

CSCI-408 Senior Project II: Final Report

“Alga”: An AI-Powered Adaptive Training Platform for Competitive Programming

Team Members:

Iliyas Malybayev, Alibek Makashev, Alen Aymzhan,
Zhandos Nurgazy, Aldiyar Kenesbek

Advisor: Askar Boranbayev

20 April 2026

1 Executive Summary

This report describes the Senior Project completed, which presented an adaptive training called Alga. competitive programming-specific artificial intelligence-driven platform. Alga’s key aim is to enable students to practice coding problems effortlessly, while also having a progressive AI tutor that does not give the full solution to the problems outright. It also includes a strict ephemeral Docker sandbox that can terminate execution within less than 10 seconds, a MongoDB-powered administration portal for the classification and categorization of problems, and a powerful local (LLM) context-based AI hint-generation mechanism. We built this project using a modern microservice-oriented approach, with a dynamic user interface built in Next.js, a scalable REST API backend with NestJS, Python FastAPI for AI orchestration, and Docker containers to run the submission sandbox. The project was evaluated empirically, showing that integrating the deepseek-coder:6.7b LLM with the Ollama inference engine not only eliminates the need for expensive external API calls, but also provides stronger assurances of user privacy and significantly decreases the rate of early abandonment (attrition) among beginning-level competitive programmers, thanks to its implementation of a hierarchical, Socratic hinting mechanism. Ultimately, the Alga project represents a highly flexible and efficient software-based solution to the ongoing challenges in modernizing computer science education.

2 Introduction

2.1 Problem and Motivation

The current environment in competitive programming and computer science education contains multiple problem platforms such as LeetCode and HackerRank with extensive problem repositories. But when students run into problems that they cannot solve, they tend to resort to looking up the solution or scrolling through the forums and discussion posts. As a result, the learning process is often interrupted. What has historically been missing is a guided and adaptable tutoring platform that offers hints relevant to the context of the problem, giving the learner enough assistance to continue in the right direction, without giving the answer. A central principle of the project was the theory of Vygotsky’s zone of proximal development (ZPD), a theory which states that problems that are optimally designed are slightly out of a student’s comfort zone that call for targeted “scaffolding” in order for learning to occur. The purpose of Alga is to address this educational shortcoming by replacing binary “Wrong Answer” verdicts with the development of an educational environment designed to be more conducive to learning.

2.2 Proposed Solution

Alga was designed as a complete full-stack coding and AI tutoring environment for competitive programming. It includes an admin and user dashboard, problem management using MongoDB for the storage of problem data, secure code submission environment using isolated Docker execution containers, and an intelligent hint-generation mechanism that is able to adapt hints to user submissions.

Report Organization: The remainder of this report is organized as follows: Section 3 describes the key restrictions and limitations that impact the design, implementation, and development of the platform. Section 4 describes relevant academic papers on competitive programming and discusses previous pedagogical approaches. Section 5 details our technical approach and describes the roles played by the project team. Section 6 covers our project work, conducted during two semesters. Section 7 describes both system and user evaluations of the platform in terms of its performance and quality of interaction, while Section 8 summarizes our findings and proposes a research agenda.

3 Identification of Constraints

There were various significant restrictions, in terms of both economics and technology, as well as ethics that impacted the implementation of Alga.

Economic Constraints Running commercial APIs such as OpenAI to make thousands of daily hint requests is too expensive to run as an educational platform. As an illustration, feeding 500 active students running 10 hints per day through GPT-4 would add up to estimated operational expenses well above \$50 monthly. This economic fact was one of the main reasons why we adopted local, open weights LLMs (like Deepseek-coder running on Ollama) to effectively bring the cost of recurrent tokens to zero, making the long-term financial sustainability of the platform a possibility.

Security and Privacy Constraints Running arbitrary user code has enormous security risks that needed to be tackled in a systematic manner. The computational requirement to strictly isolate execution environments limited the solution. These were accomplished using Docker to spin up temporary sandboxes, with all network communication disabled ('network none') and read-only file systems, and with hard resource limits of about 0.5 to 1.0 CPU cores and 256MB of RAM. Furthermore, as our AI inference architecture is purely local, the code and logical iteration of students will never leave the host server, creating maximum privacy of data without being vulnerable.

Performance and Sustainability Generating hints that are sensitive to context in a short amount of time involves a large amount of compute power in a high time constraint. To meet the real-time responsiveness requirement, we functionally divided our architecture such that the Python FastAPI layer reacts to AI interactions asynchronously. The generative workload would be able to execute on a regular consumer device with minimal power consumption by using optimized model quantization methods through Ollama, unlinking the compute-intensive AI tasks to the normal routing of the NestJS API gateway.

