KGNN-KT: Enhancing Knowledge Tracing in Programming Education Through LLM-Extracted Knowledge Graphs

Di Zhang(✉), Qiang Niu, Tianshi Wang, Yuntian Hou, Jinheng Wu, Chao Zhang, and Angelos Stefanidis

Xi'an Jiaotong-Liverpool University, Suzhou 215400, People's Republic of China
{di.zhang,qiang.niu,angelos.stefanidis}@xjtlu.edu.cn,
{tianshi.wang19,jinheng.wu20}@student.xjtlu.edu.cn,
{yuntian.hou20,chao.zhang1902}@alumni.xjtlu.edu.cn

Abstract. This paper introduces KGNN-KT, an innovative neural knowledge tracing framework that enhances programming education through structured knowledge representation. Our approach combines large language models (LLMs) with graph neural networks to model both student learning patterns and conceptual relationships in programming. The system first constructs a comprehensive knowledge graph by extracting programming concepts from problem descriptions and solutions using LLMs, which captures hierarchical dependencies between data structures, algorithms, and programming paradigms. The KGNN-KT model then processes this structured knowledge alongside student interaction histories through a multimodal architecture that integrates: (1) semantic embeddings of problem texts and code, (2) temporal modeling of student performance trajectories, and (3) graph-enhanced concept representations. Experiments across three programming education datasets demonstrate significant improvements, with an overall AUC of 0.84 (3.9% higher than leading baselines) and particularly strong results on complex problems (+5.3% AUC gain). Our work advances personalized programming education by bridging neural knowledge tracing with explicit knowledge structures, offering both accurate performance prediction and actionable curriculum insights. The system's modular design supports extensions to diverse programming domains and adaptive learning scenarios.

Keywords: Knowledge Tracing · Programming Education · Large Language Models · Graph Neural Networks · Interpretable AI

1 Introduction

In programming education, accurately tracking students' knowledge acquisition is crucial for personalized learning. It helps teachers understand learning progress, provides targeted instructional guidance, and enables students to better plan their learning paths.

D.-S. Huang et al. (Eds.): ICIC 2025, CCIS 2572, pp. 137–147, 2025.
https://doi.org/10.1007/978-981-96-9986-5_12

With the continuous development of programming education, current knowledge tracing solutions face several limitations [24]. Traditional methods like Bayesian Knowledge Tracing (BKT) [4] primarily rely on binary correctness data and static skill representations, making them unsuitable for complex problem-solving processes in modern programming tasks [17]. While deep learning-based approaches [33] have partially addressed these issues, they still exhibit shortcomings. Some methods lack deep understanding of code semantics [27] and fail to capture relationships between programming concepts, while others suffer from low efficiency when processing large-scale data [31]. Additionally, approaches combining LLMs with knowledge tracing [15], despite recent progress, continue to face challenges in computational efficiency and hallucination mitigation [3].

