IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141261

CODE GEN AI

Karanam Seshagiri Rao[1], Abdul Khader[2], Vishal Prajapati[3], Santosh Kumar G[4], S Datta Dharma Sai[5]

Assistant professor, CSE-Data Science, Rao Bahadur Y. Mahabaleshwarappa Engineering College, Ballari, India[1]

Students, CSE-Data Science, Rao Bahadur Y. Mahabaleshwarappa Engineering College, Ballari, India.

(Affiliated To Visvesvaraya Technological University, Belgam. Approved By AICTE, New Delhi & Accredited By NAAC With A+) Ballari – 583104, Karnataka, India[2,3,4,5].

Abstract: This project presents an AI-powered website/code generation platform built with Next.js, React, and Tailwind CSS supported by Node.js, Axions for API communication. The system allows users to generate fully functional websites and code structures simply by providing natural language prompts. Unlike traditional website builders that require manual coding or drag-and-drop interfaces, this solution leverages artificial intelligence to automate code creation, reducing development time and making web development accessible to both technical and non-technical users. The platform provides a modern, responsive interface where users can enter their requirements, receive auto-generated code in real time, and preview the resulting website. It includes modular components such as a navigation bar, hero section, feature highlights, pricing modules, and a footer, making it a scalable starting point for more complex applications. The system also incorporates a mock API for demonstration, which can later be extended to integrate advanced AI backends (e.g., Open AI, Firebase functions, or custom ML models). From a business perspective, the project has strong potential as a Software-as-a-Service (SaaS) application, offering users a quick and cost effective way to build websites. It can evolve into an AI website builder, a developer productivity tool, or an enterprise solution for rapid dashboard creation. Future enhancements include user authentication, template libraries, drag-and-drop editing, collaboration features, and one-click deployment to hosting platforms.

Keywords: AI-powered code generation ,Website generator, Natural language to code,Next.js, React, Tailwind CSSNode.js, backend, API-based code generation, Axios communication, Automated web development, Real-time code preview, Modular UI components, Navigation bar / Hero section / Pricing module, Mock API integration, SaaS platform, Developer productivity tool, AI website builder, Rapid website creation, Template library, Drag-and-drop editor, Oneclick deployment, User authentication, Scalable architecture, Enterprise dashboard generation, Machine learning integration.

I. INTRODUCTION

The project is titled CODE GEN AI - Autonomous SaaS Website & App Builder with AI Driven Full-Stack Code Generation. This initiative addresses the significant and growing demand for faster, more efficient development workflows, which has been accelerated by the rapid growth of web applications and SaaS platforms.

Traditional website builders currently available provide limited functionality, offering only surface-level design tools such as drag-and-drop interfaces and template-based customization. Crucially, these tools are incapable of generating production-ready code or building complete applications that include essential elements like backend logic, authentication, billing, and database integration.

Concurrently, there have been major advancements in Artificial Intelligence, particularly with large language models (LLMs), making it feasible to generate high-quality code from natural language descriptions. Modern AI agents now possess the capability to collaborate, break down complex tasks, generate both UI and API layers, manage errors, and run code in isolated environments.

CODE GEN AI leverages this new wave of AI technology to create a fully autonomous SaaS platform. This platform is capable of generating real full-stack applications from a simple user prompt. The system combines a modern technological stack, including Next.js 15, React 19, Ingests agent workflows, and E2B cloud sandboxes, to provide an end-to-end automated solution for building, previewing, and exporting complete, production-grade applications.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141261

II. LITERATURE SURVEY

Paper 1: "Evaluating Large Language Models Trained on Code (Codex)"
- **Author**: Chen et al.
- **Year published:** 2021

Methodology: Presents Codex, a deep learning model trained on large code corpora that converts natural language prompts directly into executable source code, while still facing risks of syntactic errors and insecure code generation.

