iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** III **Month of publication:** March 2026

DOI: https://doi.org/10.22214/ijraset.2026.78390

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue III Mar 2026- Available at www.ijraset.com

Nova AI: AI-Powered Code Generation SaaS Using Next.js and LLM

Shubham Das[1], Priyansh Dubey[2], Adarsh Mishra[3], Ishaan Pandey[4], Rajeshri Lanjewar[5]
[1, 2, 3]*Student, Computer Science and Engineering, Shri Shankaracharya Technical Campus*
[5]*Assisant Professor, Computer Science and Engineering, Shri Shankaracharya Technical Campus*

Abstract: ***In the contemporary software development landscape, transforming natural language ideas into runnable code remains time-consuming and requires significant expertise. Traditional development workflows involve manual coding, repeated iterations, and steep learning curves for non-experts. Recognizing these limitations, this research proposes and presents the development of Nova AI, an AI-powered code generation platform designed as a Software as a Service (SaaS) product that automates the creation of full React applications from natural language prompts. Built on a modern technology stack incorporating Next.js for server-side rendering, React and Tailwind CSS for the frontend, Convex for real-time backend and database operations, and Google's Gemini LLM for code generation, prompt enhancement, and architecture explanation the platform delivers an intuitive user experience. It offers functionalities such as prompt input with AI-based enhancement, real-time AI-generated React code, live preview and editing via Sandpack, conversational refinement through an in-workspace chat, high-level and low-level design explanations with flowcharts, workspace persistence, token-based usage tracking, and export or deploy options. Deployment on Vercel ensures scalability and accessibility. By integrating cutting-edge large language models with modern web frameworks, this research highlights a viable path toward scalable, efficient, and democratized code generation, addressing gaps in existing AI coding tools while paving the way for future enhancements such as multi-framework support and advanced code analysis.***

Keywords: AI, Code Generation, SaaS, Next.js, LLM, React, Gemini, Convex, Sandpack.

I. INTRODUCTION

Software development plays a vital role in driving innovation across industries and significantly influences the ability of organizations and individuals to transform ideas into working applications. With the rapid growth of digital products and increasing demand for rapid prototyping, building functional software has become more challenging than ever. Organizations and developers today seek tools that not only reduce development time but also lower the barrier to entry for non-experts, support iterative refinement, and provide clear understanding of generated architecture. However, many users remain unable to translate their ideas into code due to lack of programming expertise or the time required for manual development, resulting in delayed projects and underutilized creativity. Traditional development workflows rely on hand-written code, repeated debugging cycles, and specialized knowledge of frameworks and languages, which may not effectively support quick experimentation or learning.

In recent years, the advancement of large language models (LLMs) and natural language processing has opened new opportunities for applying artificial intelligence in the software development domain. LLMs can interpret natural language prompts and generate coherent, structured code across multiple files and components. By leveraging these techniques, it is possible to build systems that produce complete, runnable applications from a short description enabling users to prototype without deep coding experience and allowing experienced developers to accelerate their workflow. Such capabilities also allow for conversational refinement, where users can request changes or clarifications and receive updated code, thereby bridging the gap between idea and implementation.

The study focuses on the development of **Nova AI**, an AI-powered code generation platform designed as a Software as a Service (SaaS) product using Next.js and Google's Gemini LLM. The system aims not only to generate full React applications from natural language prompts but also to enhance user prompts for better results, provide an interactive chat interface for iterative refinement, and deliver high-level and low-level design explanations with flowcharts. The platform integrates Next.js for server-side rendering and API routes, React and Tailwind CSS for the frontend, Convex for real-time backend and workspace persistence, and Sandpack for in-browser code editing and live preview. Data flow is designed so that user input triggers workspace creation, parallel AI calls for chat and code generation, and persistent storage of messages and generated files. Beyond prediction and generation, the system provides analytical insights through the Explain feature, which summarizes architecture and design baed on the current conversation context, and supports export or deploy actions so users can take their generated project outside the platform.