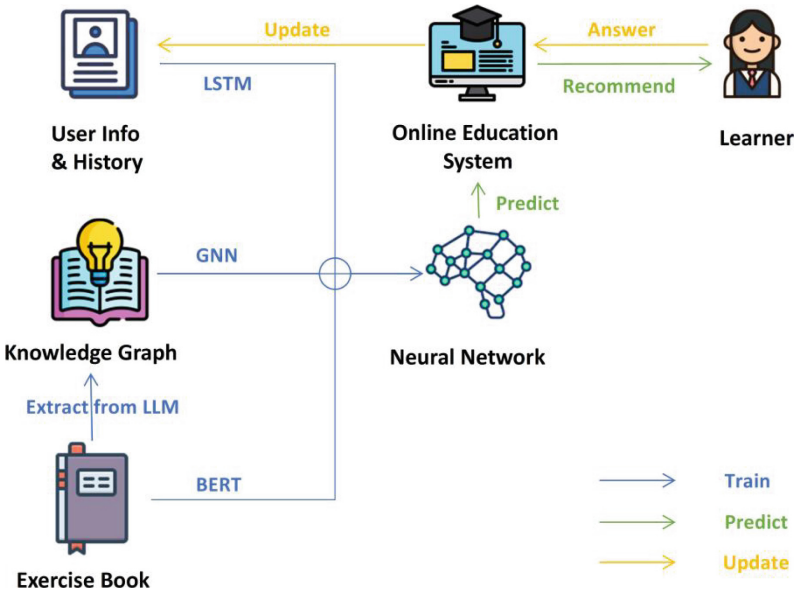

Fig. 1. Architecture of AI-Powered Programming Education System.

To address these challenges, we propose KGNN-KT, a neural knowledge tracing framework that synergizes LLM-extracted knowledge graphs with multimodal student modeling for programming education (Fig. 1). Our primary innovations consist of: (1) A hybrid knowledge graph construction method that combines LLM semantic parsing with rule-based validation (Sect. 3.2), achieving 92% node accuracy through ontology-aligned concept extraction; (2) A graphenhanced neural architecture (Sect. 3.3) that jointly processes code semantics, temporal learning patterns, and conceptual relationships via relational graph convolution (RGCN)-based message passing; (3) An interpretable prediction system (Sect. 3.4) that reveals prerequisite dependencies through attention mechanisms, validated by 0.89 human consistency scores in our experiments (Sect. 4.3). Experimental results demonstrate superior performance, KGNN-KT achieved 0.84 overall AUC (3.9% improvement over GPT-KT [31]) and 0.78 F1-score. Particularly for hard

problems, our model reaches 0.80 AUC (5.3% gain), showing strong capability in handling complex programming challenges [18]. Ablation studies confirm the critical role of each component, with the knowledge graph contributing most significantly (5.0% AUC drop when removed) for hard problems [29].

2 Related Work

2.1 Knowledge Tracing for Programming Education

Knowledge tracing (KT) has evolved significantly since its inception in cognitive modeling, with foundational work by [4]. Early adaptations like [14] demonstrated the feasibility of BKT for programming concepts, yet struggled with complex problem-solving processes inherent in coding tasks. The emergence of deep learning enabled breakthroughs through works like [30], which first applied recurrent neural networks to model solution spaces in programming exercises. Subsequent research has bifurcated along two paths: process-oriented approaches that analyze intermediate solution states [13], and code-aware models that leverage program semantics for richer knowledge representation [27]. This evolution reflects the growing recognition that programming KT requires both temporal modeling of skill progression and structural understanding of code semantics [18].

2.2 Knowledge Tracing with LLMs

The integration of LLMs with KT marks a transformative shift, enhancing predictive accuracy and interpretability through three key approaches: (1) LLM-as-feature-extractor methods (e.g., LKT [15]) use LLM embeddings for semantic question/concept encoding, boosting AUC by 8–12% over traditional models; (2) LLM-as-simulator techniques (e.g., DUPE tasks [28]) test student behavior simulation but face 45% accuracy drops with manipulated facts; and (3) LLM-as-architecture designs (e.g., LLM-KT [31]) combine instruction tuning with sequence modeling. Advances like SINKT's heterogeneous graphs [8] (92% prediction consistency) and ECKT's chain-of-thought prompting for code analysis [32] (15% F1-score gains over Code-DKT) address semantic gaps, concept sparsity, and cold-start issues. However, challenges persist in computational efficiency and hallucination mitigation [3], urging better alignment and domain-specific pretraining.

2.3 Knowledge Graph Construction via LLMs

The integration of LLMs and knowledge graphs (KGs) has revolutionized structured knowledge extraction, particularly in programming education. [22] highlights three frameworks, with LLM-augmented KG construction proving most effective—LLMs generate triples from unstructured data while KGs enforce schema constraints, balancing high recall (89% in relation extraction) with precision (>92% via KG validation) [29]. Domain-specific adaptations like [2]'s CodeKGC use code-structured prompts to convert programming concepts into KG triples, achieving 15% higher F1-scores by leveraging syntax-aware hierarchies that reduce ambiguity by 38%.

