WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

CodeGen: Automated Programming Question Generation Using AI

M. Sabeetha[1], Sujith Kumar S[2], Suriya A M[3], Surjith S[4], Surya J[5]

[1] Assistant Professor, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
[2,3,4,5] Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.

ABSTRACT :

CodeGen is an AI-based web platform that automatically prepares interview-grade programming questions for students and job seekers. It collects top interview questions from multiple websites and uses a large language model to categorize each question as Dynamic Programming, Backtracking, Greedy Algorithms, Divide and Conquer, Two Pointers, or Other; assign a difficulty; and add a company tag or General label. The system produces a detailed problem statement, examples with explanations, constraints, ten edge-case test cases, and a Python or Java solution. Each generated solution is executed against all ten test cases by an online code-execution engine. A question is accepted only when every test passes; otherwise the failing input, expected output, and error are returned to the model for correction. The React frontend, FastAPI backend, MongoDB storage, JWT authentication, and bcrypt password hashing support search, filtering, solving, results viewing, and export of only checkbox-selected questions as JSON. The supplied source describes the verification workflow and portal features but does not provide aggregate accuracy, latency, or user-study measurements.

Keywords: CodeGen; programming question generation; large language models; interview preparation; automated test-case generation; code verification; FastAPI; React; MongoDB; JWT authentication; JSON export.

I. INTRODUCTION

Preparing for technical interviews requires regular practice with algorithmic problems. Students commonly search several websites for questions on dynamic programming, backtracking, and greedy algorithms, then judge for themselves whether each item is complete, correct, and suitable for their level. This manual process is slow, and the quality of questions, test cases, and solutions varies between sources.

CodeGen is an AI-based platform designed to reduce this effort. It collects interview programming questions, groups them into algorithm categories, and presents each question with a detailed statement, examples with explanations, constraints, ten edge-case tests, and a Python or Java solution. Questions are tagged by company or General and by difficulty. Because large language models can generate incorrect tests or solutions, CodeGen verifies every generated solution by executing it against its test cases before publication. The portal also supports checkbox-based JSON export and separates conceptual questions, such as machine learning and operating-system questions, from programming questions.

II. LITERATURE SURVEY

Transformer-based large language models trained on text and code have shown strong ability to produce programs. Codex and the HumanEval benchmark measure functional correctness of generated code [1], while program-synthesis research reports improvement with model scale but continuing errors [2]. Few-shot prompting makes question and solution generation practical without training a new model [6].

Competitive Programming, Exercise Generation, and Evaluation

The APPS benchmark established a large collection of coding problems with test cases for evaluating interview-style code generation [3]. AlphaCode similarly generates and filters candidate programs using example tests [4]. Research on automatic programming exercises found that many generated exercises are sensible but not all are correct or ready to use without checking. These findings support CodeGen's execution-based verification stage.

Rigorous evaluation shows that weak test suites can accept incorrect generated code; more and harder tests expose additional failures. CodeGen therefore asks the model to consider edge cases such as empty input, single elements, duplicates, negative values, and maximum sizes before generating ten tests. The iterative correction loop follows self-debugging and refinement ideas: failed input, expected output, and error feedback are returned to the model for revision.

Existing work addresses code generation, model benchmarks, or programming exercises separately. The source paper positions CodeGen as a single portal combining multi-source collection, algorithm categorization, company and difficulty tags, edge-case test generation, execution-based solution verification, and selective JSON download.

III. PROPOSED METHODOLOGY

1.1 Question Collection, Categorization, and Enrichment

CodeGen operates as a staged pipeline. Collectors obtain top interview programming questions from multiple websites. API-based sources are read as JSON and other pages are scraped using Python. A raw record stores the title, source link, raw statement, source tags, and any stated company. Empty records and duplicates with the same title or nearly the same statement are removed.

