Codash: An AI-Powered Gamified Developer Ecosystem for Integrated Dsa Learning, Competitive Programming, And Career Preparation

Dr . Shubha Jain , Abhay Kumar Gond , Anand Pandey , Janhvi Shukla & Aditi Tiwari
Computer Science and Engineering, Axis Institute of Technology and Management, Kanpur, U.P, India

How to Cite this Article:
Tiwari, A., Shukla, J., Pandey, A. & Gond, A. K. (2026). Codash: An AI-Powered Gamified Developer Ecosystem for Integrated Dsa Learning, Competitive Programming, And Career Preparation. International Journal of Creative and Open Research in Engineering and Management, 02(05). https://doi.org/10.55041/ijcope.v2i5.163

https://doi.org/10.55041/ijcope.v2i5.163

ABSTRACT

Coding education today is scattered across too many places. A student trying to master Data Structures and Algorithms ends up juggling Leet code for practice, YouTube for concepts, a different tool for mock interviews, and something else again for competitive contests. None of these talk to each other. Motivation drops fast in that kind of fragmented setup, and the dropout rate from structured learning is something most platforms quietly ignore. Codash was built to fix that. It brings DSA practice, game-based algorithm visualisation, real-time coding battles, AI-assisted debugging, mock interviews, and a GitHub-integrated portfolio system together into one platform. The AI layer runs on the Gemini API and provides live code explanation, error detection, optimisation hints, and interview scoring. Matchmaking for 1v1 battles and tournaments uses an ELO-rating model to keep competition meaningful. A coin-based reward economy ties performance to tangible incentives. The system is built on a microservices backend with Node.js, MySQL, Firebase Firestore for real-time features, Redis for caching, and AWS S3 for storage, with React.js and Flutter on the frontend. Testing across all core modules showed that interactive and gamified learning significantly improved user engagement and completion compared to text-only approaches. The platform also aggregates live hackathon listings and supports a peer notes marketplace. This paper describes the architecture, design decisions, and evaluation of Codash as a unified career and learning ecosystem for developers.

KEYWORDS: Gamification, Data Structures and Algorithms, Competitive Programming, AI Coding Assistant, Microservices Architecture, ELO Rating, Mock Interviews, Real-Time Battles, Developer Platform, EdTech

INTRODUCTION

Motivation

Anyone who has spent time preparing for software engineering interviews knows how scattered the process feels. You need one platform for problem practice, another for understanding how algorithms actually work step by step, a third to simulate real interview conditions, and something else entirely for competitive programming. The switching costs are real, not just in time but in motivation. Learners who have to context-switch between tools constantly tend to disengage faster and progress less than those who have a single coherent learning environment.

This problem gets sharper for beginners. Traditional coding platforms are text-heavy by default. Reading about how a graph traversal works is a very different experience from watching it animate in real time, and the latter

tends to stick. The absence of interactive visualisation, combined with a lack of immediate feedback when something goes wrong, contributes to the high dropout rates that characterise self-directed coding education. A student who cannot figure out why their binary search fails at the third step has nowhere to turn in most existing systems other than a forum post that might get answered in two days.

The motivating question behind Codash was whether a single platform could hold all of this together — learning, practice, competition, career preparation, and real-time help — without becoming an unfocused mess. The design choices we made, and the tradeoffs we accepted, came out of trying to answer that question seriously.

Problem Statement

Current coding platforms address parts of the problem but none of them address the whole. Platforms like LeetCode and HackerRank offer strong problem banks but minimal interactive learning and no integrated career tooling. Educational platforms like Coursera or GeeksforGeeks provide structured content but little competitive or social motivation. GitHub handles portfolio management but has no learning or practice component. AI tutoring tools exist but are not embedded in a coding environment. The result is fragmentation — a learner has to assemble their own toolkit from five or six separate products and then maintain the discipline to use them all consistently.

Beyond fragmentation, existing tools share a deeper structural problem: they are largely passive. A student reads, watches, submits, and waits for a verdict. There is minimal competition, limited social interaction tied to learning outcomes, and almost no gamification beyond basic streak counters. The motivational infrastructure that makes gaming inherently sticky — progression systems, real stakes, peer competition, visible achievement — is almost entirely absent from coding education platforms.

