each learner's strengths, weaknesses, and interests [1]. Personalized learning has increasingly become the focus of attention internationally.

Knowledge tracking is one of the core technologies of personalized learning. It mainly measures the learner's knowledge based on the learner's test result data, thereby evaluating his or her learning status. With the continuous advancement of deep learning, the field of knowledge tracking is constantly developing. Taking research in computer education as an example, Abdelrahman et al. used the deep learning DKT model, Pandey et al. used the attention mechanism in the DKT model, and Nagatani et al. added a forgetting mechanism for knowledge tracking [2],[3],[4]. These studies have made important contributions to the development of the field of cognitive diagnostics.

In recent years, intelligent tutoring, especially based on LLM, is gradually replacing traditional manual tutoring. In particular, the University of Toronto's recent work on the practical application of LLM in computer teaching has paved the way for using LLM to promote intelligent guidance [5],[6],[7]. When applying the above technology in actual teaching, we found that there are still the following problems:

1. In terms of cognitive diagnosis: most algorithms only use easily measurable factors in students' learning behavior (such as grades, number of submissions, accuracy, etc.) as data sets for training and analysis [8], lacking a direct description of the relationship between the code itself and the degree of mastery of knowledge points. Additionally, LLM tends to provide code repairs, and there are currently many related studies using LLM for code repairs. But if we directly provide code repairs, it will greatly affect students' learning motivation [9],[10].

2. In terms of intelligent guidance: the knowledge graph of the existing intelligent guidance platform rarely considers the implicit relationships between knowledge points (such as knowledge point affinity and similarity) or does not use the knowledge graph to conduct structured modeling of massive resources. In addition, the knowledge graph of existing platforms is often not used in conjunction with LLM, and there is still a lack of accuracy.

To this end, this article takes programming courses as the research object. Firstly, it investigates a semantically enhanced knowledge graph underpinned by LLM [11]. It uses the knowledge graph and LLM technology to identify weaknesses in student code [12]. Simultaneously, it integrates cognitive diagnostic technology for precise assessments of student abilities [13],[14]. Furthermore, the article offers personalized learning path recommendations and experimentally validates the method's effectiveness.

II. DESIGN AND IMPLEMENTATION

The paper harnesses the vast learning resources available on large-scale online programming platforms, employing the comprehension and perception capabilities of LLM to automatically build relevant knowledge graphs. Meanwhile, leveraging the storage and recording of student code submissions during programming learning on these platforms, it introduces the ability of large language models (LLMs) to recognize and understand code, conducting a fine-grained scan to pinpoint students' weak knowledge points. Further, integrating the detection outcomes with the recorded data on students' learning behaviors, it feeds this information into a cognition model to generate personalized assessments of students' programming proficiency. Rooted in the premise that "the quality of assignment code reflects the mastery of knowledge, which in turn reflects the attainment of abilities," the study collaboratively establishes personalized learning paths. These paths tailor problem recommendations and learning resources to students with varying levels of competence, enhancing learning outcomes and teaching quality. The overall design workflow of this project is depicted in Figure 1.

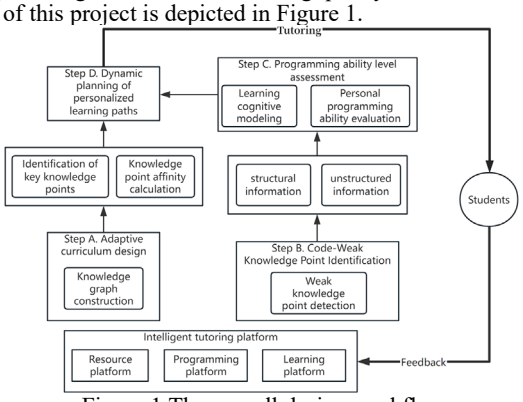

Figure 1 The overall design workflow

A. Semantically Enhanced Knowledge Graph

The Course Knowledge Graph represents teaching content in a structured manner through a clear logical framework, enabling students to comprehend the structure and inherent connections of knowledge. This study aims to adopt a new model of knowledge graph integrated with LLMs, with the objective of enhancing the knowledge representation and relational reasoning capabilities of existing course knowledge graphs. The overall design approach, involves first converting the semantic information of the knowledge graph into low-dimensional vectors; followed by analyzing and extracting features such as the affinity between knowledge points and key knowledge points; finally, visually presenting the entire curriculum's knowledge system and the network of relationships among knowledge points, which can also facilitate the construction of the course's logical framework.