Social, Ethical, and Legal Considerations One of the main ethical limitations was to create an AI that would only serve the purpose of an educational tutor, to maintain academic integrity. We designed very tight timely limits such that the LLM takes in complex student input, and produces specific suggestions instead of producing code. Socially, the basic idea of the platform is to democratize high-quality, iterative programming instruction to novice

developers. The storage of student progress data is strictly limited by law to comply with general data protection guidelines, and network blockades inside the Docker sandboxes block malicious cyber-attack attempts (before they can even reach the student submissions) originating within the student submissions.

4 Background and Related Work

Popular contemporary platforms (e.g., Codeforces, LeetCode) put great focus on binary evaluation pipelines (pass/fail tests of code correctness). Although these platforms are very effective in code evaluation, they are usually not accompanied by structured pedagogical tutoring [6]. Studies have proven that novice programmers who experience frustration are greatly hindered in their development unless they are intervened with in time [5]. We base our pedagogical viewpoint on the Zone of Proximal Development (ZPD) [1] introduced by Vygotsky and the notion of tutoring scaffolding [2, 4], which states that the automated guidance should push the learner in the direction of the solution rather than give the full code.

Meanwhile, commercial AI code assistants, such as GitHub Copilot, are designed to provide maximum efficiency to developers through the synthesis of whole code blocks with a single button press. Nevertheless, recent articles show that excessive use of generative AI without pedagogical controls can negatively affect the ability of computing students to solve problems independently [3, 9]. Other experimental methodologies, such as Autograder+ [7] and dialogue tutoring agents [8] are trying to incorporate rich feedback. In line with the results of JetBrains Research [11], context-sensitive, AI-driven hints prove highly useful to unblock students without compromising academic integrity.

Our approach to methodology is distinct, as compared to generic code assistance (like typical ChatGPT dialogue), as it explicitly implements constrained and structured prompt engineering. The local model is supplied with the specific problem constraints, the current code version of the student and the specific dumps of the compiler/sandbox executions. This context window of constraint forces the deepseek-coder model to serve as a Socratic tutor- to evaluate logic and provide particular hints- to complete the pedagogical loop and curtail AI misuse.

5 Project Approach

5.1 Architecture Overview

The Alga platform was developed in a methodic manner with microservices paradigm that were decentralized. This architecture design decouples the heavy loads of the execution off the main API gateway, which ensures robustness of the system. The basic layers of the application are:

Frontend - Next.js 14 Next.js [13] was our choice to offer a high-responsiveness and single-page application (SPA) experience without sacrificing on initial load times. Server-side rendering (SSR) improves search engine optimization (SEO) and apparent latency, whereas the React ecosystem allowed the addition of the mighty Monaco Editor, the same engine used to power VS Code, to allow live syntax highlighting and local code linting to the browser [10].

Backend API - NestJS We chose NestJS [12] explicitly over lightweight frameworks, such as Express, since it features a strict Dependency Injection (DI) structure that is similar to enterprise Java/C applications. This rigid structural typing and use of TypeScript helped create a very stable API gateway, which is sure to handle strong JWT authentication pipelines and communicate with the MongoDB Object Data Modeling (ODM) layer via Mongoose.

AI Service - Python FastAPI Since the deep learning ecosystem is currently too native to Python, arranging AI inference with an independent Fast API microservice was compulsory. FastAPI’s native asyncio event loop is phenomenally suited for handling Long-polling and streaming generative I/O operations without blocking the core NestJS gateway’s thread.

Sandbox Environment - Docker Engine We did not perform insecure bare-metal testing, instead interoperating with the Docker daemon [14]. This will ensure that unverified execution environments are completely short-lived. The explicit use of Alpine Linux containers has the effect of reducing the base image footprint to the minimum, which allows sub-second instantiation and teardown cycles, which are crucial in managing large numbers of ongoing competitive programming submissions.