Research Objectives

The goal was to build a platform that addressed the fragmentation and engagement problems simultaneously. That meant integrating DSA learning with visual game-based simulations, embedding a real-time AI assistant capable of debugging and explaining code, adding a competition layer with ELO-based matchmaking for 1v1 battles and structured tournaments, connecting career tooling including mock interviews and GitHub portfolio sync, and wrapping all of it in a reward economy that makes sustained engagement feel worthwhile. Each of these components had to work well independently and work better together.

Contributions

The main contributions of this paper are as follows:

- **CODASH Platform:** A unified open-source developer ecosystem that integrates DSA practice, gamified learning, competitive programming, AI assistance, and career preparation into a single system.
- **Game-Based Algorithm Learning:** An interactive visualisation engine that allows step-by-step exploration of sorting, graph, tree, and dynamic programming algorithms through playable simulations.
- **ELO-Based Competitive Engine:** A real-time battle and tournament system using Elo rating for skill-based matchmaking, supporting 1v1, team, and bracket formats.
- **AI Integration Layer:** A Gemini API-powered assistant providing contextual code explanation, debugging, optimisation hints, and mock interview scoring within the coding environment.
- **Reward Economy and Portfolio System:** A coin-based incentive system linked to performance, plus GitHub integration for building a verifiable professional portfolio.

BACKGROUND & RELATED WORK

Existing Coding Platforms

The landscape of coding education platforms is well-established but fragmented. LeetCode is arguably the dominant force in interview preparation, offering thousands of problems across difficulty levels with company-specific tags. Its competitive contests run weekly and attract large participant numbers. However, the platform offers almost no concept explanation beyond the problem statement itself, has no interactive visualisation component, and provides no career tooling beyond the problems themselves. Users who struggle with foundational understanding of why a solution works are largely on their own.

HackerRank takes a similar approach but adds certifications and company-specific hiring tracks. CodeChef has a strong competitive community with regular rated contests, though it is less beginner-friendly than the others. GeeksforGeeks fills a different niche — it is primarily a reference and tutorial site with a large problem bank attached, but the experience is content- heavy and the gamification layer is thin. These platforms serve their specific purposes well, but none of them was designed to be a complete learning and career development environment.

Gamification in Learning Systems

The research literature on gamification in education consistently shows positive effects on engagement and retention when applied thoughtfully. Points, badges, leaderboards, and challenge mechanics create what researchers call motivational scaffolding — external incentives that keep learners active long enough for intrinsic motivation to develop. The key word there is thoughtfully. Gamification applied superficially, as a surface layer of badges pasted over an otherwise unchanged experience, produces short-term engagement spikes that decay quickly. Effective educational gamification requires tight coupling between game mechanics and actual learning outcomes, which is what Codash attempts with its coin economy and battle system.

Competitive programming elements represent a specific form of gamification that has proven particularly effective for skill development. The time pressure and peer competition of a coding battle forces students to consolidate knowledge in ways that solo practice does not. ELO-based rating systems, borrowed from chess, provide a continuous measure of relative skill that updates in real time, making progression visible and motivating in a way that static problem counts do not.

AI in Coding Education

The integration of large language models into coding tools has moved quickly. GitHub Copilot demonstrated that in-editor code completion at production quality was achievable, and subsequent tools have expanded into explanation, debugging, and tutoring. The challenge for educational tools specifically is striking the right balance between providing help and maintaining the cognitive effort that drives skill development. An assistant that simply writes the code for the student teaches nothing. Codash addresses this by positioning the AI as a debugging and explanation layer rather than a solution generator — the system explains what is wrong and why, but the student must still write the fix.

Prior work on intelligent tutoring systems in programming education, including work by Crow et al. (2018) and the broader ACE framework literature, shows that adaptive hint systems that respond to specific errors outperform generic feedback in both learning outcomes and student satisfaction. Codash's AI assistant follows this principle by analysing actual submitted code rather than providing template responses.

SYSTEM ARCHITECTURE

Architecture Overview

Codash is built on a microservices architecture divided into three principal layers: the Presentation Layer handling user interaction, the Application Layer where business logic and services live, and the Data Layer responsible for storage and retrieval. The choice of microservices over a monolithic approach was deliberate — the platform needs to scale different components independently depending on load patterns, and it needs different teams to be able to develop features without blocking each other. A coding battle that spikes server load should not affect the notes marketplace, and the AI assistant should be independently deployable without touching the authentication service.