Beyond generation, the system provides analytical insights and personalized refinement that help users understand their project's structure and focus on specific improvements. The Explain feature delivers high-level and low-level design summaries, enabling users to grasp architecture and data flow without reading all generated code. At the user level, this supports learning and faster iteration; at the institutional or team level, the platform can support teaching, hackathons, and rapid prototyping by letting participants concentrate on requirements and refinement rather than initial boilerplate.

The integration of large language models into a full-stack code generation platform represents a significant step toward AI-assisted and democratized software development. By enabling prompt-based generation, continuous refinement through chat, and transparent design explanation, the framework contributes to faster product iteration, improved accessibility of coding for non-experts, and stronger alignment between user intent and generated implementation.

II. LITERATURE REVIEW

The capabilities of large language models for few-shot learning and text generation have been extensively studied, for example in the GPT-3 work [2]. Brown et al. (2020) introduced GPT-3, demonstrating few-shot learning capabilities that extend to code completion and generation. Chen et al. (2021) presented Codex, a model trained on code that powers tools like GitHub Copilot [4].

Commercial and open-source platforms such as Bolt.new, v0, and Replit AI offer prompt-to-code experiences but often lack integrated prompt enhancement, persistent workspaces, and structured architecture explanation in a single flow.

These models are commonly based on transformer architectures, which introduced attention mechanisms for sequence modeling [3]. Academic work on code completion and program synthesis has focused on statistical and neural approaches (e.g., Raychev et al. [11]). However, many existing tools are either commercial with limited customizability or do not combine prompt improvement, multi-turn chat, live editing, and HL/LL documentation in one cohesive platform. Our approach provides an open, modular system that integrates Next.js, a real-time backend (Convex), and Gemini for end-to-end code generation with enhancement, chat, and explain features, filling a gap between general-purpose chatbots and specialized IDE plugins. A closely related system is an AI content generator SaaS built with Next.js and large language models, which focuses on generating textual content rather than application source code [1].

III. TECHNOLOGY TABLE

Table no.3.1) Technology Table

Term	Definition	Usage in Project
Next.js	A React-based framework for server-side rendering (SSR), static generation, and full-stack applications with API routes.	Powers the web application, API routes for AI (chat, code gen, enhance, explain), and optimized loading.
React	A JavaScript library for building component-based, interactive user interfaces.	Builds the frontend UI: home page, workspace, chat, code editor, and modals.
Tailwind CSS	A utility-first CSS framework for rapid, consistent styling with pre-defined classes.	Styles the entire UI for a responsive, dark-themed, modern look.
Convex	A serverless backend platform providing real-time database, serverless functions, and reactive queries.	Manages users, workspaces, messages, and generated file data; provides real-time sync and mutations.
Google Gemini	A family of large language models (LLMs) exposed via API for text and code generation.	Powers code generation, chat responses, prompt enhancement, and HL/LL architecture explanations.
Sandpack	An in-browser bundler and runtime for running and editing React (and other) projects without a separate build step.	Renders live preview of generated code and provides an editable file explorer and code editor in the workspace.
Lucide React	A library of icon components for React applications.	Used for UI icons (buttons, sidebar, header, etc.) across the platform.
Vercel	A platform for hosting and deploying frontend and serverless applications with CI/CD.	Used for hosting and deploying the Nova AI application.

Users enter a project idea on the home page (optionally after enhancing the prompt). The system creates a workspace, redirects to the workspace page, and uses the Gemini API to generate both a conversational response and a full React project. Users can refine via chat, view and edit code in Sandpack, request HL/LL explanations, and export or deploy the result.

