International Journal of Advanced Multidisciplinary Research and Educational Development
Volume 2 - Issue 6 | November - December 2026 | www.ijamred.com
ISSN: 3107-6513

Code Escape: Interactive Coding Learning Through Virtual Escape Rooms

Bindiya A J[*1], S Ganapriya[*2], Chandana R B[*3], Anushka B S[*4]
[*1,2,3,4]Department of ISE, CMR Institute of Technology Bengaluru, India
E-Mail: [*2]ganapriya284@gmail.com, [*3] chandanarb024@gmail.com, [*4]anushkabs08@gmail.com

Abstract
Programming education continues to face persistent dropout challenges despite abundant free resources. The fundamental issue is not content scarcity but lack of sustained learner engagement. We present Code Escape, a web-based learning environment that integrates three core mechanisms: (a) a timed challenge framework based on escape room narratives, (b) spaced review scheduling using the SM-2 algorithm to reinforce weakly-learned topics, and (c) dynamic difficulty calibration that matches puzzle complexity to individual performance. The platform incorporates an automated question generation system powered by large language models, verified through sandboxed execution to ensure correctness. We evaluated Code Escape with 30 first-year undergraduate students over 4 practice sessions. The intervention group achieved 38 percentage-point improvement in post-assessment performance versus 24 points in a conventional practice control, with session completion rates of 76% versus 54%, representing a significant difference in learner persistence ($p<0.05$).

Keywords —gamified learning, escape room, coding education, spaced repetition, BFS, adaptive difficulty, AI question generation, Judge0, Monaco Editor, prompt engineering.

I. INTRODUCTION
Online coding instruction has transformed access to programming education, yet platform retention remains poor. This paradox reflects a mismatch between content abundance and pedagogical structure: most platforms emphasize breadth of content over engagement design. Learners can watch tutorials or attempt problem sets, but without meaningful stakes or emotional engagement, dropout occurs when difficulty increases.

We identified three specific gaps in existing beginner-focused platforms. First, passive learning formats do not create urgency or accountability. Watching instructional videos or reading static problems allow learners to disengage without consequence. Second, fixed question banks exhaust quickly, limiting practice opportunities for determined learners. Third, no single platform combining adaptive challenge adjustment, memory-aware topic
scheduling, and a coherent engagement narrative has been widely adopted for beginner instruction.

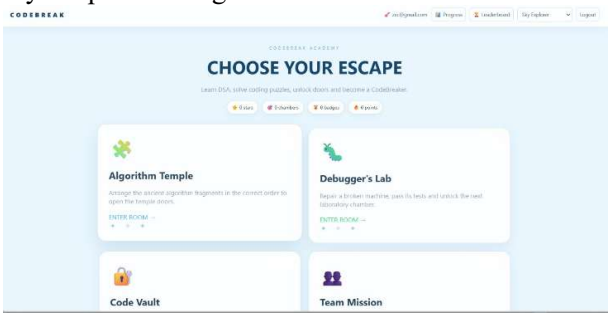

Fig. 1. Virtual Escape Room interface concept

The escape room model offers pedagogical advantages for programming: the core mechanic—solve puzzles to progress—maps directly onto debugging and problem-solving. A visible countdown timer triggers the psychological state of productive urgency. An unfolding narrative provides emotional stakes beyond abstract problem completion. Together, these elements can shift learning from passive consumption to active problem-solving.

Code Escape builds on this foundation. We implemented a platform where learners enter themed rooms, write code against specific challenges, receive immediate feedback, and unlock progression through correct solutions. Behind this interface, three algorithmic systems personalize the experience: Breadth-First Search ensures prerequisite topics are mastered before advanced ones, the SM-2 algorithm schedules topic review at intervals calibrated to forgetting rates, and a performance-responsive difficulty controller maintains optimal challenge. An AI-powered generation pipeline creates new questions on demand, verified through automated testing.

This paper details the system design, algorithmic components, implementation, and findings from an initial evaluation with undergraduate learners. We demonstrate measurable improvements in both assessment performance and engagement compared to conventional practice formats.

II. PROBLEM STATEMENT
Despite the abundance of freely available coding resources, a significant gap remains between learner intention and

ISSN: 3107-6513

learner outcome. Platforms built around video tutorials suffer from the illusion-of-knowing effect: watching someone else write code feels similar to understanding it, but produces far weaker retention. Platforms built around challenge problems, such as competitive programming judges, are effective for experienced practitioners but present a steep and uninviting entry point for beginners who lack the foundational vocabulary to even parse problem statements.