1) Knowledge point extraction based on the integration of LLMs and knowledge graphs.

Owing to the excellent performance of LLMs in natural language processing tasks, this study proposes a reinforced knowledge point extraction method, integrating LLMs with knowledge graphs (abbreviated as KG-LLM). This approach aims to enhance the text and language understanding capabilities of conventional entity extraction models, while also addressing the shortcoming of LLMs' lack of explainability. The KG-LLM effectively captures knowledge from vast amounts of both labeled and unlabeled data through a process of "pre-trained LLMs followed by fine-tuning for downstream tasks," significantly expanding the model's knowledge representation and reasoning abilities.

Specifically, we used different methods to enhance the self-developed BERT-BiLSTM-CRFs optimization model library based on LLM. Here, KG-LLM embedding involves utilizing LLM to encode the text descriptions of entities and relations, thereby enriching the representation of the knowledge graph; KG-LLM completion employs LLM as both an encoder and generator to acquire a broader range of textual information; KG-LLM construction, based on the previous two approaches, enables end-to-end knowledge graph construction (building a complete knowledge graph in one step) and directly distilling a knowledge graph from the LLM.

2) Automatic Construction of Knowledge Graphs Based on Affinity-Inspired Approaches

In the research, an TransD knowledge graph embedding model, based on affinity-inspired principles, is proposed (referred to as Affinity-TransD). This model transforms knowledge graph attributes into vectors while preserving some of the original semantic connection information. Consequently, it enables the transformation of affinity analysis tasks between entities into the computation of semantic similarity between entity vectors. Affinity-TransD relies on dynamic mapping matrices for entities and relations, Mrh and Mrt, respectively, which, under the guidance of affinity, map entities or relations represented by a triplet (head entity h, relation r, tail entity t) into separate spaces.

To address the insensitivity issue of cosine similarity, which may incorrectly identify vectors that are close in direction but far apart in distance as similar, this study proposes incorporating Euclidean distance. The similarity in Euclidean space would be measured by the ratio of the distance between the edges of the two vectors to the sum of the lengths of their edges.

For two n-dimensional entity vectors A ($x_{11}, x_{12},\ldots,x_{1n}$) and B ($x_{21}, x_{22},\ldots,x_{2n}$), their similarity measure based on Euclidean space is calculated according to Formula 1. Here, the range of values is [0,1]. The higher the value, the lower similarity between entities A and B.

$$sim(A,B) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \qquad \text{Formula 1}$$

B. Weak Knowledge Points Identification

This module constitutes the core and challenge of the research, aiming to automatically extract weak knowledge points from students' submitted programming assignments. The study proposes a novel, enhanced code representation model based on collaborative fusion. It fully leverages the chain-of-thought (CoT) capability of LLM to better "comprehend" both the implicit structured information within the code (the program itself, including its constituent elements and their relationships) and unstructured information (the execution process, such as the sequence of function calls). This approach yields more precise, explainable, and logically coherent error features in code (e.g., locations of erroneous statements, corresponding knowledge points), effectively addressing the issue where Abstract Syntax Trees (ASTs) merely focuse on grammatical structure and have no ability to understand semantics.

Specifically, for erroneous code submitted by students, the process involves: first, employing syntax trees to extract basic information from the code; then, leveraging a fine-tuned LLM to further extract both structured and unstructured information within the code; finally, devising a hybrid prompt that utilizes the code information analyzed in step two as heuristic data. This prompt is designed to guide the LLM in intelligently identifying weak knowledge points by analyzing the erroneous source code against correct code templates.