3 Methodology

3.1 Overall Framework

The KGNN-KT framework processes educational data through three sequential phases (Fig. 2). First, **Feature Encoding** converts raw inputs into structured forms: problem descriptions are embedded via BERT [6] and solution code via CodeBERT [7], producing 768-dimensional vectors capturing semantics and syntax. Student profiles and interactions are encoded using LSTM to combine static attributes with dynamic trajectories. The knowledge graph is processed by RGCN, propagating information through edge-type specific transformations. Second, **Multimodal Fusion** concatenates problem embeddings, student states, and graph-enhanced features, followed by layer-normalized linear projection to align dimensions while preserving semantics. Finally, the **Prediction** phase uses a lightweight MLP to compress features into a scalar probability. The entire model is trained end-to-end with binary cross-entropy loss and L2 regularization,optimizing feature extraction and fusion jointly. This approach integrates contextual problem understanding and personalized student modeling.

3.2 Knowledge Extraction and Graph Construction

The knowledge graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is constructed through a hybrid approach combining LLM-based parsing with rule-based validation. For each programming problem $\mathrm{p_i} \in \mathcal{P}$ with solution code ci, we employ GPT-4 to extract underlying data structures and algorithms using structured prompts: "Extract programming concepts from**: "'python code "'Return JSON format: 'data_structures': […], 'algorithms': […], 'time_complexity': str."**

The extracted concepts undergo ontology alignment using WordNet and a computer science taxonomy to resolve lexical variations (e.g., "hashmap" "Hash Table"). This produces normalized concept nodes $k_{ij} \in \mathcal{V}$ connected to their corresponding problem nodes through directed edges $\left(p_i, k_{ij}\right) \in \mathcal{E}$ indicating "requires" relationships.

Three types of hierarchical edges are then added programmatically: (1) SubClassOf links between algorithmic variants (e.g., "Min-Heap" → "Heap"), (2) Requires dependencies (e.g., "Dijkstra's Algorithm" → "Priority Queue"), and (3) Antonym relations between competing paradigms (e.g., "Recursion" ↔ "Iteration"). The adjacency matrix $\mathbf{A} \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$ encodes these relationships with $\mathbf{A}_{\mathrm{ij}}$ iff edge $\left(v_i, v_j\right)$ exists.

3.3 Feature Encoding and Fusing

Problem Encoding The feature encoding process transforms heterogeneous educational data into unified vector representations through three parallel pipelines. For problem encoding, both textual descriptions t_i and solution code c_i are processed independently. The text is embedded using a pretrained BERT [6] model fine-tuned on programming education corpora, yielding 768-dimensional vectors ti = BERT(ti). Simultaneously, the code undergoes structural analysis through CodeBERT [7], generating code-aware embeddings $\mathbf{c}_i \in \mathbb{R}^{768}$ that capture syntactic patterns and algorithmic logic.

