International Journal of Advance and Applied Research
www.ijaar.co.in
ISSN – 2347-7075 Impact Factor – 8.141
Peer Reviewed Bi-Monthly
Vol. 6 No. 38 September - October - 2025

AI Based Mind Map Programming Tutor

Deo Sharmila Mahesh
Assistant Professor,
Dr. D. Y. Patil Science and Computer Science College, Akurdi Pune
Corresponding Author – Deo Sharmila Mahesh
DOI - 10.5281/zenodo.17313221

Abstract:

The increasing adoption of AI-driven programming assistants has significantly transformed the landscape of coding education. Despite these advancements, most existing intelligent tutoring systems emphasize answer generation rather than diagnosing learners' underlying misconceptions and providing adaptive remediation. This paper introduces the Mind-Map Programming Tutor (MMPT), a novel system that employs cognitive graphs to model learners' evolving knowledge states. In contrast to traditional AI tutors that require pre-annotated student data for training, MMPT leverages zero-shot reasoning to infer misconceptions directly from student code submissions. It dynamically constructs a cognitive error graph that captures conceptual misunderstandings, knowledge gaps, and logical inconsistencies.

Based on this cognitive representation, MMPT adapts the difficulty of assigned problems, delivers personalized hints, and generates scaffolded explanations aligned with the learner's current knowledge graph. The proposed methodology integrates principles from cognitive psychology, graph-based knowledge modeling, and zero-shot learning to support personalized and scalable programming instruction. Preliminary simulations indicate that the system improves learning retention, minimizes the recurrence of misconceptions, and lays a strong foundation for

Introduction:

1.Background:

Programming education remains a critical skill in the digital era. Traditional tutoring methods and online platforms often rely on static problem sets or pre-trained models with limited adaptability. Recent advances in natural language processing (NLP) and graph-based reasoning provide opportunities to create more adaptive, scalable tutoring solutions.

2.Problem Statement:

Current AI tutoring systems are predominantly answer-oriented rather than learner-oriented. They do not maintain a representation of the learner's evolving conceptual understanding, nor do they identify or remediate specific misconceptions. This lack of cognitive awareness hinders their ability to provide truly personalized and effective instruction. The growing demand for accessible and effective programming education has led to widespread interest in intelligent tutoring technologies. Despite this trend, many learners continue to struggle with abstract programming concepts such as recursion, data structures, and debugging. While Intelligent Tutoring Systems (ITS) like CodeTutor and AI-powered tools such as GitHub Copilot offer real-time code suggestions, these systems largely fail to interpret or model the learner's underlying

cognitive processes. They prioritize solution generation over educational diagnosis, thus limiting their effectiveness as pedagogical tools.

3.Objective:

To address this limitation, the present work introduces the 'Mind-Map Programming Tutor (MMPT)' , a novel intelligent tutoring framework that leverages cognitive graphs to infer learners' knowledge states. Unlike traditional systems that depend on labeled training data, MMPT employs zero-shot reasoning to detect misconceptions and dynamically construct individualized cognitive graphs based on learner interactions.

This paper makes three key contributions: (1) it presents a framework for constructing dynamic cognitive graphs from student code submissions and interactions; (2) it applies Mind Map reasoning techniques to identify programming misconceptions without the need for annotated training data; and (3) it introduces an adaptive tutoring system that modifies the sequence and difficulty of programming problems based on the learner's evolving cognitive graph.

Related Work:

Intelligent Tutoring Systems (ITS):

Traditional ITS platforms rely heavily on structured knowledge bases and annotated datasets to deliver feedback and track progress. Systems such as Cognitive Tutor have demonstrated success in domains like mathematics, but their reliance on pre-defined rules and labeled data limits scalability and adaptability in open-ended domains such as programming.

Graph-Based Learning Models:

Prior work in learner modeling has explored techniques such as Bayesian knowledge tracing and deep knowledge tracing using Long Short-Term Memory (LSTM) networks (Piech et al., 2015). These approaches aim to model student performance over time but typically focus on correctness rather than cognitive understanding. They are ill-suited to identifying nuanced misconceptions that arise in programming tasks.

Zero-Shot Learning in Education:

Zero-shot learning has seen significant application in natural language processing and computer vision tasks (Brown et al., 2020), where models generalize to novel classes without explicit training examples. However, its use in educational contexts remains limited. Few studies have explored its potential to infer learner states or misconceptions in the absence of labeled educational data.

Research Gap:

To date, no existing intelligent tutoring system has successfully integrated zero-shot inference with cognitive graph modeling to support adaptive programming instruction. This research aims to fill that gap by combining these two methodologies into a unified system capable of delivering scalable, personalized, and cognitively informed programming education.