1) Code analysis techniques for both static and dynamic programs based on syntax trees

To enhance the representation of code, the paper adopts a fusion approach combining LLM with ASTs. Its design principle revolves around leveraging the structured nature of ASTs to augment the semantic understanding and logical reasoning abilities for code of LLM. Initially, explicit features of the code, such as syntax errors and reference relationships—semantics and logical structures recognizable by compilers—are extracted from the syntax tree. This work first employs the tree-sitter tool to transform the code into an AST structure. Subsequently, the AST is extended into a directed graph representation, and Graph2Vec is utilized to embed this graph structure into fixed-length vector spaces. These operations are applied to all code samples, transforming correct codes into sets of vectors, which are then clustered using K-Medoids to derive representative vectors for correct code templates. After acquiring template vectors, the similarity between the vector of the error code under analysis and all correct template vectors is computed to identify the most similar correct template, forming the basis for further error code analysis. Concurrently, analysis tools grounded in syntax trees, like DCC (Debugging C Compiler), are employed to assist in the source code analysis. The outcomes are integrated and fed into the LLM in subsequent steps, as depicted in Figure 2.

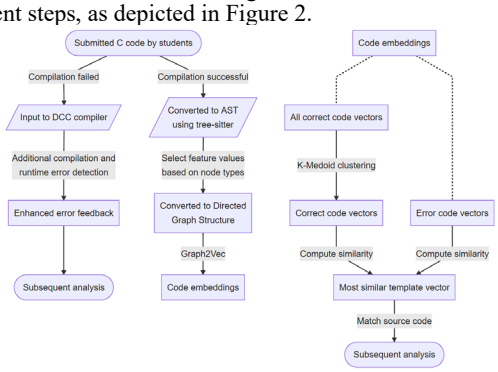

Figure 2: Error Code Analysis Based on Syntax Tree

2) Code Representation Model based on Large-Scale Models and Abstract Syntax Tree Integration

To extract weak knowledge points from students' erroneous code, it is necessary to delve into both the structured and unstructured information implicit in the code. Structured information refers to fundamental code elements that adhere to grammatical rules, such as variable declarations, function definitions and calls, control structures, data structures, which collectively form the framework of the code. Unstructured information, on the other hand, encompasses code comments, whitespace indentation, variable naming conventions, programming styles, and the like. Although these do not impact code execution, they are crucial for code readability and maintainability.

Then, design an integrated prompt to leverage the LLM in analyzing implicit features of the code, encompassing execution behaviors such as data flow and control flow. This study initiates by fine-tuning the LLM with a minimal set of annotated data, aligning its outputs with prescribed content and formatting requirements. The prompt should explicitly outline the task objective, input format, and the desired output format, accompanied by concrete examples to facilitate the model's comprehension of the needs. For instance, in extracting structured information, the prompt might read: "Given the following C code snippet, extract its structured information and output it in the JSON format shown below..." Once the prompt is meticulously crafted, the API of the fine-tuned LLM can be invoked to obtain both the structured and unstructured information from the code.

Based on pre-designed correct code templates, syntax tree information, structured and unstructured code information, as well as all course knowledge points extracted from the knowledge graph, we need to integrate various pieces of information. Through multiple rounds of dialogue with LLM, we extract the course knowledge points

related to the errors in the erroneous code. Here, we categorize course knowledge points into 12 major categories first, with each major category further divided into several subcategories. The course knowledge points are illustrated in Figure 3.

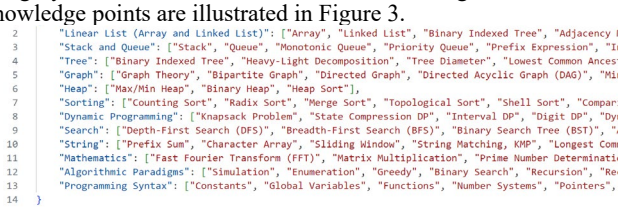

Figure 3: Selected Course Knowledge Points

To fully leverage the various acquired information, this study employs multiple rounds of dialogue. In each round, different information is utilized to pose questions to the LLM from diverse angles and on distinct topics, ultimately yielding results. Specifically, the process of invoking the LLM is primarily divided into two stages, as illustrated in Figure 4.