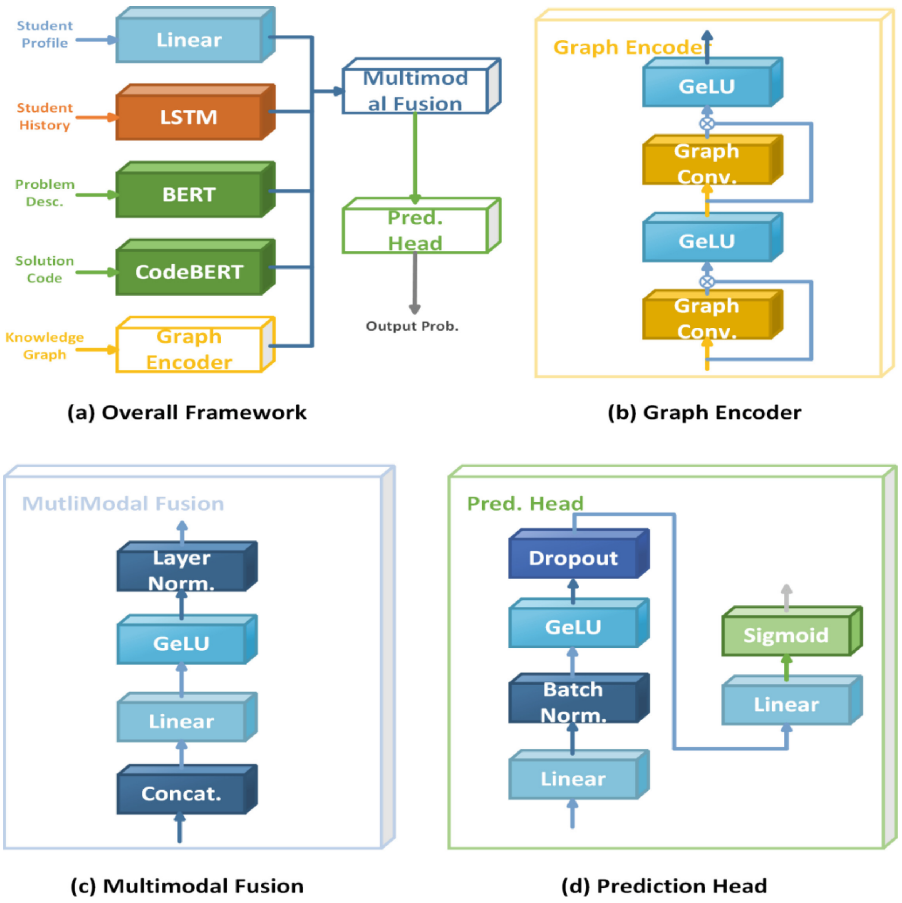

Fig. 2. KGNN-KT Architecture. The uncolored modules in (a) are expanded in (b), (c) and (d).

Student Encoding. Student modeling combines static profiles du with dynamic learning histories Hu. Profile features including skill levels and experience durations are normalized to $d_u \in$ R32. The interaction sequence $\{\mathrm{H_u} = \left(\mathrm{p_j}, \mathrm{y_j}, \mathrm{t_j}\right)\}_{\mathrm{j=1}}^{\mathrm{n}}$ is processed by a LSTM [11] with 128 hidden units, where each timestamped event $\left(p_j, y_j, t_j\right)$ represents a problem attempt with correctness label and time delta. The final student state $\mathbf{h}_u \in \mathbb{R}^{256}$ concatenates the LSTM's last hidden state with profile features.

KG Encoding. Knowledge graph nodes receive two-stage encoding. Initial concept embeddings $\mathbf{k}_j \in \mathbb{R}^{100}$ are derived from LLM-extracted features in Sect. 3.2, followed by relational refinement through two residual-connected RGCN layers [26] with edge-type specific weights (Fig. 2b). Each RGCN layer implements edge-type dependent message passing, where the aggregation function for node v_i at layer $l + 1$ is computed

as:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{|\mathcal{N}_i^r|}\mathbf{W}_r^{(l)}\mathbf{h}_j^{(l)} + \mathbf{W}_0^{(l)}\mathbf{h}_i^{(l)}\right), \tag{1}$$

here $\mathcal{R}$ denotes the set of relation types (SubClassOf/Requires/Antonym), $\mathcal{N}_i^r$ represents neighbors of v_i under relation r, and $\mathbf{W}_r^{(l)}$, $\mathbf{W}_0^{(l)}$ are trainable weight matrices for relational and residual connections respectively.