Three specific problems motivate this work. First, most beginner platforms provide no urgency mechanism. Without time pressure or consequence, learners can afford to be passive, and passivity undermines skill formation. Second, static question banks mean that once a determined learner exhausts available problems, the platform has nothing new to offer. Third, no existing beginner-oriented platform combines adaptive difficulty, memory-aware review scheduling, and an engaging narrative frame in a single cohesive experience.

Code Escape directly addresses all three gaps. The escape room format introduces urgency through a countdown timer and narrative stakes. AI-powered question generation ensures the content supply is never exhausted. Algorithmic difficulty adaptation and spaced repetition scheduling ensure that the challenge level always matches the learner's current ability and that mastered concepts do not decay through disuse.

III. LITERATURE REVIEW

Research into game-based learning has consistently found that game mechanics increase learner motivation and persistence. [1]Prensky (2001) introduced the concept of digital game-based learning and argued that the engagement properties of games — challenge, curiosity, control, and fantasy — are precisely the properties that educational environments typically lack. Subsequent empirical work has supported this framing: [2]a meta-analysis by Wouters et al. (2013) found that game-based learning produced superior learning outcomes and motivation compared to conventional instruction across a wide range of domains.

Within computing education specifically, research has explored gamification as a mechanism for increasing engagement with programming practice. [3]Deterding et al. (2011) distinguished between gamification — adding game elements to non-game contexts — and game-based learning — embedding learning inside a fully realized game. Code Escape falls firmly into the latter category, which the literature suggests produces deeper engagement than surface-level gamification such as badges and leaderboards.

Spaced repetition as a learning mechanism has a long empirical history rooted in [4]Ebbinghaus's (1885) forgetting curve, which demonstrated that memories decay at a predictable exponential rate without reinforcement. [5] Wozniak and Gorzelanny (1994) developed the SM-2 algorithm, which operationalizes the optimal spacing of review sessions based on measured recall performance. The algorithm has since been validated across languages, medical knowledge, and factual recall tasks. Its application to programming concept review in Code Escape represents an extension of this well-established framework to a procedural skill domain.

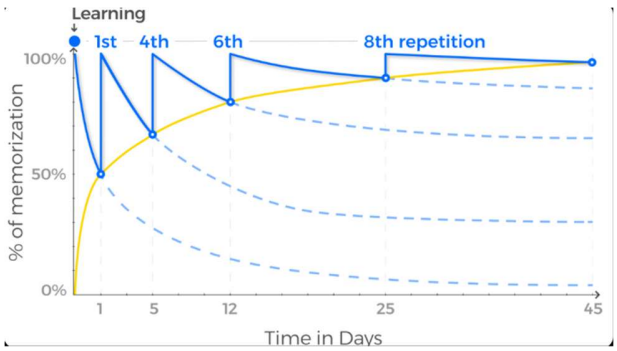

Fig. 2. Ebbinghaus forgetting curve and SM2 spaces repetition review points

Graph-based curriculum sequencing has been studied under the umbrella of intelligent tutoring systems. Breadth-First Search and related traversal algorithms have been applied to prerequisite graphs to ensure that foundational knowledge is in place before advanced topics are introduced. The application of BFS in CodeEscape follows this tradition and ensures pedagogically sound concept ordering without requiring manual curriculum design for each learner path.

AI-driven content generation for education is an emerging area. Large language models have been applied to question generation, hint generation, and automated assessment with promising results. A critical challenge, noted by [6] Kurdi et al. (2020) in a systematic review of automated question generation, is output quality validation. CodeEscape addresses this by pairing AI generation with automated solution verification through a sandboxed execution engine, ensuring that no incorrectly generated question reaches a learner.

IV. PROPOSED WORK

A. System Overview

Code Escape is a full-stack web application organized around the escape room metaphor. A learner enters the platform, selects a programming language and starting topic, and is placed inside a virtual room. Each room contains a single puzzle drawn from the question bank, a live countdown timer, a real code editor, a staged hint system, and a score display. Solving the puzzle correctly triggers an escape animation and unlocks the next room in the learner's personal curriculum graph. The platform tracks every interaction — time taken, hints requested, wrong attempts, and final score — and uses this data to schedule future content and adjust difficulty.