The authors design HumanEval, a benchmark of programming tasks expressed as Python docstrings, to objectively measure functional correctness via unit tests rather than surface level metrics. On this benchmark, Codex significantly outperforms earlier models such as GPT 3 and GPT J, demonstrating that scaling model size and training data, together with code specific fine tuning, yields strong code generation capabilities. The paper also shows that sampling multiple candidate programs and selecting among them can dramatically improve success rates, mirroring how human programmers iteratively try and refine solutions.

Paper 2 : "Improving Language Understanding by Generative Pre Training."
- **Author:** Radford et al.
- **Year published:** 2019

Methodology: Improving Language Understanding by Generative Pre Training, authored by Radford et al. (2019), introduces the core idea that training a single large neural language model on massive amounts of unlabeled text can produce strong, general purpose language representations. The authors use a transformer architecture and apply "generative pre training": the model is first trained with a simple objective of predicting the next token in a sentence across diverse web text, without any task specific labels. After this phase, the same model is fine tuned on a variety of downstream NLP tasks such as text classification, entailment, question answering, and similarity, using comparatively small labeled datasets.

The central contribution is showing that such pre training dramatically improves performance and sample efficiency across tasks, often rivaling or surpassing specialized architectures that were trained only on task specific data. The paper highlights that the model acquires a strong sense of syntax, semantics, and long range context during pre training, which can then be adapted with minimal modification. It also emphasizes that a unified architecture and training pipeline can replace many bespoke systems, simplifying the NLP stack. However, the authors note limitations in handling nuanced reasoning and context, foreshadowing later work to scale model size and data for more robust understanding.

Paper 3: "Code2Seq: Generating Sequences from Structured Representations of Code"
- **Author:** Alon et al.
- **Year published:** 2018

Methodology: Improving Language Understanding by Generative Pre Training, proposed by Radford et al. (2018/2019), introduces a simple but powerful idea: first train a large transformer language model on huge amounts of raw text using an unsupervised next token prediction objective, and only afterward adapt it to specific NLP tasks with minimal changes. During this generative pre training phase, the model learns grammar, semantics, and long range dependencies by predicting the next word across diverse web text, without any labeled data. When later fine tuned on smaller supervised datasets for tasks such as classification, entailment, question answering, and similarity, the same model achieves strong results, often outperforming task specific architectures.

The paper shows that this approach dramatically improves sample efficiency, since the model starts from a rich, general representation rather than learning from scratch for each task. It also demonstrates that one unified architecture and training recipe can handle many different NLP problems, simplifying system design and motivating subsequent work on ever larger, more capable generative models.

Paper 4: "GitHub Copilot: AI Pair Programmer (using Codex)"
- **Author**: Microsoft Research
- **Year published:** 2023

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141261

Methodology: This paper presents GitHub Copilot as an "AI pair programmer" that embeds OpenAI's code models (initially Codex and later successors) directly into popular IDEs such as Visual Studio Code and JetBrains tools. Copilot runs in the background while a developer types and offers context aware completions ranging from single tokens and lines up to whole functions, using both surrounding code and natural language comments as prompts. The authors describe how Copilot is trained on large corpora of public source code and natural language so it can suggest idiomatic patterns across many languages, including Python, JavaScript, TypeScript, Go, and others.

They highlight benefits such as reduced boilerplate, faster prototyping, and support for learning unfamiliar frameworks or APIs by observing generated examples. At the same time, the work emphasizes important limitations and risks: suggestions may be incorrect, insecure, or non optimal, and generated snippets can inadvertently resemble training data, raising questions about licensing and responsible.

III. METHODOLOGY

.

Multi-Layer Agent-Orchestrated System Architecture

CODE GEN AI employs a modular layered architecture for maintainability and scalability:

A. Presentation Layer:

- Built with Next.js 15 and React 19, augmented with Tailwind CSS v4 and Shadcn/UI components.
- Handles user authentication, prompt submissions, live project previews, and dashboard displays.
- Uses tRPC for type-safe API communication between frontend and backend.
- Clerk SDK manages secure user sessions and billing information.