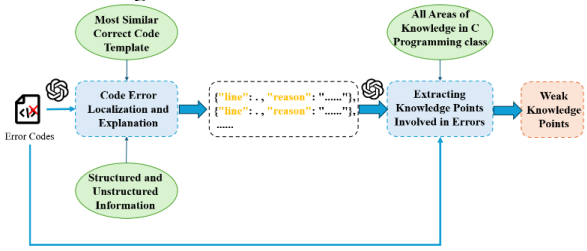

Figure 4: LLM Information Integration

After acquiring error codes, the most similar correct code, and both structured and unstructured information, these elements must be integrated into a carefully crafted prompt to pinpoint the line number of the code error and provide an explanation for it. The format follows a JSON dictionary where each error comprises two keys, "line" and "reason," specifying the line number where the error occurs and offering an explanatory note. Subsequently, these intermediate results are combined with the original erroneous code and the set of course knowledge points, generating a new prompt that steers the LLM toward analyzing the specific course concepts implicated in the code error.

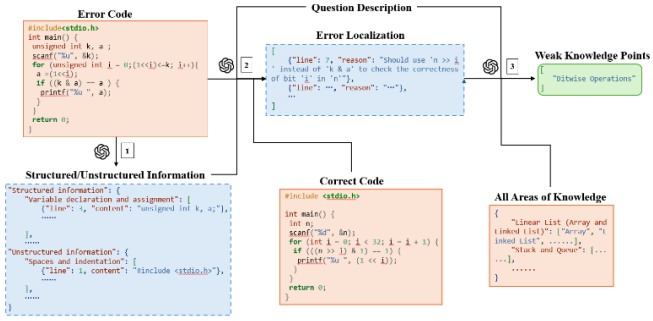

Figure 5: Example of invoking a LLM

Figure 5 provides a concrete example of the model's specific inputs and outputs during invocations. Here, all orange sections represent the inputs to the LLM; blue sections denote intermediate results generated during these calls; and green sections showcase the final outcomes. It is evident that the experiment necessitates designing three distinct prompts to invoke the LLM sequentially. In the first invocation, erroneous code is fed into the fine-tuned LLM to extract both structural and non-structural information. During the second call, we integrate the student's submitted erroneous code, its most similar correct counterpart, the structural and non-structural information obtained from the previous step, along with the problem description, to pinpoint errors in the submitted code and provide explanations for these errors. In the final round of invocation, the outcomes from the previous step, the programming problem, and a set of knowledge points are combined into the prompt, resulting in the output that identifies weak knowledge points.

Meanwhile, through continuous iterative calls, we also refine and adjust the questioning strategies across multiple rounds of dialogue, thereby enhancing the accuracy of LLM in pinpointing knowledge points.

C. *Assessment of Programming Skills*

Addressing the shortcomings of current personalized learning cognitive assessment models when applied to programming education scenarios, this approach combines code feature information with programming knowledge points to evaluate students' mastery of knowledge and achievement of skills. Given that students' learning and cognitive processes are continuous and have a distinct temporal sequence, the integration of knowledge tracing technology enables a more precise depiction of students' cognitive structures and processes. This, in turn, assesses the dynamic changes in various aspects of students' competency states, ultimately facilitating the creation of individualized learning profiles for each student.

1) Optimize the "Question-Knowledge Point" Q matrix into the "Question-Mastery Degree of Knowledge Points" P matrix.

The Q matrix is a widely used "question-knowledge" matrix that statically describes the inclusion relationship between "questions" and "knowledge points" using simple binary variables (inclusive/not inclusive), but it fails to account for variations in students' mastery of knowledge points over time. To address this, an improved P Matrix (Personalized Q-matrix) has been proposed. The P Matrix, based on the Q Matrix, incorporates a cognitive probability model and dynamically represents the intrinsic relationships among "students", "questions" and "levels of knowledge mastery" by setting a learning effectiveness factor, e ($0 \leqslant e \leqslant 1$). It emphasizes cognitive development and individual differences in learning. In short, the P Matrix optimizes the Q Matrix's indication of "whether questions contain knowledge points" into "the probability that a student has mastered the knowledge points tested by questions" based on their learning profile. For example, in matrix P, if the value at the i-th row and j-th column is e, it implies that problem i includes knowledge point j but the probability of the submitted code correctly utilizing knowledge point j is e. Each student has their own unique P matrix.