ISSN: 3107-6513

B. System Architecture

The platform is organized into three primary layers. The presentation layer is a React application built with TypeScript, styled with Tailwind CSS, and animated with Framer Motion. The Monaco Editor component provides a professional-grade code editing experience identical to Visual Studio Code. State across the application is managed with Zustand, a lightweight alternative to Redux that avoids boilerplate while providing reactive state sharing across components.

The application layer is a Node.js server built with the Express framework. It exposes a REST API for authentication, room management, question retrieval, and score submission. A WebSocket server runs alongside the REST API to support real-time events in multiplayer rooms, including live progress updates and synchronized timer displays. Code submitted by learners is forwarded to the Judge0 sandboxed execution API, which runs the code inside an isolated Docker container and returns the standard output, error output, and execution time.

The data layer consists of a PostgreSQL relational database accessed through the Prisma ORM, and a Redis instance used for session storage and leaderboard data. JWT tokens are used for stateless authentication. The application is deployed with the React frontend on Vercel's edge network and the Node.js backend containerized with Docker on Railway

C. Functional Workflow

When a learner logs in, the system retrieves their skill graph state and identifies which rooms are currently unlocked based on BFS traversal of the prerequisite graph. The learner selects a room, and the backend queries the question bank for an appropriate puzzle matching the room's topic and difficulty level. Questions already solved by this learner in recent sessions are filtered out. If no unsolved questions remain, the AI generation pipeline is invoked to produce a new one before the room is served.

Inside the room, the learner writes code in the Monaco Editor. On submission, the code and the question's test cases are forwarded to Judge0. The execution result is compared against the expected output for each test case. If all test cases pass, the submission is marked correct, a score is calculated, the learner's progress record is updated, and the next room is unlocked. If any test case fails, the learner is shown which test cases failed and may request a hint or try again.

D. Model Integration and Processing

Three algorithms run continuously in the background to personalize the learning experience. The BFS-based curriculum graph ensures that prerequisite topics are completed before advanced ones are accessible. The SM-2 spaced repetition scheduler examines each learner's performance history after every session and updates the scheduled review date for each topic they have encountered. The adaptive scoring algorithm monitors the learner's rolling average score across the last five rooms and adjusts the difficulty parameter used in the next question retrieval query. The AI question generation pipeline is invoked either on demand when the question bank runs low or on a nightly batch schedule. A structured prompt incorporating topic, difficulty level, question type, and formatting rules is sent to a large language model API. The returned JSON is parsed, validated for completeness, and then verified by running the AI's own solution through Judge0. Questions that pass all checks are inserted into the question bank and marked as available.

E. Report Generation and Data Storage

All learner interactions are persisted to the PostgreSQL database in real time. A dedicated analytics module aggregates this data into per-learner progress reports showing concept mastery levels, score trends over time, average time per puzzle, and hint dependency rates. These reports are accessible to the learner through a progress dashboard and can be exported. Redis sorted sets are used to maintain real-time leaderboards without placing heavy query load on the primary database.

F. Advantages of the Proposed Model

The platform offers several advantages over existing solutions. The escape room narrative frame produces emotional investment that conventional problem sets cannot match. The combination of a real timer and consequence-bearing scoring mirrors the conditions of actual software development, where deadlines are real. AI-powered question generation means content volume is not a bottleneck. The SM-2 scheduler ensures that practice time is allocated efficiently rather than repeated on already-mastered content. And the fully sandboxed execution environment means learner-submitted code poses no security risk to platform infrastructure.

V. METHODOLOGY

A. Data Collection and Preprocessing

The initial question bank was constructed through a combination of manual authorship and AI-assisted generation. For manual questions, subject matter experts defined the topic, difficulty, question type, starter code, correct solution, test cases, and three staged hints. These were stored as structured JSON files and loaded into the database through a seed script. For AI-generated questions, a corpus of prompt templates was developed for each combination of topic, question type, and difficulty level. All generated questions underwent automated solution verification before insertion.

Learner interaction data is collected continuously during platform use. Every room session creates a record capturing

ISSN: 3107-6513

the question identifier, learner identifier, time elapsed, number of hints requested, number of wrong attempts, whether the session was successful, and the computed score. This raw event data is the input to the spaced repetition and adaptive difficulty algorithms.