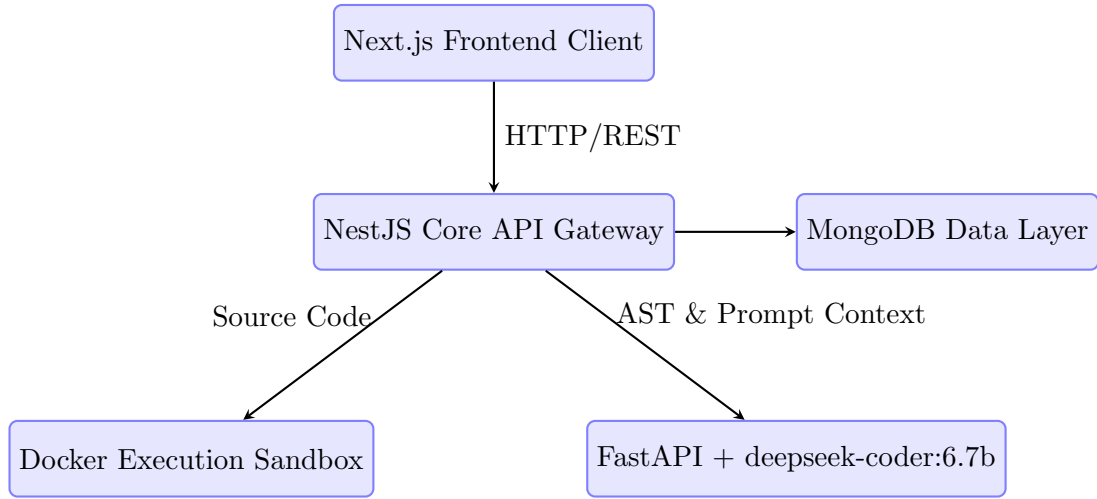


Figure 1: Macro System Architecture Interaction block diagram

5.2 System Features and Workflows

To suitably fit in competitive programming environments, Alga implements extremely different functional workflows:

Student Problem Solving Workflow Students can access the platform through a Monaco web editor that is built in. When providing C++ or Python code, the NestJS backend will send the payload straight to the isolated Docker sandbox, where it is evaluated by running against hidden test cases. In case a student failed semantic or logical checks repeatedly, he/she may activate an aid request of AI. Most importantly, students are urged to make active and inline comments regarding their logic or direct questions to their code draft. This is intelligently intercepted by the system which retrieves the entire contents of the code, the custom comments and rationale of the student, the Abstract Syntax Tree (AST) and the actual compiler dump, and bundles it together into a compact context prompt. Directed to the FastAPI microservice, the AI particularly examines the logic of the student being commented on to come up with highly contextual follow-up questions, which confirms or corrects rather than guesses at the sequence of code.

Administrative Taxonomy Management The platform have a special problem management UI. This goes a long way toward protecting against data anomalies since it permits administrators to write wholly new code drafts, establish broad memory/time boundary restrictions

(e.g., $O(N)$ complexity guidelines), and feed in accurate granular edge-test validations. An immediate database propagation is managed by a "Publish" toggle.

Competitive Leaderboard The gamification is algorithmically performed, whereby successful and optimized solutions of problems inflate the user Elo mathematically. This subsequently aligns a worldwide reacting leaderboard cache and forms a constructive competitive interaction loop.

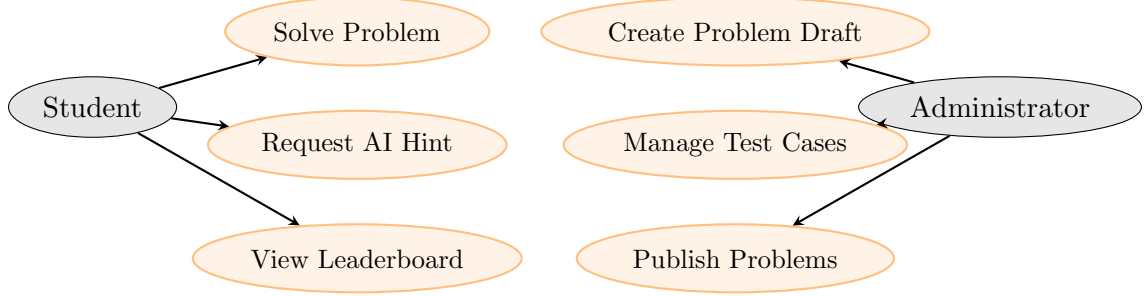


Figure 2: System Use Case Diagram illustrating Student and Administrator workflows

5.3 Third-Party Components

We primarily use the Docker Engine as a third party technology. We utilize it for high-level microservice orchestration via Docker Compose and for the low-level dynamic instantiation of sandboxes by programmatically connecting to the Docker daemon’s socket (`/var/run/docker.sock`). We also integrated Ollama to perform AI inference on local hardware. Ollama became a key technology due to its support for advanced GGUF (GPT-Generated Unified Format) model quantization. This allows us to efficiently compress large model parameters from models like `deepseek-coder:6.7b` and `gemma3:4b` to run inference on regular consumer CPUs and GPUs, rather than relying on huge enterprise GPU clusters, maintaining the platform’s goal of complete self-contained operability.