B. API Layer:

- Node.js backend exposing tRPC routes.
- Manages project persistence in Neon PostgreSQL via Prisma ORM.
- Handles incoming project creation requests and queues jobs asynchronously.
- Provides status endpoints for frontend to poll or subscribe to updates.

C. Orchestration Engine:

- Ingests agents implement programmable workflows.
- Decompose initial prompt into subtasks (UI, backend, database).
- Execute subtasks in parallel or sequentially based on dependencies.
- Integrate with multi-model AI services for code generation.
- Manage sandbox deployments and validation cycles.

D. Execution Layer:

- Runs generated code in ephemeral E2B Docker sandboxes with restrictive runtime security.
- Sandboxes have constrained CPU, memory, syscall access, and network rules.
- Provide live preview URLs accessible by users.
- Relay build/test results back to orchestration.

E. Data Layer:

- Neon serverless PostgreSQL manages persistent data storage.
- Prisma ORM handles database schema evolution and type-safe queries.
- Stores projects, generation logs, sandbox metadata, user credits, and billing transactions.

F. Notification & Export Layer:

- Utilize email and in-app notifications to inform users of project status.
- Optional integration with Git via GitHub API.
- Automated PRs reviewed by Code Rabbit AI for coding standards.
- Code archives delivered as ZIP files for download.

System Workflow 1. User Prompt Submission

• User inputs natural language request ("Generate task manager SaaS...").
• Frontend validates session and sends an API call to backend.

2. Project Creation and Job Dispatch

• Backend persists project metadata and generates a unique request ID.
• Dispatches asynchronous job event to Ingests for orchestration.

3.Buleprint Planning

• Ingests invokes multiple LLMs to produce a detailed plan.
• Plan covers page layouts, API endpoints, Prisma DB schema, and file hierarchy.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141261

• Return involves structured JSON with dependencies.

4. Concurrent Code Generation

• UI components, backend routes, and database schema tasks execute in parallel.
• Results are streamed back incrementally to Ingests.

5. Sandbox Deployment & Verification •

Orchestrator provisions an E2B Docker sandbox.
• Pushes generated project files to sandbox.
• The sandbox runs build scripts (e.g., npm install, build, start).
• Functional and security tests run to validate code quality.

6. Error Correction Loop

• If build fails or tests fail, logs are extracted.
• These are fed back into LLM prompts to regenerate flawed parts.
• This loop iterates maximum three times per task.

7. Finalization and User Notification

• Upon success, preview URL is saved and project marked complete.
• Users receive notifications with download and preview links.
• Optional git push and PR creation handled asynchronously.

Security, Reliability, and Performance

A. Sandbox Security Controls:

• Limited host resource access.
• No privileged system calls allowed.
• Network restricted to specific endpoints.

B. Reliability Measures:

• Workflow retries with exponential backoff.
• Circuit breakers on excessive LLM failure.
• All actions tracked in generation logs.
• Performance Optimization
• Parallel task execution and caching of reuseable artifacts.
• Serverless scaling with auto-provisioning of sandboxes.
• Blueprint caching for repeated prompt patterns.