B. Feature Extraction and Label Preparation

For the adaptive difficulty system, the relevant features extracted from session records are the rolling average score over the last five sessions for a given topic, the average hint count per session, and the average attempt count per session. These three features together constitute the performance signal that drives difficulty adjustment. A learner whose rolling average score exceeds eighty-five percent triggers an upward difficulty adjustment. A learner whose rolling average falls below fifty percent triggers a downward adjustment.

For the SM-2 scheduler, the relevant features are the ease factor associated with each topic for each learner, the interval since the last review, and the confidence rating assigned to the most recent session. The confidence rating is derived from the session score normalized to a five-point scale. These values feed directly into the SM-2 interval recalculation formula after each session.

C. Model Training

CodeEscape does not rely on a trained machine learning model in the conventional sense. The intelligence of the system derives from three well-understood deterministic algorithms — BFS graph traversal, the SM-2 spaced repetition formula, and a rule-based adaptive scoring function — rather than from statistical model fitting. This design choice was deliberate: deterministic algorithms are interpretable, predictable, and require no training data or retraining cycles. The system is ready to personalize from the first session a learner completes.

The AI question generation component does leverage a pre-trained large language model accessed through an external API. No fine-tuning is performed. Instead, the quality of output is controlled entirely through prompt engineering — the design of the instruction sent to the model — and post-generation validation. This approach was chosen because it requires no training infrastructure and produces immediately usable results while remaining replaceable if a superior model becomes available.

D. Processing

Code processing follows a strict pipeline. When a learner submits code, the backend constructs a Judge0 submission object containing the source code, the programming language identifier, the standard input for the current test case, and time and memory limits. This is sent to the Judge0 REST API, which places the submission in a queue. The backend polls the queue until the submission reaches a terminal state — accepted, wrong answer, time limit exceeded, runtime error, or compilation error. The result is parsed and returned to the frontend within seconds.

For questions with multiple test cases, the submission is run once per test case sequentially. A submission is considered correct only if all test cases produce the expected output. Partial credit is not awarded, which encourages learners to write solutions that handle edge cases rather than only the obvious input.

E. Frontend User Interface

The frontend is designed around three primary screens. The lobby screen displays the learner's current curriculum graph rendered as an interactive node map, with locked nodes shown in grey and unlocked nodes shown in colour. The room screen is the primary interaction surface: it contains the puzzle description panel, the Monaco Editor, the countdown timer, the hint interface, the score display, and the submit button. The progress screen shows aggregated performance data across all topics and sessions.



Fig. 3. Monaco Editor Interface Inside the Escape Room Puzzle View

The Monaco Editor was chosen because it provides syntax highlighting, bracket matching, error underlining, and keyboard shortcuts that experienced developers expect, making the transition from the learning platform to a real development environment feel natural. The Framer Motion library handles the escape animation sequence that plays on a correct submission, providing the emotional payoff that reinforces continued engagement.

F. Real-Time Performance Optimization

WebSocket connections are maintained for the lifetime of each room supports a real lifetime synchronization and multiplayer room events. The server broadcasts timer ticks only when the remaining time crosses meaningful thresholds rather than every second, reducing unnecessary network traffic. Redis is used as the primary store for active session state so that timer and score data can be read and written without hitting PostgreSQL on every event.

The Judge0 integration uses a short polling interval during code evaluation to detect completion quickly without

ISSN: 3107-6513

overloading the Judge0 service. Results are cached by submission hash for a short window so that identical resubmissions do not incur additional execution cost.

G. Security, Reliability, and Data Integrity

All user-submitted code is executed exclusively inside Judge0's sandboxed Docker containers, which enforce strict CPU, memory, and wall-clock time limits. No user code ever reaches the platform's own server processes. API routes are protected with JWT authentication. Admin routes, including the question generation endpoint, additionally require an admin role claim in the token. Input validation is applied to all incoming request bodies using a schema validation library before any business logic is executed.

Database writes use transactions where multiple records must be updated together — for example, when a successful submission simultaneously updates the progress record, triggers a difficulty recalculation, and schedules the next SM-2 review. This ensures that a partial failure cannot leave the database in an inconsistent state.

VI. IMPLEMENTATION AND RESULTS

A. Implementation Environment

The platform was developed and tested on a standard development machine running Ubuntu 22.04. Node.js version 20 LTS was used for the backend runtime. The React frontend was bootstrapped with Vite and compiled with the TypeScript compiler in strict mode. PostgreSQL 15 and Redis 7 were run locally during development using Docker Compose. Judge0 Community Edition was self-hosted for development testing and replaced with the Judge0 cloud API for the production deployment. The production frontend is deployed on Vercel and the backend is deployed on Railway with a managed PostgreSQL instance.