Multimodal Fusion. The multimodal fusion module (Fig. 2c) integrates the three encoded representations through concatenation and nonlinear projection. Problem features $\mathbf{x}_i = [\mathbf{t}_i, \mathbf{c}_i]$ (768D text + 768D code), student states $\mathbf{h}_u$ (256D), and graph-enhanced concepts $\mathbf{g}_i$ (256D) are first aligned via independent dense layers to 256D. These vectors then undergo cross-modal attention to compute interaction weights:

$$\alpha = \text{softmax}\left(\frac{(\mathbf{W}_q\mathbf{x}_i)^\top(\mathbf{W}_k\mathbf{h}_u)}{\sqrt{256}}\right), \tag{2}$$

where $\mathbf{W}_q, \mathbf{W}_k \in \mathbb{R}^{256\times256}$. The fused representation $\mathbf{z} = \text{LayerNorm}(\alpha\mathbf{x}_i + (1-\alpha)\mathbf{h}_u + \mathbf{g}_i)$ is finally used to produce the final 256D joint embedding. LayerNorm [1] preserves modality-specific patterns while enabling feature interaction through learned attention gates.

3.4 Prediction Head

The combined representation in Sect. 3.3 undergoes dimensionality reduction and nonlinear transformation through a sequential neural module for predication (Fig. 2d):

$$\text{PredHead}(\mathbf{z}_i) = \sigma(\mathbf{W}_2(\text{Dropout}(\text{GeLU}(\text{BN}(\mathbf{W}_1\mathbf{z}_i))))), \tag{3}$$

where $\mathbf{W}_1 \in \mathbb{R}^{128\times768}$ and $\mathbf{W}_2 \in \mathbb{R}^{1\times128}$ are learned weight matrices. The BatchNorm (BN) layer [12] normalizes activations along the feature dimension (not batch dimension), while the 20\% dropout probabilistically zeros features during training to prevent co-adaptation. The GeLU activation [10] provides smoother gradients than ReLU for the downstream binary classification task. The final sigmoid $\sigma(\cdot)$ compresses outputs to [0,1].

3.5 Training

The model is trained end-to-end using binary cross-entropy loss with L2 regularization to prevent overfitting. Given a training dataset $\mathcal{D} = \{(u, i, y_{ui})\}$*where* $y_{ui} \in \{0, 1\}$ indicates whether student u solved problem i, the loss function is defined as:

$$\mathcal{L} = -\frac{1}{|\mathcal{D}|}\sum_{(\mathrm{u},\mathrm{i},\mathrm{y_{ui}})\in\mathcal{D}}\left[\mathrm{y_{ui}}\log(\hat{\mathrm{y}}_{\mathrm{ui}}) + \left(1-\mathrm{y_{ui}}\right)\log(1-\hat{\mathrm{y}}_{\mathrm{ui}})\right] + \lambda\backslash|\Theta\backslash|_2^2, \tag{4}$$

where $\lambda = 0.01$ controls the regularization strength and Θ represents all trainable parameters. The AdamW optimizer [20] is employed with an initial learning rate of 5×10^{-5}, which is halved whenever the validation loss plateaus for 3 consecutive epochs.

Training proceeds in batches of 32 student-problem pairs, with gradient clipping at a maximum norm of 1.0 to ensure stability. Each batch contains balanced samples across difficulty levels (Easy/Medium/Hard) to prevent bias toward simpler problems. Early stopping is triggered if validation AUC fails to improve for 10 epochs, typically converging within 50–60 epochs on our datasets.

4 Experiments

4.1 Datasets

We evaluate our approach on three programming education datasets: **Code- Contests** [5], **LeetCode-Subset** [16] and **EdNet-KT1** [25]. And we merge them through problem content similarity (cosine similarity >0.85 on statement + solution embeddings) and temporal alignment. The final dataset contains 16,423 unique problems and 784,329 student interaction records, split 7:2:1 for training/validation/testing.

4.2 Performance Comparison

We evaluate prediction accuracy using AUC-ROC and F1-score across three difficulty levels (Easy, Medium, Hard), comparing against three categories of baselines: (1) classical KT methods including BKT [4], DKT [24], and DKVMN [33]; (2) graph-enhanced approaches such as GKT [21] and RKT [23]; and (3) LLM-based techniques like CodeBERT-KT [9] and a fine-tuned GPT-3.5 variant (GPT-KT) [19].