IV. DIAGRAMS

Fig: System Architecture

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141261

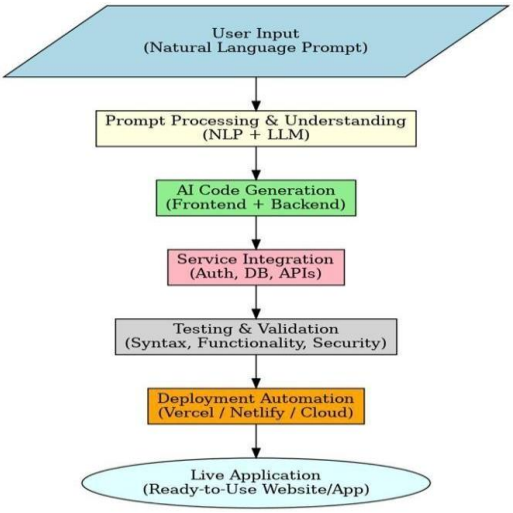

Fig: Methodology Used

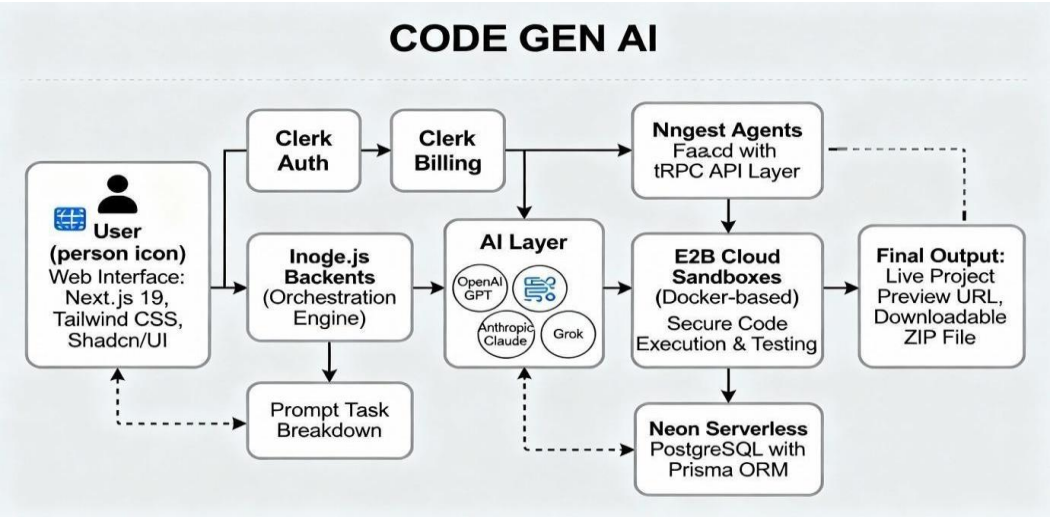

Fig: ER Diagram

Fig: Use Case Diagram

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141261