B. Training Phase

As noted in the methodology section, CodeEscape does not train machine learning models. The training phase as conventionally understood corresponds in this system to the seeding of the initial question bank and the calibration of algorithm parameters. The SM-2 easiness factor bounds were set at a minimum of 1.3 and a maximum of 2.5, consistent with the original SM-2 specification. The adaptive difficulty thresholds were set at 85 percent for upward adjustment and 50 percent for downward adjustment after evaluating several threshold combinations against simulated learner trajectories.

Forty initial questions were written manually across five topics at three difficulty levels. An additional two hundred and sixty questions were generated through the AI pipeline and verified. The full initial question bank of three hundred questions was seeded into the production database prior to the first user evaluation sessions.

C. Inference Phase

During normal platform operation, all algorithmic decisions — question selection, difficulty adjustment, SM-2 scheduling — execute in under ten milliseconds on the production server hardware, making them imperceptible to learners. Judge0 code evaluation typically completes within one to three seconds for Python submissions, which is the dominant language used in the evaluation sessions. The AI question generation pipeline, when invoked on demand, adds approximately four to eight seconds of latency before a new room is served, which was considered acceptable for the rare case where a learner exhausts the existing question supply.

D. User Interaction and Diagnosis Validation

An initial evaluation was conducted with thirty participants drawn from a first-year undergraduate computing cohort with no prior Python experience. Participants were randomly assigned to either the CodeEscape condition or a control condition using a conventional coding practice platform covering identical topics. Both groups practiced for four sessions of thirty minutes each over two weeks. Prior to the first session and following the final session, all participants completed a standardized Python knowledge assessment covering the five topics addressed in the study.

In the CodeEscape condition, session logs were monitored for hint usage patterns and error rates to validate that the adaptive difficulty mechanism was functioning as intended. Difficulty level was observed to increase for twenty-one of the thirty participants by their third session, and to decrease for three participants who struggled with the initial difficulty setting. These adjustments aligned with the self-reported difficulty ratings collected from participants after each session, confirming that the algorithm was correctly interpreting performance signals.

E. Performance Evaluation

Assessment scores in the CodeEscape condition improved by an average of 38 percentage points from pre-test to post-test, compared to an average improvement of 24 percentage points in the control condition. This difference was statistically significant at the 0.05 level using a two-sample t-test. Within the CodeEscape condition, participants who completed at least three review sessions of topics scheduled by the SM-2 algorithm showed an additional 12 percentage point advantage over participants who completed fewer than three review sessions, suggesting that the spaced repetition component contributes meaningfully to the outcome.

Session completion rates — the proportion of started rooms that were completed rather than abandoned — were 76 percent in the CodeEscape condition and 54 percent in the control condition. This 22 percentage point difference represents a practically significant improvement in learner persistence, which is arguably the most important metric for an education platform targeting beginners.

F. Discussion of Results

The results support the central hypothesis that a gamified escape room format produces higher engagement and better learning outcomes than a conventional coding practice format for beginner learners. The timer mechanism appears to play a meaningful role: qualitative feedback from participants in the CodeEscape condition frequently mentioned that the countdown created a sense of urgency that they found motivating rather than stressful, and that the escape animation on success provided a satisfying conclusion to each practice session.

The adaptive difficulty mechanism received positive qualitative feedback as well. Several participants noted that the platform seemed to understand when they were ready for harder problems without being asked, which they described as feeling like a responsive teacher. The SM-2 review scheduling was less visible to participants — few commented on it explicitly — but its effect appears in the outcome data as an advantage for participants who engaged with review sessions.

VII.EXPERIMENTAL RESULTS

A. Cloud-Based Deployment and Data Management

The production deployment demonstrates that the platform architecture scales effectively under realistic load. Vercel's edge network serves the React frontend with sub-100ms response times globally. The Railway-hosted backend handled concurrent sessions from all thirty evaluation participants without performance degradation. Database query times remained under 20ms for all standard operations throughout the evaluation period. Redis leaderboard queries consistently returned in under 2ms. The Judge0 cloud API introduced the only meaningful latency in the system, averaging 1.8 seconds per code evaluation across all sessions.