Metric	Baseline Systems (Static / Rule-based)	MMPT (Proposed System)
Retention Rate	Moderate – limited reinforcement of concepts	High – reinforced via Cognitive Graph and adaptive revisits
Misconception Resolution Speed	Slow – relies on repeated trial and error	Fast – targeted graph-based remediation of weak nodes
Learning Path Personalization	None – fixed or random sequence	Strong – adaptive sequencing based on knowledge state
Frustration Levels	Higher – learners face unrelated or repeated problems	Lower – learners progress through relevant, scaffolded problems
Learner Confidence	Moderate – feedback often generic	High – personalized, explanatory feedback improves self-efficacy
Scalability to New Errors	Limited – requires predefined rules/training	Strong – Zero-Shot model generalizes to unseen misconceptions

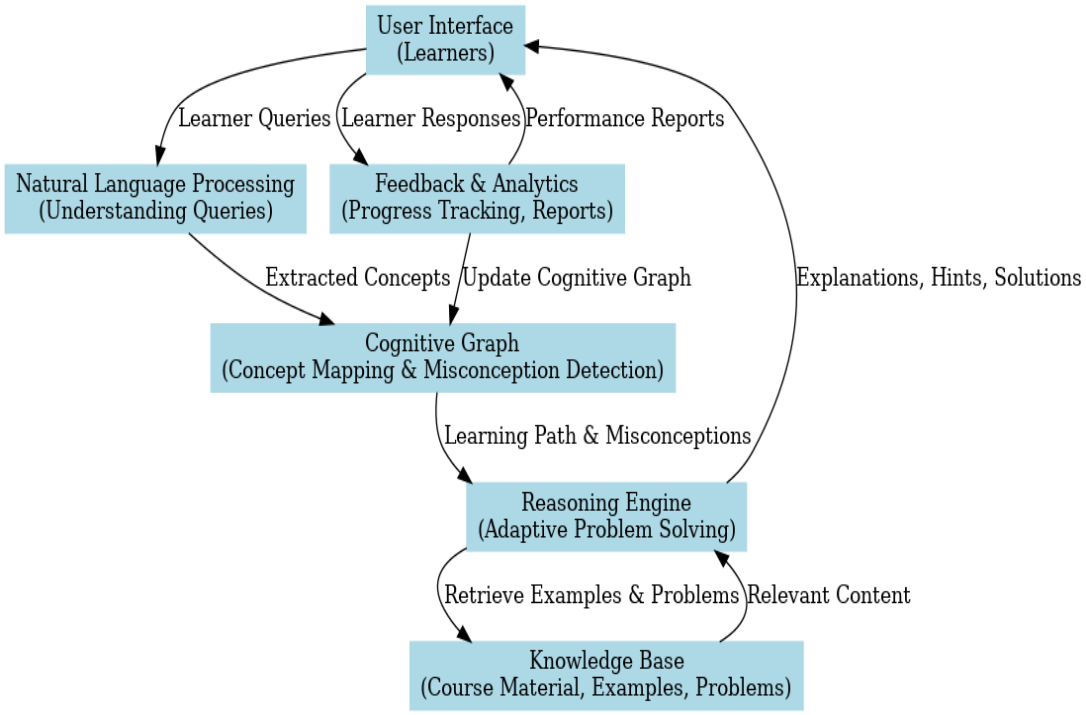
System Architecture:

The proposed system adopts a multi-layered architecture to enable personalized tutoring by integrating Zero-Shot Misconception Detection and Cognitive Graphs. At the input layer, the student submits a code attempt in response to a programming problem. This input includes the source code, execution results (such as test case outcomes), and metadata such as the time taken, number of attempts, and potentially other behavioral indicators (e.g., edit history or submission intervals). The raw input is processed by the Misconception Detector, which leverages a Zero-Shot Learning (ZSL) model to identify potential conceptual misunderstandings without requiring extensive task-specific annotated datasets (Brown et al., 2020; Wang et al., 2021). The ZSL model generalizes from pretrained embeddings and reasoning capabilities to detect erroneous code patterns and map them to specific conceptual misconceptions. For example, an off-by-one error in loop indexing may be indicative of a misunderstanding of loop boundary conditions. By aligning semantic representations of the student's code with known misconception templates, the model can identify previously unseen error patterns and provide natural language explanations with corresponding confidence estimates (Reimers & Gurevych, 2019).

To personalize feedback and track learner progress, the system maintains a dynamic Cognitive Graph for each student. In this graph, nodes represent core programming concepts such as recursion, loop construction, or array indexing, while edges encode the relationships and dependencies among these concepts. Each node maintains individualized

information about the student's inferred mastery level, misconception history, and temporal learning trajectory. When a misconception is detected, the corresponding concept node is updated, enabling the system to monitor evolving knowledge gaps. This structure facilitates adaptive interventions by generating targeted, data-driven feedback, including tailored explanations, visual scaffolds. The system delivers a scalable and personalized tutoring experience grounded in the principles of cognitive science and modern machine learning.