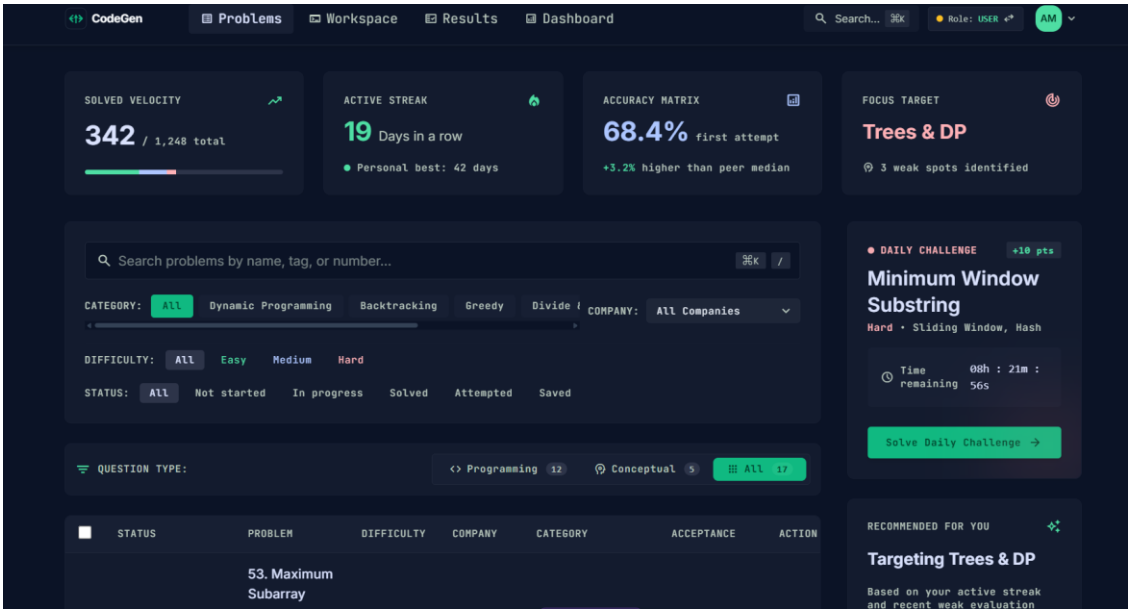


Table 1 - CodeGen components described in the source paper

Component	Role in CodeGen
Frontend	React interface for portal pages, filtering, workspace, results, and dashboard
Backend	FastAPI REST service and Python-based AI pipeline
Persistence	MongoDB collections for users, raw questions, verified questions, and conceptual questions
Collection	API-based JSON collection and Python web scraping with duplicate removal
AI processing	LLM-based categorization, tagging, enrichment, and correction prompts
Verification	Judge0 execution service runs generated solutions against ten test cases
Authentication	JWT sessions and bcrypt password hashing
Export	Checkbox-selected question records returned as JSON

Each cleaned question is sent to the LLM with a strict JSON instruction. The model assigns one category from Dynamic Programming, Backtracking, Greedy, Divide and Conquer, Two Pointers, or Other; estimates Easy, Medium, or Hard; and returns a company name or General. Requests are batched with delays to respect free-service rate limits. The LLM then writes a detailed problem statement, examples with explanations, constraints, relevant edge cases, ten test cases, and a Python or Java solution.

1.2 Verification Loop

The generated solution is executed against all ten test cases through an online code-execution engine. If every test passes, the question is marked verified. If any test fails, the failing input, expected output, and error are returned to the model for correction. After a fixed number of failed attempts, the question is marked for review and is not published. Generated code is executed by the external service rather than on the application server.

Before enrichment, a classifier distinguishes programming questions from conceptual questions such as machine learning, operating-system, or database interview questions. Conceptual questions are stored separately and shown through a Conceptual filter in the portal.

1.3 Portal and Selective Export

Verified questions are served through REST APIs. Users can search and filter by category, company, difficulty, and status; solve questions in a workspace; view results; select questions with checkboxes; and export only the selected records as a JSON file containing their statements, examples, constraints, test cases, tags, and solutions.

IV. SYSTEM IMPLEMENTATION

Frontend. The React frontend uses a dark, code-oriented interface with landing, login, and account-creation pages. Its main navigation includes Problems, Workspace, Results, and Dashboard. The Problems page has search, category, company, difficulty, and status filters; a Programming, Conceptual, or All switch; and checkboxes for selecting questions.

Backend. FastAPI in Python provides the REST service and AI-pipeline support. The login API validates credentials and returns a signed JSON Web Token; account creation validates user information, rejects duplicate email addresses, hashes passwords with bcrypt, and stores the new user. Protected routes check the token before returning data.

Database. MongoDB stores users, raw questions, verified questions, and conceptual questions. A verified-question document contains the title, source, category, difficulty, company tag, problem statement, examples, constraints, ten test cases, solution language and code, and verification result. Indexes on category, difficulty, and company support filtering.

AI Pipeline. Python scripts call free LLM services or open models. Each stage requests strict JSON, checks required fields before storage, and batches or delays calls to remain within free usage limits.

Code Verification. Solutions and test cases are sent to the Judge0 execution service. The returned output is compared with the expected output, and a solution is accepted only when all ten comparisons match.

JSON Export. When a user selects questions and chooses download, the frontend sends their identifiers to the export API. The server returns a JSON file containing only those questions and their associated learning material.