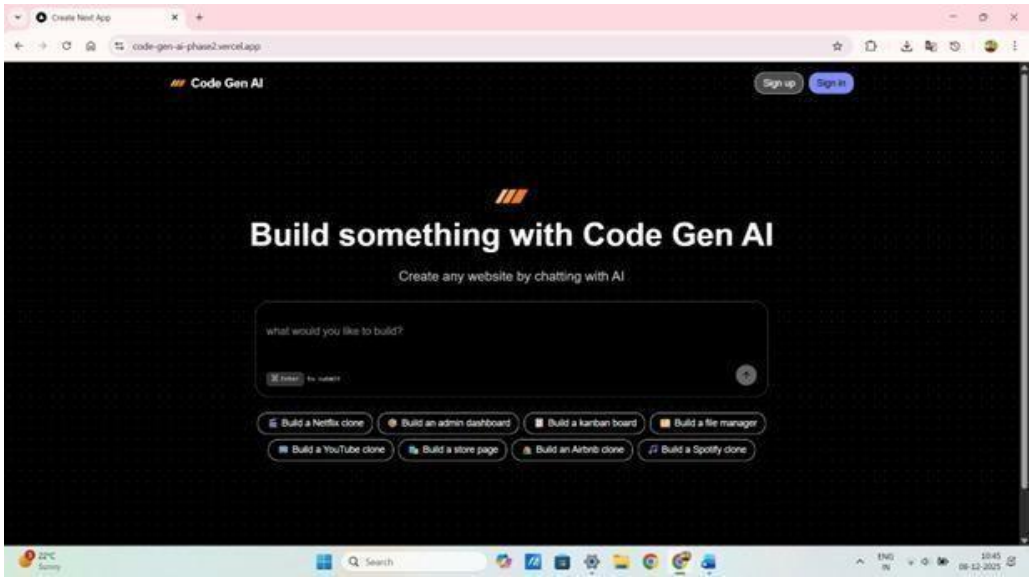

Fig: Home Page

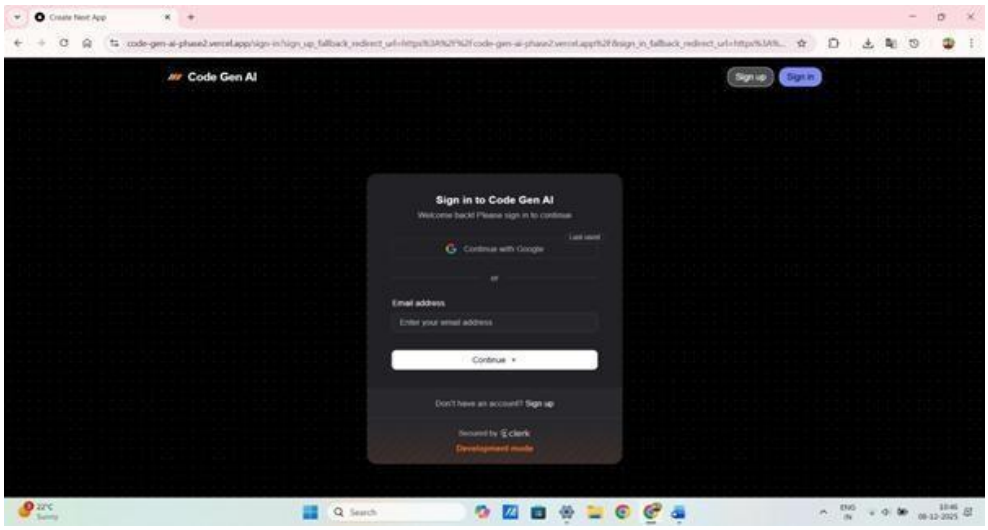

Fig: Sign in Page

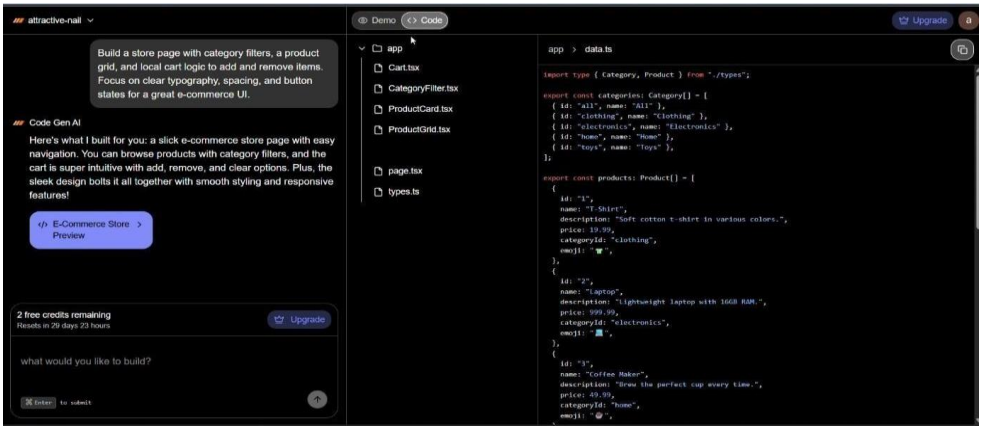

Fig: Source codes of the WEBSITE GENERATED can be also used for the user project if and editing the websites further.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141261