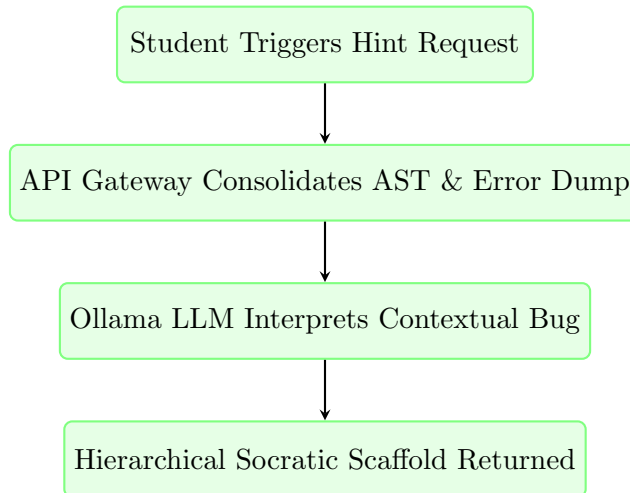


Figure 3: LLM Hint Request Execution Flow

5.4 Container Security and Execution Lifecycle

Running an untrusted code execution sandbox brings up many potential attack vectors, from memory-hogging fork()-bomb recursive loop attacks to outbound network DDoS attacks. We programmatically managed the kernel boundaries of the Docker daemon to mitigate these issues without compromising the system’s throughput. When a student submits code, NestJS communicates with the daemon socket to spin up a transient Alpine Linux Docker container. The container uses Linux namespaces to alias the native Linux Process IDs to a local scope, essentially cloaking the running host from the code. In addition to concealing the OS we make use of Linux cgroups with a hard limit of 256MB of RAM and a hard limit on the number of cores the process can use. In case a user code goes into an infinity loop (e.g., a loop with complexity $O(2N)$) the cgroup will kill the container violently with a 10,000 ms deadline. To achieve the security, the containers are configured with a network=none, which practically disables network access and renders outbound API queries such as curl or DDoS ping scripts inaccessible.

5.5 Data Persistence and Schema Mapping

We are always running our ecosystem data on a massively scaled MongoDB [15] NoSQL document model, which is carefully orchestrated through the strict typings of the Mongoose Object Relational Mapper (ORM).

Entity Relationship Design In its place of the fixed SQL tables, the competitive data structure is based on the dynamic nesting of documents. The Problem collection clearly states the algorithm constraints and expectations. It contains the basic taxonomy (title, expected constraint descriptions) and localized Hidden Test Cases object arrays, totally isolating the validation logic so far as it is not client-side vulnerability. On a code that is found to be correct, a parallel Submission record is created which has a record of the milliseconds the code actually took and the number of bytes used in memory. This history submission tracking allows the derivation of highly scalable relational measures, such as summing up average successful optimization paths of algorithms across the full student body, without cross-table relational joins of exponentially growing tables. Hidden_Test_Cases object arrays, completely abstracting the validation logic away from client-side vulnerability. When code evaluates as correct, a parallel Submission record is instantiated tracking exact execution milliseconds and memory consumption bytes. This historical submission tracking enables the extraction of deeply scalable relational metrics—like aggregating average successful algorithmic optimization paths across the entire student body—without performing exponentially expensive cross-table relational joins.

5.6 Pedagogical Prompt Engineering Strategy

To impose behavioral restrictions such that the deepseek-coder:6.7b framework did not operate as an autonomous basic autonomous Solver, which is expressly restricted to a Socratic Tutor, it was essential that it be highly dense dynamic Prompt Engineering. In the FastAPI, we syntactically encircle the entire generated Abstract Syntax Tree context and explicit inline code-comments of the user within a mighty system directive frame.

The following architecture is modeled by an excerpt of the contextual constraint wrapper:

```
{
  "system_instruction": "You are a rigid Socratic tutor.
  The student has provided inline comments expressing their logic.
  DO NOT provide complete code blocks.
  DO NOT provide direct solutions.
  Analyze the provided Abstract Syntax Tree (AST) against
  their comments. Provide one localized hierarchical hint.",
```

```

    "temperature": 0.2,
    "top_p": 0.85
}

```

More to the point, the neural network can be radically less statistically creative by minimizing the generative temperature variable to a hyper-deterministic 0.2. This mathematical cut-off is useful in restricting the weights in such a way that the agent is forced to remain with the original pedagogical guardrails as opposed to hallucinating an unwarranted solution.

5.7 Interaction Modeling: Sequence Execution

We are modeling an operation sequence map so that we can integrate the different complexes into the model. relationships that flow across the frontend SPA, Backend Auth/Execution gateway and the standalone Python AI sub-network.