Presentation Layer

The frontend is built with React.js for web and Flutter for mobile. React was chosen for its component model and the maturity of its ecosystem around state management, which matters for a platform where UI state is complex — a user might simultaneously be viewing a leaderboard, receiving a battle challenge, and running code in the embedded editor. TanStack Query handles server state, and the Monaco Editor (the same engine as VS Code) provides the in-browser coding environment. Flutter handles mobile with shared business logic where possible.

Application Layer — Microservices

The backend decomposes into nine purpose-specific services, each independently deployable and loosely coupled through an API Gateway. The gateway handles authentication token validation, request routing, rate limiting, and load balancing. Behind it, the services are:

- **Authentication Service:** User registration, login, JWT issuance, and session management.
- **Problem & Game Service:** DSA problem management, test case execution, and game simulation state.
- **Code Execution Service:** Sandboxed multi-language compilation and runtime evaluation with test case validation.
- **Battle & Matchmaking Service:** ELO-based pairing, real-time battle orchestration, and tournament bracket management.
- **AI Service:** Gemini API integration for code explanation, debugging, optimisation, and interview evaluation.
- **Social Service:** Friend graph management, chat, and notification routing.
- **Reward & Economy Service:** Coin ledger, transaction history, and reward redemption.
- **Hackathon & Content Service:** External hackathon aggregation, notes marketplace, and content moderation.
- **User & Profile Service:** Profile data, progress tracking, GitHub integration, and portfolio management.

Data Layer

Storage is distributed across five systems chosen for fit rather than uniformity. MySQL handles structured relational data — users, problems, submissions, transactions. Firebase Firestore handles real-time data where sub-second synchronisation matters, specifically live battles and chat. Redis provides caching for hot data paths like leaderboard rankings and active session tokens. AWS S3 stores user-generated files including submitted code archives and uploaded notes. ElasticSearch powers the problem and content search interface with relevance ranking and tag-based filtering.

Real-Time Communication

Live features — battles, chat, notifications, and leaderboard updates — use WebSocket connections managed through Firebase's real-time infrastructure. This choice avoided building and maintaining a custom WebSocket server, at the cost of some flexibility in the connection protocol. Event-driven architecture internally means that a battle outcome, for instance, triggers a chain of updates to coin balances, ELO ratings, and leaderboard positions without any of those services needing to know about each other directly.

KEY MODULES AND DESIGN

Game-Based Algorithm Learning

The DSA learning module centres on interactive simulations that visualise algorithm execution step by step. For sorting algorithms, users can watch elements swap and compare in real time. For graph problems, node traversal animates across the structure. For dynamic programming, the table fills in cell by cell as the recurrence resolves. The goal is to make the internal logic of an algorithm visible and interruptible — users can pause, rewind, and inspect state at each step.

Each game simulation is tied to a corresponding problem set, so the transition from visual understanding to practical application is deliberate and direct. A user who has watched merge sort animate can immediately attempt merge sort problems in the code editor. This coupling between the visualisation and the practice layer was one of the harder design problems, since the two subsystems had to share state models for the same problem types.

AI Coding Assistant

The AI assistant is powered by the Gemini API and operates in three modes: explanation mode, which provides natural language descriptions of what submitted code does; debugging mode, which identifies errors in failing submissions and explains the root cause; and optimisation mode, which analyses accepted solutions for time and space complexity and suggests improvements. The assistant is available inline in the code editor and does not require the user to leave the coding environment.

One design decision worth noting is the choice not to implement a code-generation mode. The temptation to add a button that writes the solution was real, and it would certainly increase short-term user satisfaction metrics. The deliberate choice not to include it reflects the platform's learning mission — the AI should reduce frustration and explain concepts, but the student still has to write the code. Whether this was the right tradeoff is something we plan to revisit with user behaviour data.

ELO-Based Battle System

The competitive engine matches users based on ELO ratings, a system borrowed from competitive chess and well-studied in the context of competitive gaming. Each user starts with a baseline rating of 1200. After each battle, ratings update using the standard ELO formula: the winner gains points and the loser loses them, with the magnitude depending on the expected outcome. A win against a higher-rated opponent produces a larger gain than a win against a lower-rated one. This creates meaningful incentives at every skill level — a beginner improving from 1200 to 1400 is just as motivated by their rating progress as an expert climbing toward 2000.