Table 2 - System components and roles

Component	Implementation	Purpose
Question collection	API and page sources	Obtain raw interview questions and source metadata
Categorization	LLM strict JSON output	Assign algorithm category, difficulty, and company or General tag
Enrichment	LLM strict JSON output	Generate statement, examples, constraints, edge cases, tests, and solution
Verification	Online code-execution engine	Accept only solutions passing all ten generated tests
Conceptual detection	Classification step	Separate non-programming interview questions
Portal	React and REST APIs	Search, filter, solve, view results, and export selected questions

Testing and deployment. The source describes checks for valid and missing login input, wrong passwords, and duplicate emails. The pipeline is checked question by question through its verification result. It proposes free static hosting for the frontend, free cloud hosting for the backend, and a free MongoDB cluster; no production deployment measurements are reported.

V. ADVANTAGES

The source paper identifies the following design advantages. These are feature descriptions, not comparative experimental findings.

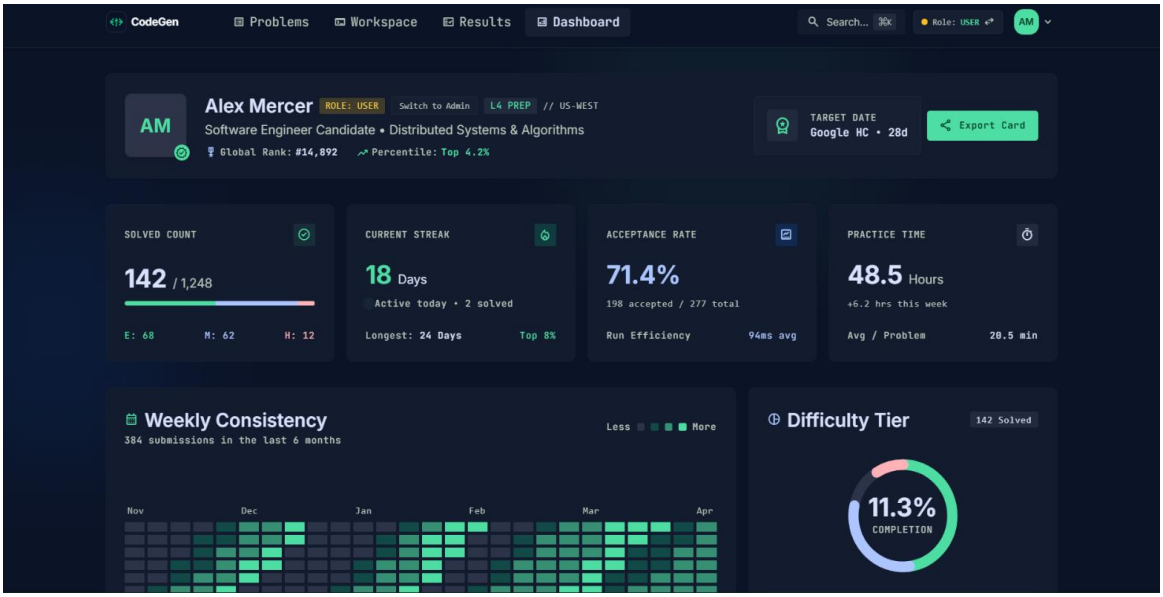


Table 3 - Advantages described for CodeGen

Advantage	Source-supported detail	Scope
Verified accuracy	A question is published only after its generated solution passes all ten designated test cases	Verification workflow; no aggregate accuracy rate reported
Self-correction	Failure feedback is returned to the model for correction	Workflow description
Complete question package	Statements, examples, constraints, tests, tags, and a solution are stored together	Feature description
Reduced manual effort	Collection, categorization, and generation are automated	Design intent
Selective download	Only checkbox-selected questions are exported as JSON	Implemented portal function
Secure modular design	JWT, bcrypt, and separable application modules support extension	Implementation description
Wider coverage	Conceptual questions are separated from programming questions	Portal feature

Table 4 - Source-supported portal functions

Portal area	Source-described function	Evidence limitation
Problems	Search; category, company, difficulty, and status filters; Programming, Conceptual, and All views; checkboxes	Interface description from source
Workspace	Problem statement, examples, hints, editor, and test-case panel	Interface description from source
Results	Verdict, language, execution time, memory use, coverage, submitted code, and recommendation	Interface description from source
Dashboard	Progress summary, solved count, streak, acceptance rate, practice time, and	Screenshot values are explicitly sample data

	charts	
Export	JSON download of selected questions and their associated fields	Feature description from source