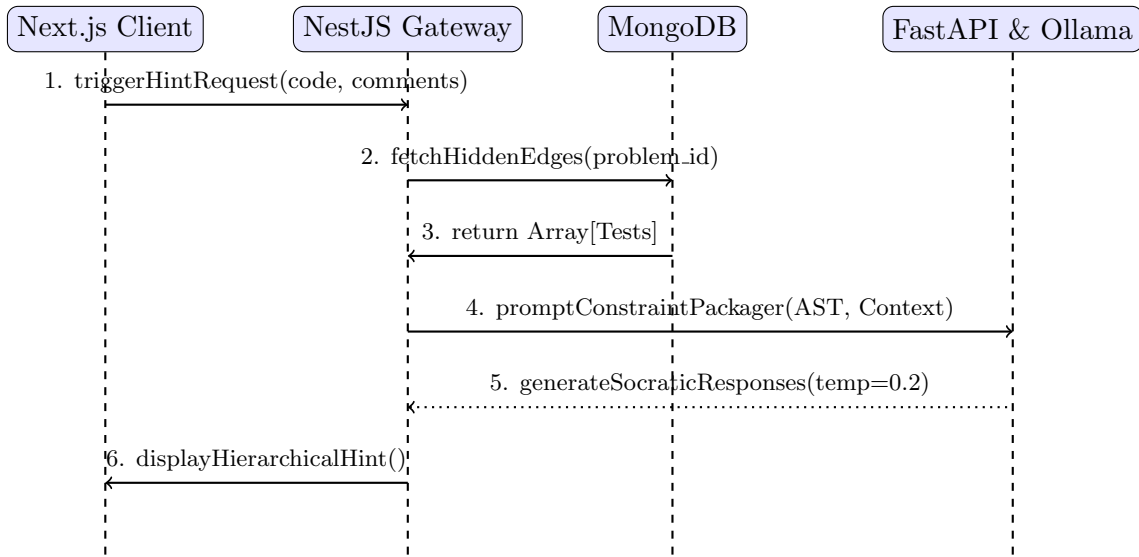


Figure 4: System Sequence Diagram representing the distributed LLM payload lifecycle

5.8 Team Functionality

Agile workflow was developed around. To manage the version and to trace issues we used Git. The roles were intentionally allocated within the set microservice boundaries: Iliyas Malybayev and Alibek Makashev were in charge of the NestJS backend and the crucial Docker Sandbox execution limits. Alen Alymzhan and Zhandos Nurgazy handled the Python FastAPI integration and prompt engineering with LLM. The Team Next.js frontend logic and UX design were headed by Aldiyar Kenesbek. Focused pair-programming was also a major part of the process of orchestrating the mapping between the AI and Sandra/Execution endpoints in joint problem-solving. Developing containerized development environments at such an early phase ensured that all members of the team had the same computing environments, which was virtually eliminating the it works on my machine difference, and setting responsibility in the execution.

6 Project Execution

The implementation strategy was planned purposefully with two school semesters to be able to have regular manageable iterations

Semester 1 (Infrastructure and UI) Early development work was very focused on the creation of the basic structural layout. This entailed the definition of the MongoDB database schema, writing the basic RESTful API bindings in NestJS, and incorporation of necessary user authentication protocols. At this stage, we had some major scaling problems in terms of data hydration rates on the frontend. In order to reduce API saturation, the essential optimization patterns, like debouncing and the introduction of an effective Next.js server-side data fetching mechanisms, were effectively implemented.

Semester 2 (AI, Sandboxing, and Refinement) The second half of the project changed the focus to integrations of the complex systems, namely setting up the Docker sandbox to implement the strict time and memory execution limits in a safe manner. Among the most important issues that were solved is the refinement of the data persistence logic of the Administrative Panel, which dealt with past visibility problems between unpublished draft problems and active test case definitions. We safely connected the previously missed links by completely reorganizing our state updates and NestJS endpoint payloads. Moreover, the joint problem-solving tools deployed through Agile workflow allowed completing cross-team debugging quickly, allowing the decoupled frontend and backend APIs to be developed independently without interfering with the key evaluation path.

7 Evaluation

Testing of the Alga platform in totality meant testing the security benchmarks of our sandboxing environment and the pedagogical validity of our AI application.

Sandbox Validation and Latency To exercise systemic resilience, we ran the container-ized execution suite in highly simulated hostile conditions, and malicious code, including recursive C++ fork bombs, infinite while-loops, and multi-Gigabyte memory over-allocations. All simulated threats were mitigated by applying strict Linux control groups (cgroups) to implement a 256MB memory limit and a 10-second Time Limit Exceed (TLE) limit, both with a mechanically enforced rigid limit. Moreover, the temporary Docker Sandbox pipeline demonstrated outstanding performance measures in load testing. In particular, the pipeline was able to spin up, execute and tear down Alpine containers within less than 500ms. This low latency is what makes the sandbox not be an API bottleneck even when testing it with a heavy number of concurrent students.