2) Transform static cognitive models into temporal dynamic cognitive models.

Students' cognitive development is founded upon a process of knowledge accumulation. It's not merely a straightforward addition of quantity, but rather a continuous restructuring of cognitive frameworks. In the context of programming courses, as practical exercises increase, both the accuracy of solutions and the speed of responding generally improve. However, this improvement rate diminishes over time, and after sufficient practice, the increment in capability becomes negligible. Upon reaching this stage, it is assumed that students have essentially mastered the subject matter, with the probability of correctly answering similar questions again approaching 1. Hence, the extent to which students grasp knowledge varies over time, exhibiting features of learning (ability growth) and forgetting (ability decline).

To achieve this, based on existing deep learning cognitive models, statistical research is employed to design a joint factor of learning effectiveness. This factor decomposes the elements impacting students' programming competency into two categories of features: external and internal factors. External factors, related to the problems, encompass the difficulty of the question, the quantity and variety of knowledge points included, as well as the individual difficulty level of each knowledge point. Internal factors, pertaining to the students, include learning behaviors, attitudes, efficiency, and the extent of prior knowledge mastery. During modeling, these factors must be quantified and introduced as training parameters into the cognitive model, as illustrated in Formula 2.

$$E(e_{ij}(t)) = \text{Prob}(\alpha, \beta, \theta) \qquad \text{Formula 2}$$

Among these, E represents the time-series analysis of students' programming proficiency; eij(t) denotes the extent to which student i has mastered knowledge point j at time t; α signifies the exogenous vector, β represents the endogenous vector, and θ denotes the learning forgetting index.

D. *Personalized Learning Pathway Planning*

Personalized learning path recommendation realizes the precision and intelligence of educational services. It leverages the inferential and self-referential properties of knowledge graph methods to construct relationships between knowledge points, further refining learning points across three progressively advanced layers: data, knowledge, and skills. Based on learners' current learning status and objectives, it provides an efficient guiding strategy tailored to the learner, with the design process outlined in Figure 6.

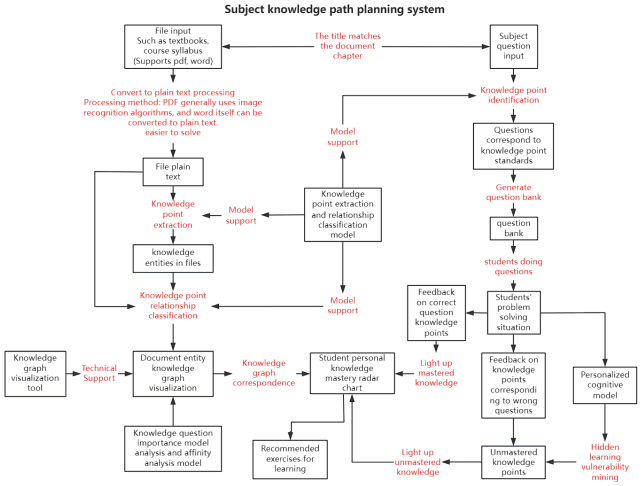

Figure 6: Design Flowchart for Personalized Learning Path

1) *Formulate an initial learning path recommendation based on the importance of knowledge points.*

When constructing personalized learning paths, priority should be given to learning or reviewing key knowledge points that have a significant (positive or negative) impact on students' programming skill development. To achieve this, we plan to utilize the importance metric imp(k_i) of the vital knowledge points identified through knowledge graph analysis (Part A) for measurement, with the calculation formula as shown in Formula 3.

$$imp(k_i) = \frac{in(k_i)out(k_i)}{\ln[in(k_i)]} \qquad \text{Formula 3}$$

Among these, in(k_i)and out(k_i) respectively represent the number of incoming and outgoing links for a knowledge point, that is, the out-degree and in-degree of the corresponding vertex in the knowledge graph. Consequently, if a knowledge point has a higher number of connections in the knowledge graph, it indicates a closer relationship with more knowledge points. In the personalized learning path recommendation algorithm, we first filter out the list of the student's weak knowledge points according to Part B. Then, based on the calculated importance of knowledge points, we select the top n knowledge points with high affinity to the weak ones to construct an intelligent tutorial path.