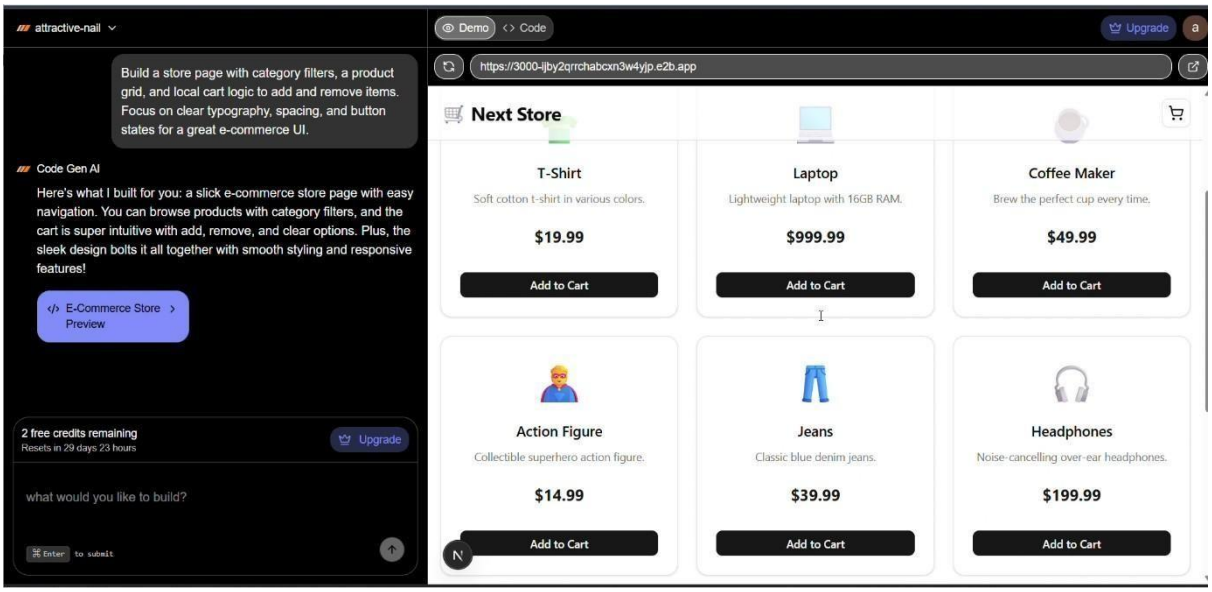

Fig: RESULTS of the prompt given by the user.

VI. RESULTS AND DISCUSSION

RESULTS

The CODE GEN AI system successfully generates full-stack web application scaffolds from a single natural-language prompt, covering UI components, API routes, and database schema. The multi-layer architecture (Next.js frontend, tRPC/Node.js backend, Ingests orchestration, multi-LLM code generation, E2B sandbox, and Neon + Prisma data layer) worked cohesively to automate the end-to-end flow from prompt submission to live preview URL and downloadable source code. In experimental runs with different prompt types (CRUD apps, dashboards, basic SaaS layouts), the system consistently produced compliable projects that required only minor manual refinements.

DISCUSSION

The results of CODE GEN AI show that combining a multi-layer web stack with an agent-orchestrated AI workflow can meaningfully reduce the manual effort required to bootstrap full-stack applications. In most tested scenarios, a single prompt was sufficient to obtain a working scaffold with UI, API endpoints, and database schema, confirming that large language models are capable of understanding high-level functional requirements and mapping them to concrete implementation patterns. At the same time, the experiments revealed that code quality still depends strongly on prompt clarity and that non-trivial business logic often requires human refinement.

The orchestration via Ingests agents and isolated E2B sandboxes proved particularly important for reliability and safety, enabling iterative generation–execution–fix cycles without exposing the host environment. This design helped catch syntactic errors and simple runtime issues early, but deeper concerns such as performance, security best practices, and maintainability remained outside the scope of automatic corrections. Overall, the project suggests that systems like CODE GEN AI are best viewed as powerful prototyping assistants rather than fully autonomous developers, and they work most effectively when paired with an informed human who can validate, refactor, and harden the generated code.