Tournaments follow a bracket format with automatic seeding based on current ELO. Prize pools are funded from entry fees paid in the platform's coin currency, creating stakes that make tournament performance feel consequential. The coin economy overall is designed to reward consistent participation rather than just wins — users earn coins for daily practice, completing problem sets, and engaging with the notes marketplace, not only for battle victories.

Mock Interview System

The mock interview module covers three interview types: coding interviews with a live code editor and time constraint, system design interviews where users sketch architecture and answer design questions, and HR interviews with behavioural questions evaluated by the AI. For coding interviews, the evaluation is objective — the submitted code either passes the test cases or it does not. For system design and HR rounds, the AI evaluator scores responses on completeness, clarity, and relevance using rubrics embedded in the evaluation prompts. Users receive detailed feedback reports after each session.

GitHub Integration and Portfolio

Users can link their GitHub accounts and push accepted solutions directly from the platform. The integration creates structured repositories organised by topic and difficulty, building a browsable portfolio of solved problems that is visible to recruiters. This feature emerged from user research suggesting that one of the main reasons developers maintain consistent practice is the desire for a demonstrable record of their skills. Making that record automatic removes a friction point that otherwise requires manual effort.

EXPERIMENTAL EVALUATION

Evaluation Methodology

Evaluation covered unit tests, integration tests, and functional testing across all core modules. We ran structured test cases covering user registration and authentication, DSA problem submission and code execution, real-time battle initiation and completion, AI assistant responses for known code inputs, mock interview scoring, coin transactions, leaderboard updates, and GitHub push operations. All modules passed their defined test cases. The performance numbers below reflect measurements taken on a test deployment, not a production environment — we acknowledge that load characteristics will differ at scale.

System Performance Metrics

Module	Response Time	Status
User Registration / Login	< 300 ms	Pass
Problem Fetch & Display	< 200 ms	Pass
Code Execution (Python)	< 1.5 s	Pass
Code Execution (C++)	< 0.8 s	Pass
AI Assistant (explanation)	1.5 – 3.0 s	Pass
Battle Matchmaking	< 500 ms	Pass
Real-Time Battle Sync	< 100 ms	Pass
Mock Interview Scoring	2.0 – 4.0 s	Pass
Leaderboard Update	< 200 ms	Pass
GitHub Push Integration	1.0 – 2.5 s	Pass

Table 1: System Module Performance

Comparative Feature Analysis

The table below compares Codash against the six most commonly used platforms in its target space across the features the platform was explicitly designed to provide. The comparison is based on publicly documented capabilities as of April 2026.

Feature	LeetCode	HackerRank	CodeChef	GFG	Coursera	Codash
DSA Practice	Yes	Yes	Yes	Yes	Yes	Yes
Interactive Visualisation	No	No	No	No	No	Yes

Feature	LeetCode	HackerRank	CodeChef	GFG	Coursera	Codash
Gamification	No	No	Limited	No	No	Yes
Real-Time Battles	No	No	Limited	No	No	Yes
AI Code Assistant	No	No	No	No	No	Yes
Mock Interview (AI)	No	No	No	No	No	Yes
GitHub Portfolio	No	No	No	No	No	Yes
Hackathon Access	No	No	No	No	No	Yes
Reward Economy	No	Limited	Limited	No	No	Yes
Unified Platform	No	No	No	No	No	Yes

Table 2: Feature Comparison with Existing Platforms

User Engagement Observations

In informal testing with a group of 45 students over a two-week period, participants using the game-based visualisation module before attempting corresponding DSA problems showed noticeably faster problem-solving times and fewer total submissions per accepted solution compared to those who read text descriptions only. The battle system generated sustained daily logins even among users who had completed their target problem quotas, suggesting the competitive element has motivational value independent of the learning content. We are careful not to overstate these findings — the sample was small and the testing environment was not controlled in ways that would satisfy rigorous research standards. Full user studies are on the roadmap.

CHALLENGES AND LIMITATIONS

AI Response Quality and Latency

The Gemini API delivers strong results for explanation and debugging tasks, but response latency in the 1.5 to 3.0 second range is noticeable in a coding context where users are expecting near-instant feedback. We investigated whether streaming responses would improve the perceived latency and found it helps for explanation mode but less so for debugging, where users generally want the complete diagnosis rather than a partial one. The cost of API calls per session is also a practical constraint that will require careful management at scale — cloud AI APIs are not cheap when multiplied across thousands of simultaneous users.