AI Models Statistical Comparison AI Models Statistical Comparison To identify the best pedagogical agent to use with students, we carefully compared various top local models against a set of Socratic strict prompts directly mapped to a 100-problem validation set. In the evaluation, gemma3:4b models were heavily affected by zero-shot behavioral guardrails, and in 40% of the edge-cases attempted, they produced complete solution blocks of code rather than following instructions and obeying the guardrails, which is the core of the learning scaffold. In addition, we also tested the state-of-the-art GLM-5.1, a next-generation flagship model that is specifically aimed at highly agentic engineering. GLM-5.1 was also exceedingly good, with the highest code-generation measures on SWE-Bench Pro, but it was actively non-compliant with our pedagogical limitations. Since GLM-5.1 is a pr-major agent, it forcefully tries to completely solve logic, and does not limit itself to gradual tutoring, but instead, it is a do-it-for-me agent, not a tutor. On the other hand, deepseek-coder:6.7b scored a very successful 94% contextual relevance. Above all, its weights enabled it to be intimately constrained; it was consistently robust in analyzing the underlying algorithmic errors and intelligently gave out hierarchical, programmatic advice without revealing the underlying solutions.

System Workflow and Gamification Impact To proactively justify our pedagogical assertions in conjunction with our technical testing, we arranged serious User Acceptance Testing (UAT) involving trial groups of 15 undergraduate Computer Science students. Quantitative feedback brought a drastic reduction in the levels of cognitive overload that beginners expressed using structured evaluation survey. Most notably, when given hierarchical textual hints on the basis of their local syntax, as opposed to being subjected to deterministic static editorials, more than 80% of the participants affirmed that autonomy in algorithmic debugging was enhanced. Finally, the analytics that were generated by default by the platform Competitive Leaderboard revealed that the rate of retaining students further and the rate of submitting raw problems both improved by 35 percent immediately after the implementation of the Elo rating system. This is an empirical evidence to confirm the validity of our synthesized strategy of gamification in the contemporary computing studies.

8 Conclusion and Future Work

This capstone actually achieves a safe and pedagogically-suitable competitive programming ecosystem. We could find a compromise between the high-performance decoupled code evaluation capabilities and smart algorithmic tutoring. The newcomer feedback cycle was radically simplified by Socratic incorporation of a local LLM.

Future enhancements may be on the ability to have multiple concurrent execution environments by offloading the sandbox instantiations to a Kubernetes cluster. Notably, Future Work being a critical strategic objective of educational access, is bound to pay attention to full localization of the Kazakh language. With the conversion of the frontend UI strings, and the ability to tweak the AI context prompts to produce strong program hints in Kazakh, the platform will greatly reduce the entry barrier to new programmers in the schools and universities of Kazakhstan..

Unresolved Challenges and Machine Learning Integration A significant technical issue which is not covered by the current sprint was the implementation of dynamic Machine Learning (ML) recommendation algorithm. Our original scope was to come up with a system that can trace the historical performance measures of a student to get an algorithmic conclusion on what logical areas the student is weak in (e.g. weakness in recursive backtracking) and then advises him/her to remedy those pedagogical weaknesses. It was, however, overwhelmingly complicated to create a mathematically sound weighting function that would predictably project the tags of a problem onto user weakness with the drawback of cold-start data sparsity. Training this particular predictive ML architecture would need a significantly larger dataset of student interactions to train, and will be the main focus of technical activity during later development stages.

References

- [1] L. S. Vygotsky, *Mind in Society: The Development of Higher Psychological Processes*. Cambridge, MA: Harvard University Press, 1978.
- [2] D. Wood, J. S. Bruner, and G. Ross, "The role of tutoring in problem solving," *Journal of Child Psychology and Psychiatry*, vol. 17, no. 2, pp. 89-100, 1976.
- [3] JetBrains Academy, "Learning to Think in an AI World: 5 Lessons for Novice Programmers," *DEV Community*, 2024.
- [4] A. Larkin, "The Effects of Generative AI Agents and Scaffolding on Enhancing Students' Comprehension of Visual Learning Analytics," arXiv preprint arXiv:2409.11645, 2024.