A. *Existing Method*

Traditional code creation relies on manual coding, which is time-consuming and requires expertise. Existing AI code tools often focus on snippets or single-file generation, lack integrated prompt enhancement, or do not provide persistent workspaces with chat and live preview in one place. Limitations include:

1) Lack of a unified flow from prompt enhancement to code generation to explanation.
2) Limited integration of real-time backend with workspace and token management.
3) Few tools offering both "try without sign-in" and full workspace history for signed-in users.

Comparison Table: Traditional Methods vs. Existing AI Code Tools vs. Nova AI

'Table no.3.2) Comparison Table

Aspect	Traditional Methods	Existing AI Tools	Nova AI
Code Generation	Human-written; slow and expertise-dependent.	Human-written; slow and expertise-dependent.	Full multi-file React project from one prompt; customizable via chat.
Coherence and Structure	Depends on developer skill and conventions.	Variable; may lack full project structure.	Structured output (App.js, components, Tailwind) with explanation and HL/LL design.
Customization	Full control but manual.	Limited to prompt and sometimes follow-up.	Prompt enhancement, multi-turn chat, and live code editing in one workspace.
User Engagement	Direct coding; steep learning curve.	Often single-shot or limited conversation.	Interactive chat, live preview, and explain overlay for learning.
Data and Workspace Persistence	Local or separate version control.	Often session-based or limited history.	Convex-backed workspaces and user history; token tracking.
Integration with Web Stack	Manual setup of frontend, backend, and deploy.	Standalone tools or IDE plugins.	Full stack: Next.js, Convex, Gemini, Sandpack, deploy/export from the same UI.
Scalability	Scales with team size and effort.	Depends on product; often rate-limited.	Token-based usage; scalable backend (Convex) and deploy (Vercel).

IV. METHODOLOGY

The system follows a modular architecture:

1) Client Interface: React components for prompt input (Hero), chat (ChatView), code editor and preview (CodeView), explain overlay (ExplainOverlay), header, and sidebar (workspace history).
2) Authentication Layer: Google OAuth and demo user creation; Convex stores user identity and token balance.
3) API Layer: Next.js API routes (/api/gen-ai-code, /api/ai-chat, /api/enhance-prompt, /api/explain) that call the Gemini API.
4) Database Layer: Convex handles users and workspaces (messages, fileData); real-time queries and mutations from the frontend.
5) AI Layer: Gemini used for code generation (JSON schema), chat, prompt enhancement, and HL/LL explanation.

A. *Implementation*

The platform includes:

1) Prompt input on the home page with an "Enhance" button that calls the enhance-prompt API.
2) Workspace creation on submit; redirect to /workspace/[id].
3) Chat interface for refinement; messages and AI responses persisted in Convex.
4) Parallel code generation from the same message context; output merged with default Sandpack files and displayed in CodeView.
5) Explain button that sends chat context to the explain API and shows HL/LL design (with Mermaid) in a modal.
6) Export (open in CodeSandbox) and Deploy (open live app URL) via Sandpack client.

Sidebar listing user workspaces; token deduction and optional pricing/subscription UI.

B. *Technologies Used*

1) Frontend: React, Tailwind CSS, Lucide React, React Markdown, Framer Motion
2) Backend: Next.js (API routes), Convex (database and real-time)
3) AI: Google Gemini (Gemini 2.5 Flash)
4) Code runtime: Sandpack (React template)
5) Authentication: Google OAuth (@react-oauth/google), Convex for user data
6) Deployment: Vercel