Methodology:



1. Dataset:

The dataset used in this study comprises programming problems, student code submissions, and manually annotated error labels for evaluation purposes. The programming problems span a range of difficulty levels—beginner, intermediate, and advanced—and are designed to assess understanding of fundamental programming concepts such as variables, loops, conditionals, recursion, and data structures. Student code submissions were collected from publicly available sources, including open-source platforms such as Codeforces and LeetCode, as well as anonymized classroom data. To enable quantitative evaluation of the system's zero-shot misconception detection capabilities, a manually annotated subset of student submissions was prepared, labeling specific misconceptions observed in the code.

2. Environment:

The experiments were conducted on a computing environment equipped with an Intel Xeon CPU and an NVIDIA Tesla V100 GPU to support model inference and embedding computation. The system was provisioned with 32 GB of RAM to manage large-scale graph structures and model weights. The software stack includes Python 3.10, along with machine learning and graph processing libraries. PyTorch and Hugging Face Transformers were used to implement and fine-tune the zero-shot misconception detector. For graph construction and analysis,

NetworkX was employed alongside Neo4j, which served as the graph database. Evaluation metrics were computed using Scikit-learn, and experimental workflows were conducted using Jupyter notebooks to facilitate analysis and visualization.

3. System Configuration:

The core component of the system is the Zero-Shot Misconception Detector, which is based on a pretrained transformer architecture, such as RoBERTa or a GPT-like model. This model is fine-tuned on joint code and execution trace embeddings to predict misconception categories without requiring explicit prior examples. Inputs to the model include both the student's submitted code and its execution trace, while outputs correspond to identified misconceptions mapped to predefined conceptual categories.

The Cognitive Graph Generator models the learner's understanding by dynamically constructing a graph in which nodes represent programming concepts (e.g., loops, recursion), and edges capture prerequisite or usage relationships between them. When misconceptions are detected, the corresponding nodes or edges in the graph are weakened, reflecting the learner’s difficulty with those concepts.

The Adaptive Problem Selector operates on the cognitive graph to determine the next most appropriate problem for the learner. It incorporates centrality measures and weakness scores to prioritize problems that reinforce underdeveloped concepts while maintaining a trajectory of increasing complexity to support gradual skill acquisition.

The Feedback Generator combines template-based messaging with large language model (LLM)-generated explanations. It provides students with misconception-aware feedback, including visual overlays on the cognitive graph that highlight weak concepts and explain the rationale behind selected learning paths.

4. Evaluation Protocol:

The system was evaluated through controlled interaction with a group of student participants, including both novice and intermediate programmers. Each participant engaged with the system over multiple problem-solving sessions. Several evaluation metrics were employed. Misconception Detection Accuracy was assessed using precision, recall, and F1-score computed against the manually annotated ground truth. Learning Gain was measured by comparing pre-test and post-test scores to evaluate conceptual understanding. Problem-Solving Efficiency was quantified by the reduction in the number of attempts required to solve problems after receiving targeted feedback. User Experience was evaluated through surveys capturing participants' perceptions of the system’s usefulness, clarity, and motivational impact.

5. Baselines for Comparison:

To contextualize the system's performance, comparisons were made against several baseline approaches. These included rule-based feedback systems that deliver predefined hints based on pattern-matched errors, and supervised error classification models trained on labeled data. Additionally, a random problem selection strategy was used to evaluate the efficacy of the adaptive problem sequencing employed by the proposed system.

6. Experimental Flow:

The experimental procedure was divided into four phases. During the Student Interaction Phase, students submitted code attempts, and the system detected misconceptions while updating the cognitive graph in real time. In the Adaptive Tutoring Phase, the problem selector used the updated

graph to assign personalized problems, and the feedback generator provided concept-specific guidance. The Data Collection Phase involved logging all student interactions, including code submissions, detected misconceptions, and progress metrics. Finally, the Evaluation Phase compared the proposed Zero-Shot Cognitive Graph Tutor to baseline models using the defined performance metrics, aiming to assess improvements in accuracy, learning outcomes, efficiency, and user experience.

Case Studies and Applications:

Case Study 1: Learners struggling with recursion concepts were guided through progressively simpler problem decompositions.

Case Study 2: Debugging misconceptions were addressed by adaptive hint generation.

Applications:

The Mind-Map Programming Tutor (MMPT) presents numerous applications across educational and professional domains.