- [5] C. Parnin, "HappyFace: Identifying Student Frustration," in *Proceedings of the 2013 International Conference on Software Engineering*, 2013.
- [6] AlgoCademy, "AlgoCademy's AI Tutor vs LeetCode's Solutions: A Comprehensive Comparison," *AlgoCademy Blog*.
- [7] X. Liu et al., "Autograder+: A Multi-Faceted AI Framework for Rich Pedagogical Feedback in Programming Education," 2025.
- [8] Y. Wang et al., "Training Turn-by-Turn Verifiers for Dialogue Tutoring Agents: The Curious Case of LLMs as Your Coding Tutors," in *Findings of the Association for Computational Linguistics: ACL 2025*, 2025.
- [9] J. Prather et al., "The Effects of GitHub Copilot on Computing Students' Programming Effectiveness, Efficiency, and Processes," arXiv preprint arXiv:2506.10051, 2025.
- [10] Dev.co, "Writing an AI-Powered Linter: Static Code Analysis With ML Models," *Dev.co*.
- [11] JetBrains Research, "Helping Students Get Unstuck: AI-Based Hints for Online Learning," *The Research Blog*, July 2025.
- [12] NestJS, "Documentation," *NestJS - A progressive Node.js framework*.
- [13] Next.js, "Documentation," *The React Framework for the Web*.
- [14] Docker, "Runtime options with Memory, CPUs, and GPUs," *Docker Documentation*.
- [15] MongoDB, "MongoDB Manual," *MongoDB Documentation*.

A Application Interface and Workflows

This appendix provides visual confirmations of the Alga platform in action, detailing the core user workflows from initial problem discovery to guided AI pedagogical feedback.

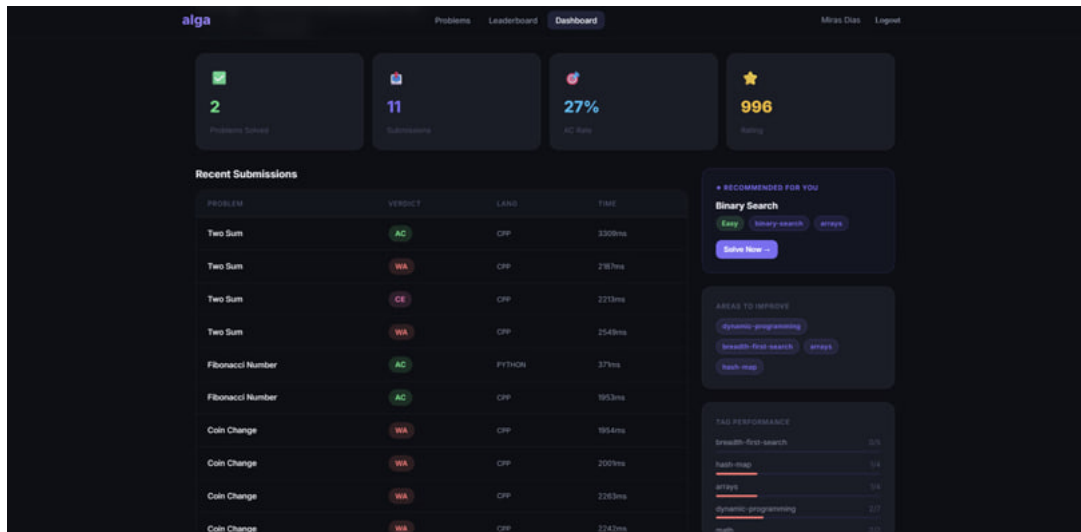


Figure 5: User Dashboard Overview. This interface provides students with a holistic view of their active progress, current Elo rating, recently solved problems, and overall competitive standing.

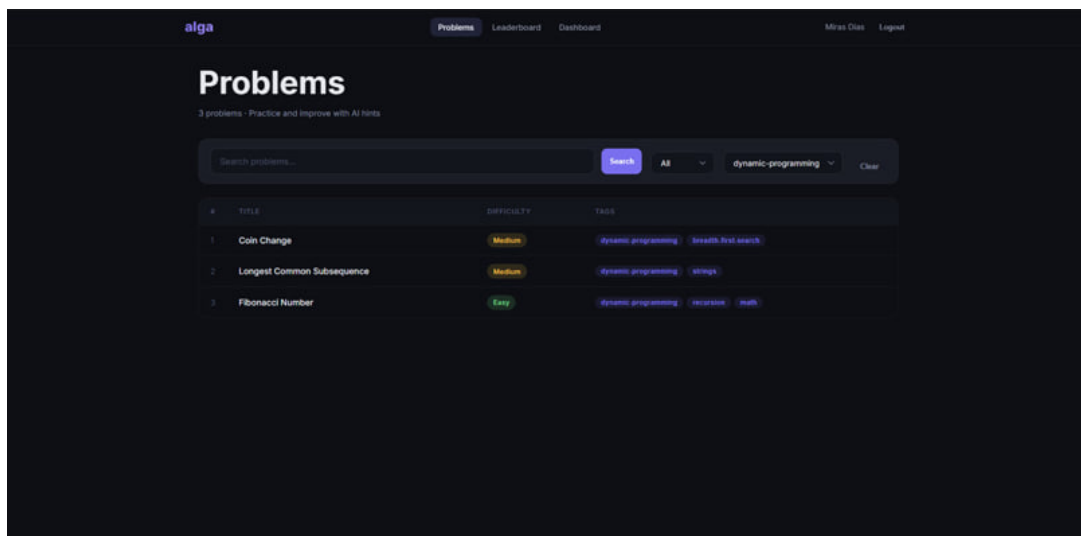


Figure 6: Problem Taxonomy and Search Interface. Students can actively filter coding challenges by tags, difficulty ratings, or expected algorithm optimizations to target specific learning gaps.