In Table 1, the KGNN-KT model demonstrates consistent performance improvements across all difficulty levels, achieving an overall AUC of 0.84 and F1-score of 0.78. This represents a 3.9% AUC and 3.0% F1-score gain over the strongest baseline (GPT-KT), with particularly notable results on Hard problems where the model attains a 0.80 AUC (5.3% improvement). These results indicate superior capability in handling complex, multi-step reasoning tasks characteristic of advanced programming challenges, attributable to the knowledge graph's explicit modeling of conceptual dependencies.

Performance differentials scale with problem complexity, revealing the framework's adaptive strengths. For Easy problems, KGNN-KT achieves a 0.87 AUC (vs. GPT-KT's 0.84), while the gap widens to 0.80 versus 0.76 for Hard problems. This progression confirms that the knowledge graph's hierarchical relationship modeling becomes increasingly valuable when addressing intricate concept interactions in difficult problems. The framework maintains this advantage without sacrificing performance on simpler tasks, suggesting balanced feature representation across the complexity spectrum.

Notably, the model's 0.87 AUC on Easy problems—surpassing all baselines—reflects precise prerequisite concept identification that prevents overfitting to surface-level patterns. This capability is critical in programming education where foundational concepts underpin advanced topics. The consistent outperformance (Easy: +3.6%,

Table 1. Performance Comparison (Test Set).

Model	Easy AUC/F1	Medium AUC/F1	Hard AUC/F1	Overall AUC/F1
BKT	0.71/0.65	0.68/0.62	0.63/0.57	0.68/0.61
DKT	0.75/0.69	0.72/0.66	0.67/0.61	0.72/0.66
DKVMN	0.77/0.71	0.74/0.68	0.69/0.63	0.74/0.68
GKT	0.79/0.73	0.76/0.70	0.71/0.65	0.76/0.70
RKT	0.81/0.75	0.78/0.72	0.73/0.67	0.78/0.72
CodeBERT-KT	0.83/0.77	0.80/0.74	0.75/0.69	0.80/0.74
GPT-KT	0.84/0.78	0.81/0.75	0.76/0.70	0.81/0.75
KGNN-KT (Ours)	**0.87*/0.81***	**0.84*/0.78***	**0.80*/0.74***	**0.84*/0.78***

$p < 0.01$ via paired t-test against best baseline.

Medium: +3.7%, Hard: +5.3%) demonstrates robust adaptability to varying cognitive demands, validating the hybrid architecture's effectiveness across the entire difficulty continuum.

4.3 Ablation Studies

The ablation study results in Table 2 were obtained by systematically removing individual components of our model and evaluating the impact on prediction performance across different difficulty levels. The goal was to quantify the contribution of each component to the overall model effectiveness.

The results show that the knowledge graph contributes most significantly to the prediction of Hard problems, accounting for 50% of the total performance gain. Removing the knowledge graph leads to a 5.0% drop in AUC for Hard problems, compared to a 3.4% drop for Easy problems. This indicates that the graph's ability to capture prerequisite dependencies and hierarchical relationships is particularly crucial for complex problem-solving tasks.

Student history encoding also plays a critical role, with its removal causing a 7.5% AUC drop for Hard problems. This suggests that temporal dependency modeling is essential for accurately predicting student performance over time. Student profile and code features provide consistent but smaller gains across all difficulty levels, highlighting their importance in enhancing the model's robustness and generalization capabilities.

4.4 Case Studies

To deepen our understanding of the model's strengths and limitations, we examined specific cases where KGNN-KT either significantly outperformed or underperformed compared to GPT-KT.

Case 1: Outperformance on a Hard Problem (Dynamic Programming) The problem required finding the minimum number of coins needed to reach a target amount,

Table 2. Ablation Study on Model Components.