III. EXPERIMENT

A. *Code - Weak Knowledge Point Identification Module*

1) *Dataset*

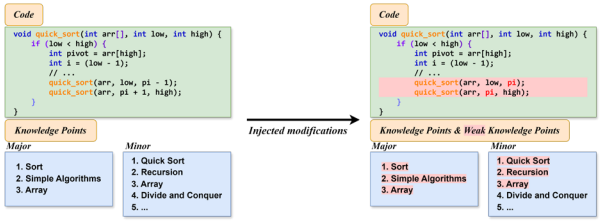

Figure 7: Illustration of dataset label construction.

As shown in figure 7, to validate our design's capability to effectively identify weak knowledge points in code, we collected and cleaned 1,200 solution codes from various Online Judges (OJs) including our self-built platform, Luogu[1], LeetCode[2], Codeforces[3], encompassing 12 major topics and 200 minor topics. We utilized an automated annotation plus manual verification approach to label these 1,200 codes with pertinent knowledge points. Subsequently, by injecting errors into the code to intentionally create faulty versions, we modified the answer codes and labeled the erroneous knowledge points. Following the annotation and cleansing process, we obtained 2,276 codes annotated with all knowledge points and error knowledge points. These error knowledge points served as labels for our testings.

2) *Experiment Design*

We tested multiple LLMs with our code representation methods, using a zero-shot evaluation to reflect their innate ability and real-world scenarios. We included a synonym dictionary to count equivalent terms as hits and allowed for output redundancy. Outputs hitting the core theme receive partial score, with node affinity used to assess and grant partial score for valid, theme-aligned answers that don't completely match the assigned label.

3) *Experiment Results and Analysis*

The experimental results are shown in Table 1.

Table 1: Code - Weak Knowledge Point Identification Test. "ft" denotes models after fine-tuning.

Model	with code representation	Prec(all)	Prec(major)	Prec(strict)
Qwen1.5-7B-Chat	×	0.13	0.19	0.09
	√	0.14	0.19	0.10
Qwen1.5-7B-Chat(ft)	×	0.15	0.16	0.13
	√	0.17	0.20	0.16
CodeLLaMA-13B-Inst	×	0.04	0.04	0.01
	√	0.05	0.06	0.04
CodeLLaMA-13B-Inst (ft)	×	0.08	0.10	0.03
	√	0.13	0.15	0.10
GPT-3.5-Turbo	×	0.20	0.22	0.18
	√	0.24	0.27	0.21
GPT-4-Turbo	×	0.36	0.39	0.31
	√	**0.38**	**0.44**	**0.35**

We assessed precision in three ways: Prec(all) considers all relevant hits even with topic mismatches, Prec(major) counts any hit with a matching core topic, and Prec(strict) requires exact knowledge point alignment, including synonyms. The experiment showed that our code representation improves identification of weak knowledge points across LLMs, with LLMs showing more significant improvements due to their greater capacity for following instructions and integrating code references.

B. *Personalized Learning Path Planning*

1) *Dataset*

To validate the effectiveness of the personalized cognitive model proposed in this chapter, a dataset collected from the Educoder[4] was utilized. This experimental dataset comprises six programming problems, totaling 16,428 data records submitted by 608 students. The description of the knowledge points for the dataset's problems is illustrated in Table 2.