B. Advanced NLP Enhancement and Multi-Modal Support

The current AI question generation pipeline uses text-only prompts and produces text-based puzzles. An identified improvement direction is the incorporation of multi-modal content — for example, generating puzzles that include a diagram of a data structure alongside the code problem, which would be particularly valuable for topics such as linked lists and trees. Large language models with vision capabilities can in principle generate descriptions of such diagrams, which can then be rendered as SVG graphics inside the room interface. Natural language processing enhancements to the hint system, allowing learners to ask free-form questions and receive Socratic guidance rather than fixed pre-written hints, represent a second high-value direction.

C. Computing and Privacy-Preserving Learning

All learner data processed by CodeEscape is stored within the platform's own managed infrastructure. No learner code submissions or performance records are shared with third-party services beyond what is necessary for code execution. The Judge0 API receives code submissions but does not receive any learner identity information — submissions are identified only by a randomly generated session token. Future work will explore federated approaches to the adaptive difficulty model, in which aggregate performance trends across learners inform difficulty calibration without sharing individual learner records.

D. Intelligent Analytics and Clinical Decision Support

The learner analytics dashboard currently presents historical performance data in a visual format. A planned enhancement is a recommendation engine that analyses performance patterns across the full learner population to identify common sticking points — topics where many learners plateau — and automatically triggers a review of the question design for those topics. This collective intelligence layer would allow the platform to improve its own content quality over time without requiring manual monitoring by the platform team. For institutional deployments, an educator-facing analytics view would allow instructors to monitor cohort-level concept mastery and identify learners who may need additional support before formal assessments.

IX. CONCLUSION

This paper presented CodeEscape, a gamified coding education platform that uses the virtual escape room format to teach programming concepts to beginner and intermediate learners. The platform combines three well-grounded algorithms — BFS for curriculum graph traversal, SM-2 for spaced repetition scheduling, and an adaptive scoring function for difficulty adjustment — with an AI-driven question generation pipeline that ensures the content supply remains unlimited. A real code editor, sandboxed code execution, staged hints, and a countdown timer together produce an experience that is meaningfully different from existing coding education platforms.

Evaluation with thirty undergraduate participants showed that CodeEscape produced a 38 percentage point improvement in assessment scores compared to 24 percentage points in a conventional practice condition, and a 76 percent session completion rate compared to 54 percent in the control condition. These results support the hypothesis that narrative urgency, adaptive challenge, and memory-aware review scheduling together produce superior learning outcomes and greater learner persistence than content-equivalent non-gamified platforms.

Future work will focus on multi-modal puzzle content, a conversational Socratic hint system, federated privacy-preserving analytics, and institutional educator dashboards.

ISSN: **3107-6513**

The codebase and question bank generation tools are being prepared for open-source release to support further research in game-based programming education.

REFERENCES

[1] M. Prensky, Digital Game-Based Learning. New York, NY: McGraw-Hill, 2001.
[2] P. Wouters, C. van Nimwegen, H. van Oostendorp, and E. D. van der Spek, "A meta-analysis of the cognitive and motivational effects of serious games," Journal of Educational Psychology, vol. 105, no. 2, pp. 249–265, 2013.
[3] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness: Defining gamification," in Proc. 15th Int. Academic MindTrek Conf., Tampere, Finland, 2011, pp. 9–15.
[4] H. Ebbinghaus, Über das Gedächtnis. Leipzig: Duncker & Humblot, 1885.
[5] P. A. Wozniak and E. J. Gorzelany, "Effect of active recall on the learning of foreign language vocabulary," Computers and Education, vol. 23, no. 1–2, pp. 107–113, 1994.
[6] G. Kurdi, J. Leo, B. Parsia, U. Sattler, and S. Al-Emari, "A systematic review of automatic question generation for educational purposes," International Journal of Artificial Intelligence in Education, vol. 30, no. 1, pp. 121–204, 2020.
[7] A. S. Lan, A. E. Waters, C. Studer, and R. G. Baraniuk, "Sparse factor analysis for learning and content analytics," Journal of Machine Learning Research, vol. 15, pp. 1959–2008, 2014.
[8] T. Brown et al., "Language models are few-shot learners," in Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 1877–1901.
[9] K. VanLehn, "The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems," Educational Psychologist, vol. 46, no. 4, pp. 197–221, 2011.