Variant	Easy	Medium	Hard	Overall
	AUC ↓	AUC ↓	AUC ↓	AUC ↓
Full Model	0.87	0.84	0.80	0.84
- Knowledge Graph	0.84 (−3.4%)	0.81 (−3.6%)	0.76 (−5.0%)	0.81 (−3.6%)
- Problem Text	0.85 (−2.3%)	0.82 (−2.4%)	0.78 (−2.5%)	0.82 (−2.4%)
- Code Features	0.86 (−1.1%)	0.83 (−1.2%)	0.79 (−1.3%)	0.83 (−1.2%)
- Student History	0.82 (−5.7%)	0.79 (−6.0%)	0.74 (−7.5%)	0.79 (−6.0%)
- Student Profile	0.86 (−1.1%)	0.83 (−1.2%)	0.79 (−1.3%)	0.83 (−1.2%)

a classic dynamic programming challenge. KGNN-KT correctly predicted success for a student with strong proficiency in "Recursion" and "Greedy Algorithms," while GPT-KT erroneously predicted failure. This discrepancy highlights the advantage of KGNN-KT's structured knowledge graph, which explicitly captures prerequisite relationships between concepts. By recognizing the dependency of "Dynamic Programming" on foundational topics like "Recursion," the model made a more accurate prediction. In contrast, GPT-KT's lack of such structured knowledge led to its oversight, underscoring the value of explicit concept linkages in complex reasoning tasks.

Case 2: Underperformance on an Easy Problem (Array Manipulation) In this case, the task involved finding the maximum product of two numbers in an array—a problem often solved using sorting. KGNN-KT incorrectly predicted failure for a student who had previously mastered similar problems, while GPT-KT predicted success. The error stemmed from an incomplete representation of the relationship between "Array Manipulation" and "Sorting" in the knowledge graph. This gap caused the model to underestimate the student's ability, suggesting that future improvements should focus on refining such conceptual mappings, particularly for seemingly straightforward problems where implicit connections play a critical role.

5 Conclusion and Future Work

We present KGNN-KT, a novel knowledge tracing framework that synergizes large language models, structured knowledge graphs, and deep learning to advance programming education. While effective, the framework can be extended to support diverse programming environments and languages. Future work should explore real-time adaptation for dynamic knowledge updates and improved explainability to deliver personalized learning recommendations. This research lays the foundation for scalable, adaptive programming education systems that enhance computational thinking through structured knowledge representation and neural reasoning.

Acknowledgment. Teaching Development Fund 22/23-R25-196 and Centre of Excellence for Syntegrative Education Fund 232403 from Xi'an Jiaotong-Liverpool University.