Table 2: Description of Personalized Cognitive Model Dataset

Index	Topic	Knowledges
C-1	String Concatenation	Variables, Operators, String, Expressions, I/O
C-2	Modify List Elements	Constants, Variables, Operators, Expressions, Lists, I/O
C-3	Calculate Mass	Constants, Variables, Operators, Expressions, I/O
C-4	Data Classification	Constants, Variables, String, Expressions, Conditions, I/O
C-5	Factorial Calculation	Constants, Variables, Operators, String, Expressions, Conditions, I/O
C-6	Dictionary Operations	Constants, Variables, String, Expressions, Dictionaries, I/O

2) *Experiment Design*

[1] https://www.luogu.com.cn/
[2] https://leetcode.cn/
[3] https://codeforces.com/
[4] https://www.educoder.net/

We created three tasks: Task 1 aims to predict if a student will correctly solve an upcoming attempt using their submission history for a question. Task 2 involves predicting whether a student will get the next problem right based on their code submissions for a current problem. Task 3 focuses on a comprehensive prediction of a student's success on the next problem, considering their submissions on a series of related problems.

3) Experimental Results and Analysis

Table 3 shows the CFM model has the lowest accuracy in forecasting tasks due to its non-specific handling of errors, affecting learning rates and proficiency parameters negatively. In contrast, our personalized joint factor model and two deep learning-based cognitive models outperform probabilistic models by better feature extraction, with the personalized model excelling in accuracy through effective use of coding submissions and detailed P matrix data. Experiments also show that predictive accuracy is higher in Task 1 and Task 2 than in Task 3, as Task 3 deals with longer input sequences, making predictions more challenging.

Table 3: Predictive experimental results of cognitive models.

Task type	Model Type	Name	Accuracy	AUC
Next Attempt Prediction	Probabilistic	CFM	0.645±0.013	0.686±0.009
		PFM	0.693±0.023	0.705±0.012
	Deep	DKT	0.789±0.005	0.834±0.007
		PCM	**0.853±0.010**	**0.907±0.007**
Next Problem Prediction	Probabilistic	CFM	0.677±0.004	0.746±0.001
		PFM	0.695±0.014	0.806±0.015
	Deep	DKT	0.706±0.007	0.866±0.006
		PCM	**0.875±0.009**	**0.876±0.011**
Composite Prediction	Probabilistic	CFM	0.625±0.001	0.616±0.001
		PFM	0.710±0.003	0.623±0.030
	Deep	DKT	0.728±0.021	0.672±0.005
		PCM	**0.790±0.015**	**0.833±0.007**

C. Evaluating the Impact on Programming Education

1) Experiment Design

To comprehensively assess ProgMate's effectiveness and gather students' reactions and evaluations in real-world scenarios, we piloted its implementation in a C language programming course at a large university in China over one semester. Course specific information is shown in Table 4.

Table 4: Basic Information of Deployed Course.

Key	Value
Course name	Foundations of Programming
Targeted at beginners or not	Yes
Duration	32 theoretical hours + 16 practical hours
Number of participants	~1,400
Accessible time	non-testing time

2) Data Collection

To facilitate improved data collection, we conduct weekly surveys and gather open-ended suggestions through the Online Judge discussion forum. In our weekly surveys, we consider the following dimensions as shown in Table 5.

Table 5: Dimensions of the Questionnaire Survey.

#	Name	Options
1	Usage(hour)	integers, filled out by students
2	Most frequently used functions	sorting results for all functions
3	Score relative to ChatGPT et al.	integers, from 0 to 100, with 50 as tie
3*	Why ProgMate is better than ChatGPT et al.	optional, open question
4	Score relative to human tutoring	integers, from 0 to 100, with 50 as tie
4*	Why ProgMate is better than human tutoring	optional, open question
5	Overall assistance score	integers, from 0 to 10

3) Results Analysis

We have received positive feedback across all eight questionnaire collections, with an average response rate of approximately 88%.

Table 6: ProgMate Evaluation Sheet. Note that ProgMate based on GPT-4-Turbo is reported in a class of 100 students.