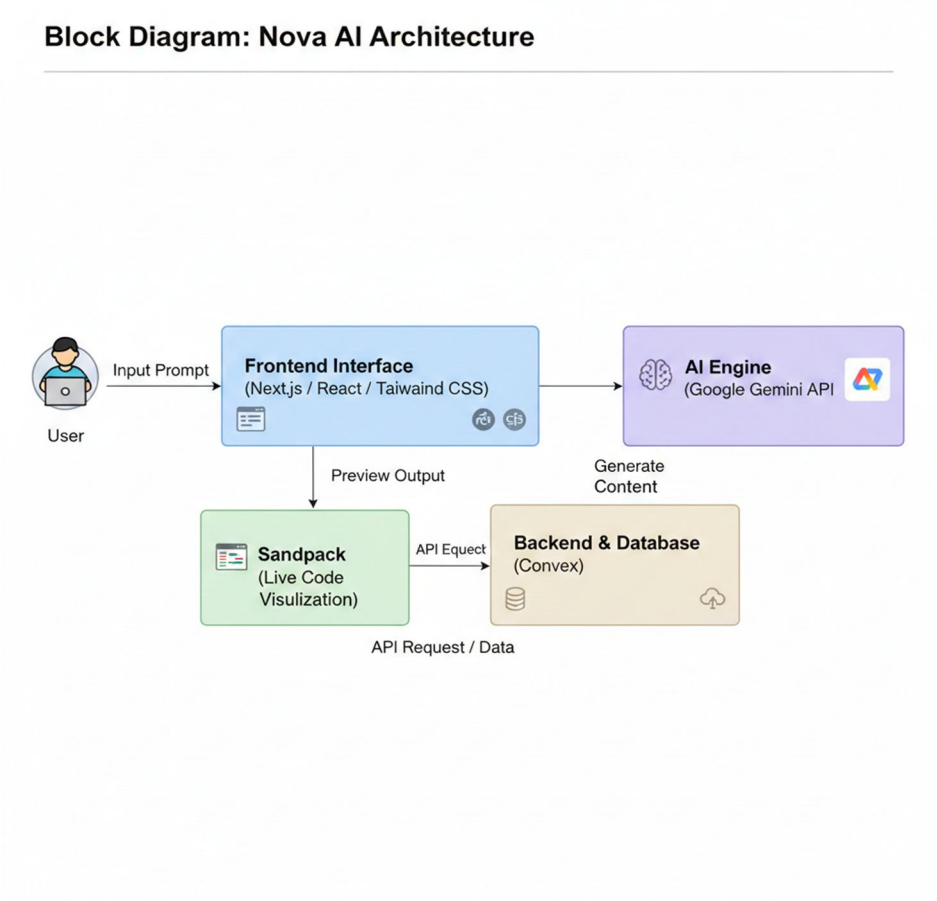


Figure no.4.1) Block Diagram

V. RESULTS AND DISCUSSION

Nova AI provides a working flow from natural language prompt to runnable React application with chat, enhancement, and explanation. Limitations include dependence on the Gemini API, token limits for free/demo users, and the current focus on React and Sandpack-supported dependencies. The following screens represent the main user journey:

1) Home Page: The main entry point where users enter or paste their project idea, use the Enhance button to improve the prompt, and submit to create a new workspace.

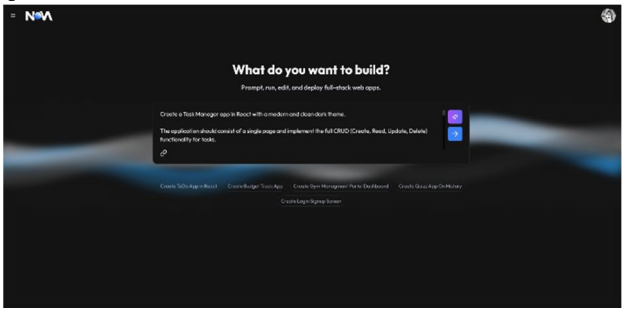


Figure no.5.1) Homepage

2) Sign In / Skip Sign In Dialog: A modal that allows users to sign in with Google or use “Skip Sign In” for limited-token demo access without an account.