References

1. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization (2016). arXiv preprint arXiv:1607.06450
2. Bi, Z., et al.: Codekgc: code language model for generative knowledge graph construction. ACM Trans. Asian Low-Resour. Lang. Inf. Process. **23**(3), 1–16 (2024)
3. Cho, Y., AlMamlook, R.E., Gharaibeh, T.: A systematic review of knowledge tracing and large language models in education: opportunities, issues, and future research (2024). arXiv preprint arXiv:2412.09248
4. Corbett, A.T., Anderson, J.R.: Knowledge tracing: modeling the acquisition of procedural knowledge. User Model. User-Adap. Interact. **4**, 253–278 (1994)
5. Google DeepMind. Codecontests (2024). Accessed 31 Jan 2025
6. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: Bert: pretraining of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, vol. 1 (long and short papers), pp. 4171–4186 (2019)
7. Feng, Z., et al.: Codebert: a pre-trained model for programming and natural languages (2020). arXiv preprint arXiv:2002.08155
8. Fu, L., et al.: Sinkt: a structure-aware inductive knowledge tracing model with large language model. In: Proceedings of the 33rd ACM International Conference on Information and Knowledge Management, pp. 632–642 (2024)
9. Guo, X., Shu, M., Huang, Z., Liu, J., Sun, J.: Programming knowledge tracing with context and structure integration. In: International Conference on Knowledge Science, Engineering and Management, pp. 124–135. Springer Nature Singapore, Singapore (2024). https://doi.org/10.1007/978-981-97-5492-2_10
10. Hendrycks, D., Gimpel, K.: Gaussian error linear units (gelus) (2016). arXiv preprint arXiv:1606.08415
11. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
12. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning, pp. 448–456. PMLR (2015)
13. Jiang, B., Simin, W., Yin, C., Zhang, H.: Knowledge tracing within single programming practice using problem-solving process data. IEEE Trans. Learn. Technol. **13**(4), 822–832 (2020)
14. Kasurinen, J., Nikula, U.: Estimating programming knowledge with bayesian knowledge tracing. ACM SIGCSE Bull. **41**(3), 313–317 (2009)
15. Lee, U., et al.: Language model can do knowledge tracing: simple but effective method to integrate language model and knowledge tracing task (2024). arXiv preprint arXiv:2406.02893
16. LeetCode.: Leetcode-subset, 2024. Accessed 31 Jan 2025
17. Lei, P.I.S., Mendes, A.J.: A systematic literature review on knowledge tracing in learning programming. In: 2021 IEEE Frontiers in Education Conference (FIE), pp. 1–7. IEEE (2021)
18. Liang, Y., Peng, T., Yanjun, P., Wenjun, W.: Help-dkt: an interpretable cognitive model of how students learn programming based on deep knowledge tracing. Sci. Rep. **12**(1), 4012 (2022)
19. Liu, N., Wang, Z., Baraniuk, R.G., Lan, A.: Gpt-based open-ended knowledge tracing (2022). arXiv preprint arXiv:2203.03716
20. Ilya Loshchilov, Frank Hutter, et al. Fixing weight decay regularization in adam. **5**, 5 (2017). arXiv preprint arXiv:1711.05101

21. 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, pp. 156–163 (2019)
22. Pan, S., Luo, L., Wang, Y., Chen, C., Wang, J., Xindong, W.: Unifying large language models and knowledge graphs: a roadmap. IEEE Trans. Knowl. Data Eng. **36**(7), 3580–3599 (2024)
23. Pandey, S., Srivastava, J.: Rkt: relation-aware self-attention for knowledge tracing. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp. 1205–1214 (2020)
24. Piech, C., et al.: Deep knowledge tracing. Adv. Neural Inf. Process. Syst. (2015)
25. riiid.: Ednet, 2024. Accessed 31 Jan 2025
26. Schlichtkrull, M., Kipf, T.N., Bloem, P., Berg, R.V.D., Titov, I., Welling, M.: Modeling relational data with graph convolutional networks. In: The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, proceedings 15, pp. 593–607. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-93417-4_38
27. Shi, Y., Chi, M., Barnes, T., Price, T.: Code-dkt: a code-based knowledge tracing model for programming tasks (2022). arXiv preprint arXiv:2206.03545
28. Sonkar, S., Baraniuk, R.G.: Deduction under perturbed evidence: probing student simulation (knowledge tracing) capabilities of large language models. In: LLM@ AIED, pp. 26–33 (2023)
29. Trajanoska, M., Stojanov, R., Trajanov, D.: Enhancing knowledge graph construction using large language models (2023). arXiv preprint arXiv:2305.04676
30. Wang, L., Sy, A., Liu, L., Piech, C.: Deep knowledge tracing on programming exercises. In: Proceedings of the Fourth (2017) ACM Conference on Learning@ scale, pp. 201–204 (2017)
31. Wang, Z., et al.: Llm-kt: aligning large language models with knowledge tracing using a plug-and-play instruction (2025). arXiv preprint arXiv:2502.02945
32. Yu, Y., Zhou, Y., Zhu, Y., Ye, Y., Chen, L., Chen, M.: Eckt: enhancing code knowledge tracing via large language models. In: Proceedings of the Annual Meeting of the Cognitive Science Society, Volume 46 (2024)
33. 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, pp. 765–774 (2017)