Options	ProgMate based on GPT-3.5-Turbo	ProgMate based on GPT-4-Turbo
Score relative to ChatGPT et al. (50 for tie)	67.4	82.6
Score relative to human tutoring (50 for tie)	40.6	55.5
Overall assistance score (0~10)	7.7	8.9

Table 6 shows promising assistance scores for ProgMate, indicating its effectiveness in aiding student learning. ProgMate's GPT-4-Turbo-based feedback outdoes that based on GPT-3.5-Turbo and even outperforms teaching assistants to some extent. Survey responses favor ProgMate over ChatGPT and human guidance due to its detailed, task-based explanations and its constant availability for timely responses, underlining the success of our development efforts.

IV. CONCLUSION

The AI teaching assistant ProgMate accelerates the application of intelligent education by deeply integrating LLMs with digital educational technologies. (1) Personalizing learning behavior: It has solved the problem of large-scale personalized teaching, facilitating the intelligent, scenario-based, and precise transformation of education. (2) Improving educational quality: Through the construction of a digital and intelligent teaching platform and the development of new forms of teaching materials, the aim is to improve the quality of education for programming courses. The digital and intelligent empowerment methods better meets the learning needs of students, making it easier for them to understand and master complex programming concepts, which improves academic performance and learning experience.

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Project of China (2021ZD0110700) and National Natural Science Foundation of China (No.62107002) and the Engineering Research Center of Integration and Application of Digital Learning Technology, Ministry of Education (1321008).

REFERENCES

[1] Shemshack, Atikah, and Jonathan Michael Spector. "A systematic literature review of personalized learning terms." Smart Learning Environments 7.1 (2020): 33.
[2] Abdelrahman, Ghodai, and Qing Wang. "Knowledge tracing with sequential key-value memory networks." Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval. 2019.
[3] Pandey, Shalini, and George Karypis. "A self-attentive model for knowledge tracing." arXiv preprint arXiv:1907.06837 (2019).
[4] Nagatani, Koki, et al. "Augmenting knowledge tracing by considering forgetting behavior." The world wide web conference. 2019.
[5] Kazemitabaar, Majeed, et al. "CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs." arXiv preprint arXiv:2401.11314 (2024).
[6] Bakas, Nikolaos P., et al. "Integrating LLMs in Higher Education, Through Interactive Problem Solving and Tutoring: Algorithmic Approach and Use Cases." European, Mediterranean, and Middle Eastern Conference on Information Systems. Cham: Springer Nature Switzerland, 2023.
[7] Agarwal, Vibhor, et al. "" Which LLM should I use?": Evaluating LLMs for tasks performed by Undergraduate Computer Science Students in India." arXiv preprint arXiv:2402.01687 (2024).
[8] J. S. Santos, W. L. Andrade, J. Brunet and M. R. A. Melo, "A Systematic Literature Review on Predictive Cognitive Skills in Novice Programming," 2022 IEEE Frontiers in Education Conference (FIE), Uppsala, Sweden, 2022, pp.1-9,doi: 10.1109/FIE56618.2022.9962582.
[9] Bouzenia, Islem, Premkumar Devanbu, and Michael Pradel. "RepairAgent: An Autonomous, LLM-Based Agent for Program Repair." arXiv preprint arXiv:2403.17134 (2024).
[10] Xia, Chunqiu Steven, Yuxiang Wei, and Lingming Zhang. "Automated program repair in the era of large pre-trained language models." 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). IEEE, 2023.
[11] Pan, Shirui, et al. "Unifying large language models and knowledge graphs: A roadmap." IEEE Transactions on Knowledge and Data Engineering (2024).
[12] Sun, Yuqiang, et al. "LLM4Vuln: A Unified Evaluation Framework for Decoupling and Enhancing LLMs' Vulnerability Reasoning." arXiv preprint arXiv:2401.16185 (2024).
[13] Abu-Rasheed, Hasan, et al. "Supporting Student Decisions on Learning Recommendations: An LLM-Based Chatbot with Knowledge Graph Contextualization for Conversational Explainability and Mentoring." arXiv preprint arXiv:2401.08517 (2024).
[14] Park, Minju, et al. "Empowering Personalized Learning through a Conversation-based Tutoring System with Student Modeling." arXiv preprint arXiv:2403.14071 (2024).