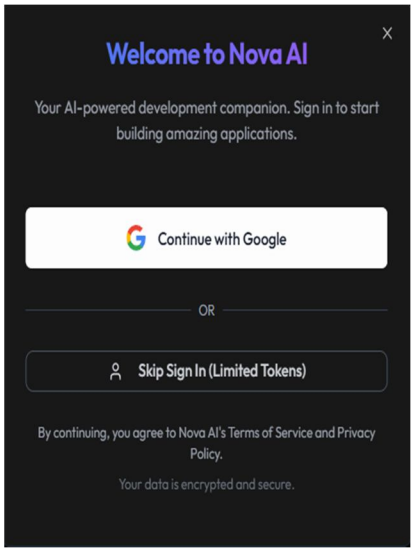


Figure no.5.2) Sign In / Skip Sign In

3) Workspace – Chat View: The left panel showing the conversation history (user and AI messages) and an input for follow-up requests to refine the project or ask questions.

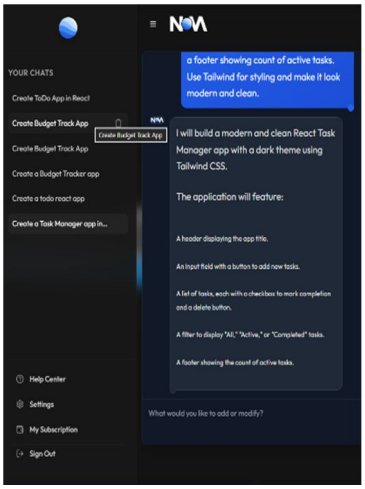


Figure no.5.3) Chat View

4) Workspace – Code View: The center panel with a file explorer and code editor (Sandpack) displaying the generated React project files (e.g. App.js, components).

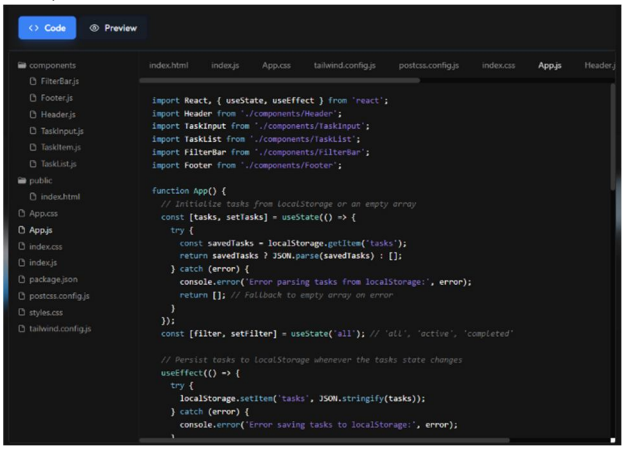


Figure no.5.4) Code View

5) Workspace – Preview Tab: The live preview of the generated application as rendered by Sandpack in the browser.

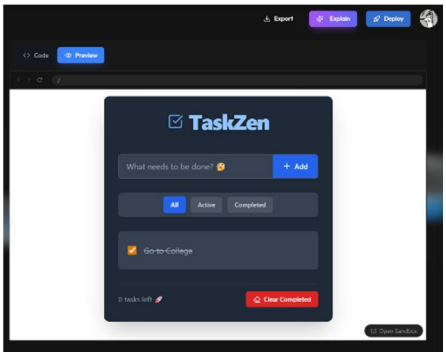


Figure no.5.5) Preview Tab

6) Explain Overlay: A modal that displays high-level and low-level design explanations and optional Mermaid flowcharts based on the current chat context.

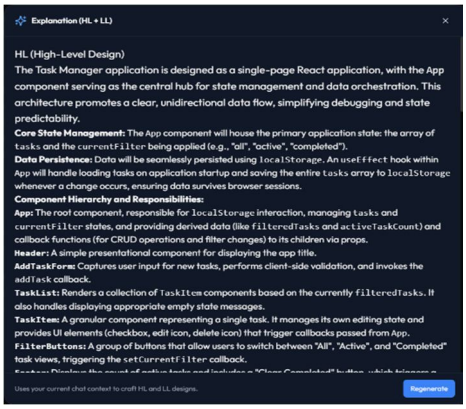


Figure no.5.6) Explain Overlay

7) Header (Export / Deploy): Buttons to export the project (open in CodeSandbox) or deploy and open the live preview URL.