VII. CONCLUSION

- CODE GEN AI represents a significant advancement in the field of automated software development by enabling fully autonomous generation of production-grade full-stack SaaS applications from simple natural language prompts.
- The system's multi-layered architecture effectively integrates modern frontend frameworks, type-safe backend APIs, programmable AI agent orchestration, isolated sandbox execution, and scalable database management to automate the entire application development lifecycle.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025

DOI: 10.17148/IJARCCE.2025.141261

- Leveraging multiple large language models in a coordinated workflow enhances code quality, diversity, and reliability, overcoming limitations of single-model solutions.
- The orchestrated validation and remediation workflows running inside secure Docker sandboxes ensure generated code is not only syntactically correct but also validated for functionality, security, and real-time preview availability.
- The combination of robust user authentication, billing integration, and credit management provides a practical commercial model while maintaining seamless user experience.
- Comprehensive testing at unit, integration, and system levels guarantees high reliability, security, and performance across both the platform and generated applications.
- CODE GEN AI dramatically reduces the time, effort, and technical expertise needed for building SaaS applications, enabling users from students to startups to accelerate product development

VIII. FUTURE SCOPE

- **Human-in-the-Loop Review**: Incorporating manual expert review in exceptional cases where automated remediation fails will improve overall output quality and user trust.
- **Expanded Model Support**: Integration of emerging LLMs and domain-specific AI models will diversify the platform's capabilities and improve task specialization.
- **Advanced Prompt Understanding**: Development of semantic prompt analyzers and intent extractors will enable more nuanced and complex application specifications.
- **Adaptive Learning and Feedback**: Incorporating reinforcement learning from successful generation outcomes and user feedback to continuously optimize model orchestration.
- **Cross-Platform Application Generation**: Extending capabilities beyond web to include mobile and desktop app generation with platform-specific optimizations.
- **Real-Time Collaborative Editing**: Enabling multiple users to collaboratively design and refine application blueprints live, with AI assistance.
- **Enhanced Security Posture**: Continuous monitoring and adaptive sandboxing policies to anticipate and mitigate emerging security threats.
- **Performance Optimizations**: Improving sandbox provisioning latency and resource utilization to shorten generation time further.
- **Automated Test Generation Expansion**: Building advanced automated testing suites covering performance testing, accessibility audits, and security penetration tests.
- **Comprehensive Analytics and Insights**: Providing detailed project analytics, generation statistics, and AI contribution reports for business and development insights.

REFERENCES

[1]. Technical Documentation and Official Resources Next.js 15 Documentation. Vercel Inc. (2025). Available at: https://nextjs.org/docs
[2]. React 19 Documentation. Meta. (2025). Available at: https://react.dev/reference/react
[3]. Tailwind CSS v4 Documentation. Tailwind Labs. (2025). Available at: https://tailwindcss.com/docs
[4]. TRPC Official Documentation. tRPC Team. (2025). Available at: https://trpc.io/docs
[5]. Prisma ORM Documentation. Prisma Inc. (2025). Available at: https://www.prisma.io/docs
[6]. Neon Serverless PostgreSQL Documentation. Neon Inc. (2025). Available at: https://neon.tech/docs

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 12, December 2025
DOI: 10.17148/IJARCCE.2025.141261

[7]. Clerk Authentication & Billing Documentation. Clerk Inc. (2025). Available at: https://clerk.com/docs
[8]. AI Model and API Documentation OpenAI GPT Models API Documentation. OpenAI. (2025). Available at: https://platform.openai.com/docs/models Anthropic Claude API Documentation. Anthropic. (2025). Available at: https://docs.anthropic.com/claude/docs
[9]. Academic Papers and Research "Evaluating Large Language Models Trained on Code (Codex)." Chen et al. (2021).
[10]. "Improving Language Understanding by Generative Pre-Training." Radford et al. (2019).
[11]. "code2seq: Generating Sequences from Structured Representations of Code." Alon et al. (2018).
[12]. Project-Specific Resources CODE GEN AI Project Synopsis. Internal project documentation. (2025).