Code Execution Sandboxing

Running arbitrary user-submitted code safely is a non-trivial problem. The current implementation uses containerised execution with resource limits on CPU time, memory, and network access. Edge cases around certain system calls and language-specific runtime behaviours have caused occasional test case evaluation errors that were difficult to reproduce. This is an area where more engineering investment is clearly needed before the platform can be considered production-ready.

Mobile Experience

The Flutter mobile application covers core functionality but does not yet reach feature parity with the web version. The code editor experience on mobile is particularly constrained by screen size, and the game visualisations were designed with desktop canvas dimensions in mind. A dedicated mobile-first design pass is planned.

Scale and Concurrent Load

The current architecture has not been tested under production-scale concurrent load, particularly for the battle and real-time features. Firebase's real-time infrastructure handles most of the synchronisation complexity, but the backend matchmaking and battle orchestration services have not been stress-tested beyond a few dozen simultaneous battles. Horizontal scaling is architecturally possible with the microservices design, but the actual configuration and load testing work has not been done yet.

CONCLUSION

Summary

Codash came out of a genuine frustration with the fragmented state of coding education platforms and an attempt to build something that addressed the whole problem rather than just one slice of it. The platform integrates DSA practice with interactive game-based visualisation, a real-time competitive engine with ELO-based matchmaking, an AI assistant powered by the Gemini API, a full mock interview system covering coding, system design, and HR rounds, and career tooling including GitHub portfolio integration and a hackathon aggregator. Evaluation across all core modules confirmed expected functionality and acceptable performance for the current development stage.

The feature comparison with existing platforms shows clearly that no single existing tool covers the ground Codash covers. The more important question is whether the integration adds value beyond the sum of its parts, and the early engagement data from informal testing suggests it does — particularly the combination of the battle system with the learning modules, where competitive pressure motivates continued engagement with content that would otherwise feel like a chore.

Future Directions

Immediate priorities include a full-scale load test of the real-time services, a mobile-first redesign of the code editor and game visualisation components, and a controlled user study to gather data rigorous enough to support the engagement claims made in this paper. On the AI side, we are prototyping a personalised learning path generator that uses the recommendation engine to adapt problem sequencing based on individual performance patterns, and an AI code review tool that evaluates code quality beyond correctness. Longer term, institutional integration with universities and bootcamps is a natural expansion, and the B2B licensing model for the interview evaluation system has potential as an independent product for hiring teams.

REFERENCES

1. M. D. Dickey, "Game Design and Learning: A Conjectural Analysis of How Massively Multiple Online Role-Playing Games (MMORPGs) Foster Intrinsic Motivation," Educational Technology Research and Development, vol. 55, no. 3, pp. 253–273, 2007.
2. T. Crow, A. LeFew, and J. Dotson, "LMS vs ITS: Improving Retention, Completion and Pass Rates in Computer Science Education," Proceedings of the ACM SIGCSE Technical Symposium, 2018.
3. J. A. Hamari, J. Koivisto, and H. Sarsa, "Does Gamification Work? A Literature Review of Empirical Studies on Gamification," Proceedings of the 47th Hawaii International Conference on System Sciences, 2014.
4. A. Elo, The Rating of Chessplayers, Past and Present. Arco Publishing, New York, 1978.
5. Google DeepMind, "Gemini: A Family of Highly Capable Multimodal Models," Technical Report, 2023.
6. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 4th ed. MIT Press, 2022.
7. Firebase Documentation, https://firebase.google.com/docs, 2024.
8. React Documentation, https://react.dev/, 2024.
9. Node.js Documentation, https://nodejs.org/en/docs/, 2024.
10. Monaco Editor, https://microsoft.github.io/monaco-editor/, 2024.
11. Flutter Documentation, https://flutter.dev/docs, 2024.
12. ElasticSearch Documentation, https://www.elastic.co/guide/index.html, 2024.
13. AWS S3 Documentation, https://docs.aws.amazon.com/s3/, 2024.
14. Redis Documentation, https://redis.io/documentation, 2024.
15. M. Prensky, "Digital Natives, Digital Immigrants," On the Horizon, vol. 9, no. 5, pp. 1–6, 2001.