VI. RESULTS AND ANALYSIS

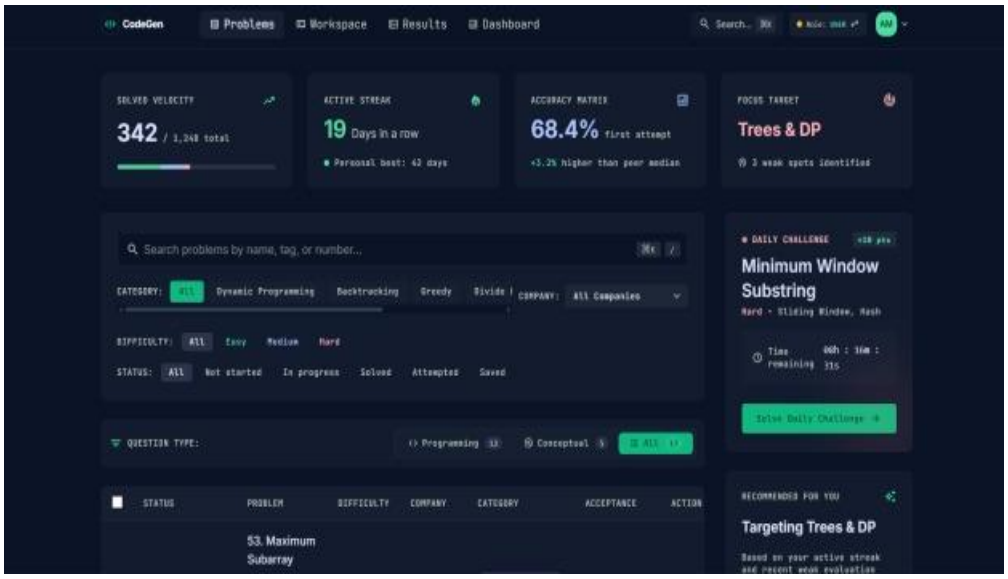


Fig. 3 - Problems page with filters and checkbox selection

Table 5 - Source-supported observations from the implemented portal

Observation	Source-supported description	Caution
Question path	A collected question can be enriched with tags, tests, and a solution before publication	No aggregate throughput or success-rate result supplied
Verification	Solutions are executed against ten generated test cases	Passing generated tests does not establish benchmark-level correctness
Filtering	Questions can be filtered by algorithm category, company, difficulty, and status	No usability study reported
Conceptual view	Non-programming interview questions are shown separately	Classification accuracy not reported
Dashboard	Progress statistics are displayed	Source labels screenshot values as sample data

Results and Analysis

The source describes an implemented portal that brings question collection, enrichment, solving, and export into one application. The Problems page supports search and filtering; the Workspace presents the statement, examples, hints, editor, and test panel; the Results page displays verdict and execution details; and the Dashboard presents progress information. The source explicitly states that dashboard values shown in its screenshot are sample data. The verification stage is central to the design because it filters generated solutions before publication. It also notes limitations: generated statements and tests

depend on the LLM, free services limit requests, and source websites must be collected according to their terms. The supplied paper does not report aggregate verified-question rate, expert review, benchmark comparison, latency, or user-study outcomes.

VII. CONCLUSION

CodeGen demonstrates a workflow in which interview programming questions are collected, categorized, enriched, and verified automatically in one portal. Each published question can carry a category, difficulty, company tag, ten edge-case tests, and a Python or Java solution that has passed its designated test cases in an execution engine. React, FastAPI, and MongoDB provide the application structure, while checkbox-based JSON export lets users retain only the questions they choose. The project illustrates how free AI services can be combined with execution-based checking to improve the reliability of generated educational content.

VIII. FUTURE WORK

The source proposes adding more question sources and algorithm categories, including graphs, trees, binary search, and bit manipulation; more solution languages such as C++ and JavaScript; and larger-scale evaluation with verified-question-rate measurement and expert review. Other proposed extensions include personalized recommendations, an administrator review view for flagged questions, timed mock interviews, broader conceptual-question coverage, additional export formats, and a mobile application.

REFERENCES

[1] M. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv:2107.03374, 2021.

[2] J. Austin et al., "Program Synthesis with Large Language Models," arXiv:2108.07732, 2021.

[3] D. Hendrycks et al., "Measuring Coding Challenge Competence With APPS," Proc. NeurIPS Datasets and Benchmarks Track, 2021.

[4] Y. Li et al., "Competition-level code generation with AlphaCode," Science, vol. 378, no. 6624, pp. 1092-1097, 2022.

[6] T. Brown et al., "Language Models are Few-Shot Learners," Advances in Neural Information Processing Systems, vol. 33, 2020.