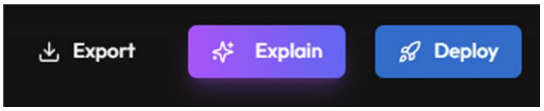


Figure no.5.7) Export / Deploy

8) Sidebar – Workspace History: A list of the user's saved workspaces with the first message as title; users can open or delete workspaces.

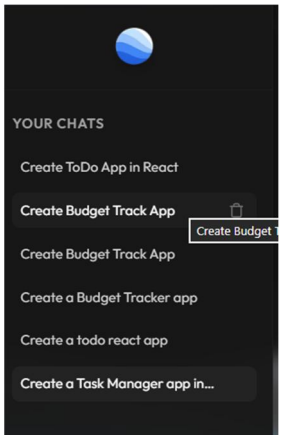


Figure no.5.8) Sidebar, Workspace History

VI. FUTURE SCOPE

In further future we want to add the following points:

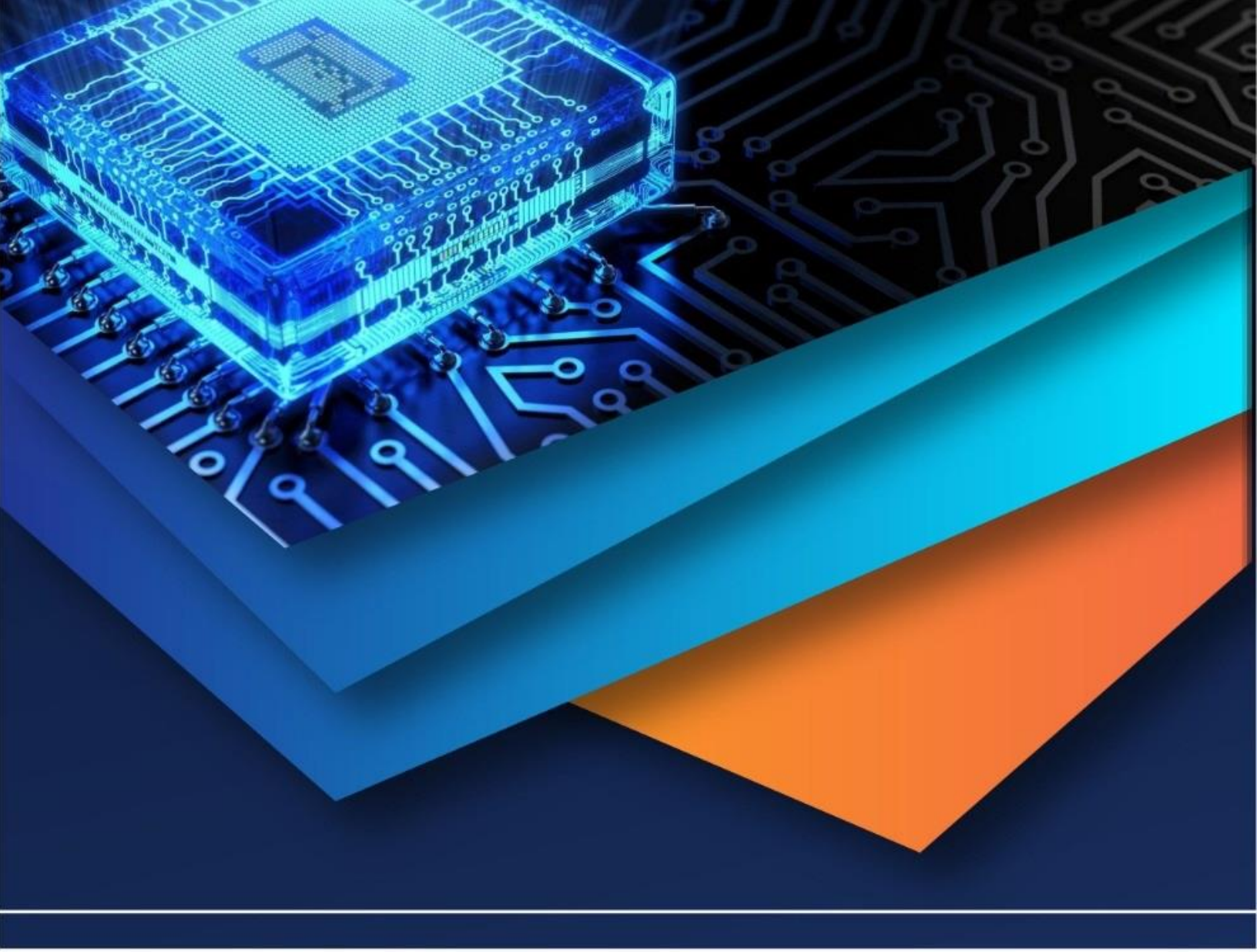
1) Multi-framework support: Generate Vue, Svelte, or other frameworks in addition to React.
2) Version control integration: Link workspaces to Git repositories and track changes.
3) Advanced code analysis: Linting, security checks, and performance suggestions on generated code.
4) Team collaboration: Shared workspaces and roles for teams.
5) Custom theme and templates: User-defined themes and starter templates for code generation.
6) More AI model options: Allow switching between Gemini and other LLMs or local models where feasible.