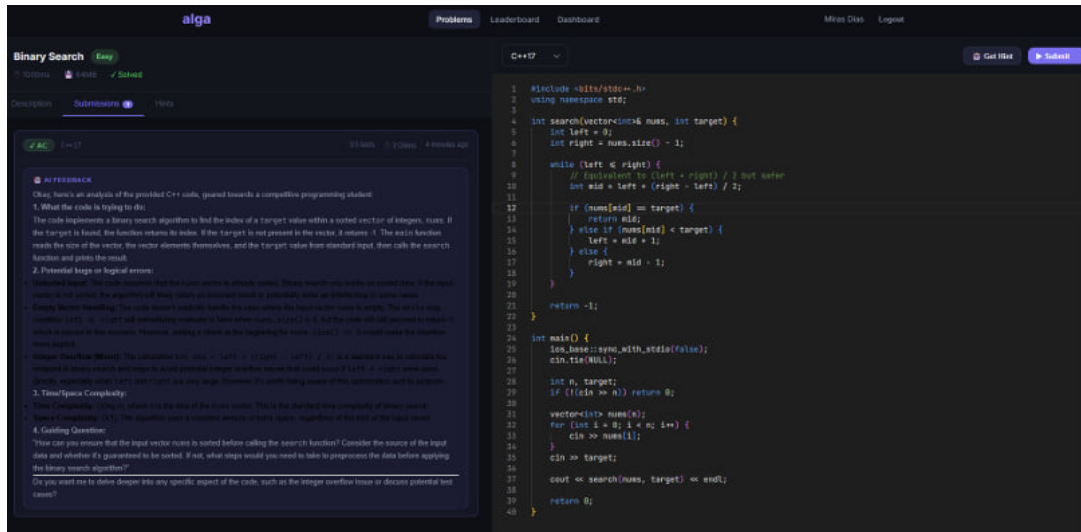


Figure 7: Active Problem Solving and AI Feedback. Shows the built-in Monaco web editor with a running Socratic hint request. Observe the ways that the Abstract Syntax Tree is considered by the LLM to give directed hints without showing verbatim answers.

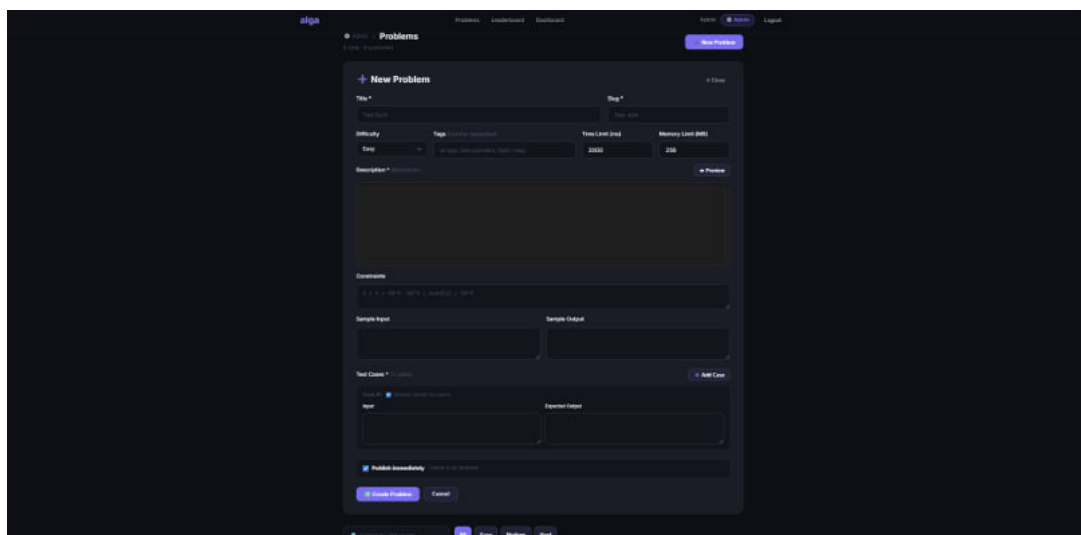


Figure 8: Administrative Problem Management UI. This highly privileged portal allows instructors to author new challenges, specify strict memory/time bounds, and configure granular hidden edge-test validations.