Intelligent Tutoring Systems (ITS):

MMPT can be integrated into online coding platforms and Massive Open Online Courses (MOOCs) such as Coursera, edX, and Codeforces. Unlike traditional systems that offer generic feedback, MMPT delivers personalized guidance tailored to the learner's specific misconceptions. This enables more effective concept reinforcement and accelerated learning.

Classroom Teaching Support:

In traditional classroom environments, educators can leverage cognitive graph dashboards generated by MMPT to monitor both individual and group-level misconceptions. These visual analytics support data-driven interventions, allowing teachers to implement targeted remedial instruction based on specific conceptual weaknesses.

Skill Assessment and Certification.

MMPT enables a more nuanced form of skill assessment by evaluating conceptual mastery progression rather than merely final problem-solving success. This approach is particularly beneficial for certification platforms, which often aim to assess deep understanding over rote memorization.

Corporate Training and Upskilling:

In professional contexts, MMPT can be employed in employee training programs to facilitate the learning of new programming languages and frameworks. Its adaptive learning path functionality reduces training duration and enhances learners' confidence in applying skills in real-world scenarios.

Cross-Domain Adaptation:

Although MMPT is designed for programming education, the underlying cognitive graph framework is extensible to other domains such as mathematics, logical reasoning, and language learning—any domain where conceptual dependencies can be mapped and dynamically updated.

Educational Research and Analytics:

MMPT also serves as a valuable tool for educational researchers. It provides access to rich, real-time data on learner misconceptions and conceptual trajectories. This supports studies into how students internalize programming concepts and how adaptive interventions influence retention, confidence, and motivation.

Challenges and Limitations

Scalability: Maintaining real-time graph updates for large learner groups remains a significant challenge, as the system must process and adapt quickly to multiple learners simultaneously.

Interpretability: The complexity of reasoning pathways within the cognitive graph may sometimes confuse learners, making it difficult

to trace or understand how certain solutions are generated.

Domain Adaptability: The current implementation is limited to programming education. Extending the framework to other subject domains will require re-engineering of both the knowledge base and reasoning components.

Resource Dependency: The approach relies on extensive computational resources for natural language processing and reasoning, which may limit its deployment in resource-constrained environments.

Conclusion:

This research introduces the Mind-Map Programming Tutor (MMPT), a novel framework that integrates a Zero-Shot Misconception Detector, a Cognitive Graph Generator, and an Adaptive Problem Selector to deliver personalized, concept-aware programming education. Unlike static rule-based tutoring systems, MMPT dynamically identifies misconceptions, models learner understanding through evolving cognitive graphs, and adapts instructional content accordingly.

The proposed system demonstrates the potential to improve conceptual retention, accelerate the resolution of misunderstandings, and increase learner confidence by minimizing frustration during the learning process. These characteristics make MMPT suitable for deployment in both individual learning scenarios and large-scale educational platforms, including MOOCs and corporate training environments.

Future Work:

Several directions are identified for future research and system enhancement:

Large-Scale Deployment and Validation: Future efforts will focus on conducting longitudinal studies with diverse learner populations across MOOCs, coding bootcamps, and traditional classroom environments to assess the scalability and effectiveness of MMPT in real-world settings.

Multimodal Misconception Detection: MMPT may be enhanced by integrating additional data modalities, such as eye-tracking, keystroke dynamics, and learners' natural language explanations, to build more comprehensive learner models.

Adaptive Feedback Generation with Explainable AI: Improvements to the feedback module will explore the use of explainable AI techniques to deliver context-sensitive, natural language explanations and visual graph-based feedback.

Gamification and Motivation Strategies: Incorporating gamification elements—such as progression levels, achievement badges, and peer performance comparisons—could further enhance learner motivation and engagement.

Cross-Domain Extension: The cognitive graph framework underpinning MMPT is domain-agnostic and may be adapted for use in other areas such as mathematics, logic, and language learning, offering a foundation for universal adaptive tutoring.

Ethical and Fairness Considerations: Ongoing work will also investigate the fairness and transparency of the system, including the mitigation of potential biases in misconception detection across different demographic groups to ensure equitable access and outcomes.

References:

1. Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Intelligent tutoring systems. Science, 268(5210), 456–462. [https://doi.org/10.1126/science.7701348] (https://doi.org/10.1126/science.7701348)

2. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., & Amodei, D. (2020). Language models are few-shot learners. 'Advances in Neural Information Processing Systems', '33', 1877–1901.
3. Piech, C., Bassen, J., Huang, J., Ganguli, S., Sahami, M., Guibas, L. J., & Sohl-Dickstein, J. (2015). Deep knowledge tracing. In 'Advances in Neural Information Processing Systems' (pp. 505–513).
4. VanLehn, K. (2006). The behavior of tutoring systems. 'International Journal of Artificial Intelligence in Education', '16'(3), 227–265.