VII. CONCLUSION

This project demonstrates that combining a modern web stack (Next.js, React, Convex) with a large language model (Google Gemini) can deliver a functional, user-friendly SaaS platform for generating full React applications from natural language. Nova AI integrates prompt enhancement, multi-turn chat, live code editing and preview, and HL/LL design explanation in a single flow, with workspace persistence and token-based usage. The system is deployable on Vercel and supports both quick demo access and signed-in users with history. Limitations such as API dependency and framework scope can be addressed in future work through multi-framework support, version control, and richer code analysis. In summary, this research shows that a coherent, scalable AI-powered code generation product is feasible and can lower the barrier to turning ideas into runnable applications.

REFERENCES

[1] X. Purushottam Kumar, S. Sekhar, and R. Singh (2025). AI Content Generator SaaS Product Using Next. JS and LLM. International Journal of Scientific Research and Technology, 2(6), 52–60. https://www.ijsrtjournal.com/article/AI+Content+Generator+SaaS+Product+Using+Next+JS+and+LLM
[2] Brown, T., Mann, B., Ryder, N., et al. (2020). Language Models are Few-Shot Learners. arXiv preprint arXiv:2005.14165. https://arxiv.org/abs/2005.14165
[3] Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is All You Need. arXiv preprint arXiv:1706.03762. https://arxiv.org/abs/1706.03762
[4] Chen, M., Tworek, J., Jun, H., et al. (2021). Evaluating Large Language Models Trained on Code. arXiv preprint arXiv:2107.03374. https://arxiv.org/abs/2107.03374
[5] Google AI. (2024). Gemini API. https://ai.google.dev/
[6] Convex. (2024). Convex – Backend for your product. https://www.convex.dev/
[7] Vercel. (2024). Next.js by Vercel. https://nextjs.org/
[8] Meta. (2024). React – A JavaScript library for building user interfaces. https://react.dev/
[9] Sandpack. (2024). Sandpack – In-browser bundler. https://sandpack.codesandbox.io/
[10] Tailwind Labs. (2024). Tailwind CSS. https://tailwindcss.com/
[11] Raychev, V., Vechev, M., & Yahav, E. (2014). Code completion with statistical language models. In Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). https://dl.acm.org/doi/10.1145/2594291.2594321



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY