

AI Code Review System

Adil Abid Jamadar¹, Shakila Siddavatam²

¹Master's Student, Department of Computer Science, Abeda Inamdar Senior college, India

²Head of Department, Department of Computer Science, Abeda Inamdar Senior college, India

Abstract:

The world of software development moves fast, and keeping code clean and reliable is more important than ever. Manual code reviews help, but they're slow, often inconsistent, and depend too much on the reviewer's experience. That's where our AI Code Review System (ACRS) comes in. It uses advanced AI and machine learning to automatically analyze code, using large language models trained on massive datasets. These models don't just spot simple errors—they actually understand how code is written, how it's structured, and how it should work.

So what does this system actually do for developers? It scans through code and gives instant feedback: points out mistakes, suggests ways to make things run smoother, and explains its reasoning. This makes the code easier to read, maintain, and upgrade. On top of that, it generates an overall "code quality score" based on how correct, efficient, and clean the code is.

The best part is, the interface is designed to be simple and helpful. Developers don't have to rely on lengthy manual reviews, which means projects move faster.

The research also digs into how AI code reviews stack up against classic static analysis tools. Basically, AI systems have an edge when it comes to understanding context and offering smarter recommendations. The ACRS isn't just for veteran coders, either—it's a teaching tool for students and beginners, too. All in all, this system is a big leap forward for automated, intelligent, and scalable code reviews in today's fast-paced software industry.

Keywords: Artificial Intelligence, Code Review, Machine Learning, Code Quality, Bug Detection, Automated Feedback, Programming, Software Development

1. Introduction

Building reliable, secure, and scalable software matters more than ever, but these days, writing high-quality code is anything but simple. Software keeps getting more complex, and with big teams working on the same codebase, it's tough to keep things consistent and readable. If code quality slips, you get bugs, security problems, mounting maintenance, and slower systems.

That's where code review comes in. Teams around the world ask developers to comb through each other's code, spot mistakes, improve the design, and make sure everyone's following the same standards. Along the way, people learn from each other and stay on the same page. But let's be honest—manual code reviews take time, eat up resources, and depend a lot on who's available and how experienced they are. Big projects can end up bottlenecked, and sometimes, tricky logic or hidden inefficiencies slip through anyway.

So teams started using automated tools like static analyzers and linters. They catch basic errors and flag rule-breaking code, but they miss the deeper logic and context. You don't get meaningful feedback or smart suggestions—just a checklist of violations.

Recently, AI turned things around. Cutting-edge machine learning and large language models now “read” huge datasets of code. These models spot patterns people sometimes miss, catch logic errors, and suggest real improvements based on context. AI suddenly looks like the future of code review—faster, smarter, and much more helpful.

This research introduces an AI Code Review System (ACRS)—an automated tool built on AI techniques. It checks code for errors, points out inefficiencies, and gives developers actionable feedback. Plus, it generates a code quality score, weighing readability, correctness, and how closely code follows best practices. Put together, these features mean developers spend less time manually reviewing code and end up with better results.

Here's how the rest of the paper unfolds: Section II walks through related work, Section III breaks down our methodology, Section IV gets into the design and implementation details, Section V shares results and analysis, and Section VI wraps up with conclusions and ideas for the future.

1.1.Problem Statement:

These days, keeping code quality high is absolutely essential — you can't get reliable, secure, or high-performing software without it. Code reviews are supposed to help with that, catching bugs and making sure everyone follows best practices. But honestly, the way people usually do code reviews is pretty inefficient, especially when projects grow and the code just keeps piling up.

Manual reviews eat up a ton of time and can wear people out. Plus, if you've got a busy team or not everyone's an expert, mistakes slip through the cracks. Sometimes you just miss things, like weird logic errors, slow algorithms, or security holes that automated tools don't catch. The experience and availability of reviewers make a big difference, but you can't always count on them.

Static analysis tools try to fill the gap, but they're not exactly game-changers. Most of them stick to checking syntax and simple rule violations, and they don't really get the bigger picture — they can't figure out what the code's supposed to do, or spot tricky bugs hiding in plain sight. They rarely give solid advice for actually making the code better and can totally miss nuanced errors.

Now, software is only getting more complicated, so we really need something smarter and faster — something beyond just manual reviews and basic tools. The real challenge here is to build an automated, intelligent, and context-aware system for code review. We need it to spot mistakes, judge code quality, and give useful feedback, and do all this without being inconsistent or wasting developers' time.

That's exactly what the AI Code Review System (ACRS) sets out to do. It's designed to use artificial intelligence to make the review process smarter, faster, and more reliable. With this approach, developers could spend less time slogging through manual checks and end up with better software, all around.

1.2.Secondary Data:

Secondary data is essential for building and evaluating the AI Code Review System (ACRS). In this work, secondary data means anything we didn't create ourselves—stuff like existing datasets, published research, technical documentation, and code that's already out there for anyone to access.

The main sources? Think big open-source code repositories like GitHub. These sites have a huge variety of code, touching all sorts of programming languages and problem domains. They're packed with examples—some clean and polished, others full of mistakes. That variety teaches the system what solid programming looks like, helps it spot common errors, and even highlights inefficient solutions. The data doesn't end there, though; code samples from competitive programming sites and educational platforms give the system more to work with. That way, it learns to evaluate different styles and approaches people use when solving problems.

Published research, technical papers, and documentation for AI models are also part of the mix. Articles on things like machine learning, deep learning, and automated code analysis point out what methods are already working in the field. When it comes to model design, official docs for models such as CodeBERT or DeepSeek guide the system's architecture and boost its analytical abilities.

And then you have the official coding guidelines and standards—like those for Python, Java, or web development. These sources give the system a solid reference point for what good code should look like. By comparing submitted code to these recommended practices, ACRS can call out what needs fixing and suggest clear improvements.

Pulling all these secondary sources together means the AI doesn't just analyze code in a vacuum. It understands and gives feedback rooted in real-world development. This approach makes the system more reliable and scalable, and helps it stay effective as the tech landscape changes.

1.3.Proposed Solution:

Traditional code reviews and static analysis tools have their limits—it's time for something smarter. This research lays out an AI Code Review System (ACRS) that uses artificial intelligence and machine learning to make code reviews quicker, clearer, and way less of a hassle. The idea is simple: give developers real-time, smart, and consistent feedback, which means better code and less manual work.

Here's how it's set up. The system breaks down into a few main parts: there's a frontend interface, a backend processor, an AI-driven analysis module, and a database to keep track of everything. The frontend is built with current web tech, so developers can just upload or paste their code right in. Once they do, the code moves to the backend via secure APIs.

The backend, built on a fast and reliable framework, runs the show. It picks up the code, then hands it over to the AI module. That's where things get interesting—the AI model, trained on massive datasets of real-world code, really understands programming. It gets syntax, structure, logic—you name it. It finds mistakes, spots inefficient bits, and flags when the code doesn't follow best practices.

After crunching through the code, the system sends back a detailed report. You see every issue it finds, suggestions to clean things up, and explanations for why something's a problem. On top of that, it gives your code a score—grading for how readable, correct, efficient, and clean your code is overall. You get a quick snapshot of how you're doing.

All the submitted code, analysis, and feedback get stored in the database. This way, developers can track their growth, spot patterns, and dig into past reviews. The system isn't locked in, either—it's built to scale, ready to add new programming languages, connect with IDEs, and even tackle security checks as extra features.

In short, by weaving AI into automated code analysis, the ACRS brings accuracy and intelligence to reviews, cutting down busy work and pushing software quality—and productivity—a step ahead.

2. Literature Review

Code review sits at the heart of software engineering. It's how teams catch bugs, sharpen their designs, and keep their codebases in decent shape. The old-school way, you probably know, is all manual — developers poring over each other's code, trying to spot mistakes and making sure everyone follows the same standards. Research backs that up: manual reviews find bugs and lift the overall quality. Problem is, it eats up time, demands skilled people, and, as you'd expect, the results can swing depending on who's reviewing.

To lighten the load, developers rely on tools like static analyzers and linters. These handle basic stuff — flagging syntax errors or style issues. They're handy for surface-level checks, but let's be honest, they barely scratch the surface. They just can't grasp why the code does what it does or spot those sneaky logical flaws and clunky algorithms.

Lately, people are excited about AI and machine learning for code analysis. Models like CodeBERT dive into code and natural language, so they do a lot more than just check rules. Since they're built on mountains of data, they can find patterns, sniff out bugs, and even suggest fixes. Deep learning adds more muscle: neural networks chew through past code changes and come up with smarter feedback. Plus, there's the graph-based approach — turning programs into structured graphs that map out how everything connects. This works wonders for finding complex logical problems and understanding the code as a whole.

Now, large language models — the kind powering modern AI coding assistants — have really stepped things up. They generate code, break down logic, and give suggestions that actually make sense within context. Unlike traditional tools, these models see both the nuts-and-bolts syntax and the bigger picture of what the code's supposed to accomplish.

When researchers pit old-school static analysis against AI-driven systems, the verdict's pretty clear. The AI tools dig deeper, catch subtler issues, and offer suggestions that help with readability and maintenance. They're not just faster — they bring better accuracy and smarter feedback.

So, the takeaway? AI-based code review systems outpace traditional methods by a mile. They save time, pick up mistakes humans might miss, and help teams write cleaner, more reliable code. That's exactly why the proposed AI Code Review System (ACRS) aims to harness these advances, making code review easier and raising the game for software quality.

Methodology (Development Process)

3.1. Design of Research:

This research is all about building and testing an AI-powered solution to tackle the hassle of slow, tedious manual code reviews. Instead of just talking theory, the focus is on actually making a real, working prototype — the AI Code Review System.

First, I dig into what's wrong with manual code reviews and why traditional static analysis tools don't really cut it. Once the problem's clear, the plan shifts to creating an AI-driven tool that can take over some of that manual grunt work. The design breaks the system into clear parts: frontend, backend, the AI model itself, and a database to keep things organized.

When it comes to development, the approach stays modern—think solid web frameworks for the interface, backend APIs to keep things running smoothly, and a smart AI model at the center for analyzing code, spotting errors, and wIt shows off how AI can step in to make code reviews faster, smarter, and a whole lot more effective

3.2. Technical Research:

When it comes to the AI Code Review System (ACRS), the technical research dives deep into how modern tech, automated code analysis, and AI all fit together. The real mission here is to figure out how you can mix different tools, frameworks, and AI models to build a smart, effective code review system that actually works.

First, the team started by looking at old-school code review methods and classic static analysis tools. Those are good for catching typos and obvious mistakes, but honestly, they don't really “get” the logic behind what someone's trying to do. This gap pushed everyone to check out newer stuff, like machine learning and cutting-edge AI.

They really looked into those AI-based models, especially large language models trained on real programming data. These things can read code, notice patterns, and spot logical mistakes. Deep learning techniques like transformers piqued a lot of interest—how could they be used not just to analyze code, but to make smart suggestions?

Building out the system meant bringing modern web tech into the mix, too. The frontend uses a web framework so users have a slick spot to submit their code. On the backend, there's a high-performance API framework that crunches the data and chats with the AI model. All those inputs, analysis results, and user feedback get stored in a database, so nothing's lost.

Stitching all these pieces together is pretty critical. APIs connect the frontend, backend, and the AI model so the code flows smoothly. The AI takes the submitted code and spits back errors, suggestions, and a quality score right away—no fuss.

Testing and debugging? That's baked in from the start. They threw all sorts of code at the system to check how well it holds up and how accurate it is.

At the end of the day, this research shows that stacking AI, web tech, and good data management can put traditional code review methods to shame. The new approach makes the whole process faster and sharper, and let's face it, it's way more user-friendly.

3.3. Architecture of the System:

It's a pretty straightforward setup. Users jump in and submit their code through the frontend. Once that happens, the backend grabs their request and gets to work. It hands the code off to the AI model, which spots mistakes, suggests fixes, and comes up with a score. After that, everything gets saved in the database and pushed right back to the frontend, so users can see their results.

4. Design And Implementation:

4.1. System Architecture:

Frontend (Client side): The interface runs on React.js. It's simple—users just upload or paste their code, hit submit, then see the results. Everything happens over APIs that connect to the backend.

Backend (Server Side): The backend runs on Python FastAPI. It listens for incoming requests, then routes the code samples to the AI model for analysis.

AI Model: Here's the heart of it—this part reviews the code, spots errors, suggests improvements, and assigns a quality score.

Database (MySQL): All the basics get stored here: user information, code snippets, and the results from the analysis.

API Integration: APIs keep everything stitched together—frontend, backend, and AI model—all working smoothly so data moves where it needs to go.

System Workflow: It goes like this: users submit their code, the backend does some processing, the AI model analyzes, and then the final results pop up for users to see.

System Workflow:

1. The user uploads or pastes the code into the system..
2. The backend receives the code and sends it to the AI model.
3. The AI model analyzes the code and detects errors and improvements.
4. A code quality score and suggestions are generated.
5. The results are displayed to the user.

• System Architecture Diagram:

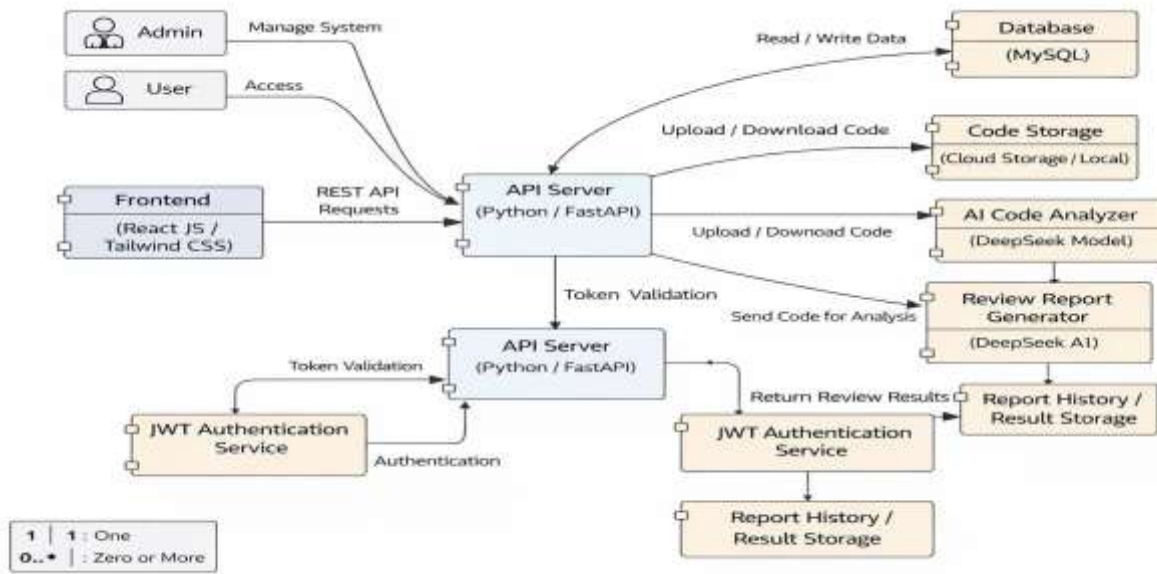


Figure 1: System Architecture of AI Code Review System

4.2. Technology Used:

This system leans on up-to-date web tech and AI to speed up code review and make it smarter. The frontend uses React.js, so everything feels responsive and slick. On the backend, FastAPI takes care of all the communication. MySQL keeps track of user info and results. The AI model jumps in to scan code and suggest improvements. REST APIs help all the pieces talk to each other and move data around smoothly.

Table 1: Technology Stack for AI Code Review System

Component	Technology Used
Frontend	React.js, Tailwind CSS, HTML, JavaScript
Backend	Python, FastAPI
Database	MySQL
AI Model	DeepSeek / LLM Model
API Communication	REST APIs
Authentication	JWT (JSON Web Token)
Development Tools	VS Code, GitHub

4.3. User Interface (UI) Screenshots

The AI Code Review System looks sharp and modern, keeping things simple and easy for everyone—developers, reviewers, admins. No clutter, just what you need, clear and straightforward. It works smoothly on desktops, tablets, and phones, so you can hop in wherever you are without losing any features.

The login screen isn't just there to look nice—it's built for security. You've got the standard email and password fields, backed by session authentication, so only the right people get in. It uses a dark theme with gradients—think clean, enterprise-style apps, but readable and pleasant to use.

Moving around the system feels natural. You don't have to dig to find things like code submission, review tracking, analytics, or user management. They're all organized on dashboards you can get to quickly, designed to make your work faster, not harder.

The system actually changes the interface depending on who's logged in:

- **Home Page:** When you land here, you see what the platform's about and find quick options to sign up or log in.
- **Admin Login:** Admins have their own secure login for all the big-picture controls.
- **Admin Dashboard:** Here, admins keep tabs on everything—monitor activity, manage users, oversee code reviews, maintain the whole system.
- **User Registration (Developers/Reviewers):** New users just fill in their info (name, email, all the basics) and set up an account.
- **User Login:** One login screen for both reviewers and developers, as you'd expect.
- **Code Submission Module:** Upload your code or just paste it in; the AI takes it from there and starts reviewing.
- **AI Review Dashboard:** Here's where you see what the AI found—bugs, quality scores, suggestions for improvement, all laid out clearly.
- **Reviewer Panel:** Human reviewers jump in, check the AI's feedback, add their own thoughts, and approve or suggest fixes.
- **User Profile:** Gives you a snapshot of your past activity, code submitted, what's been reviewed, plus options to manage your profile.

5. UI Screenshots:

The following figures show the AI Code Review System

Figure 1:Home Page:

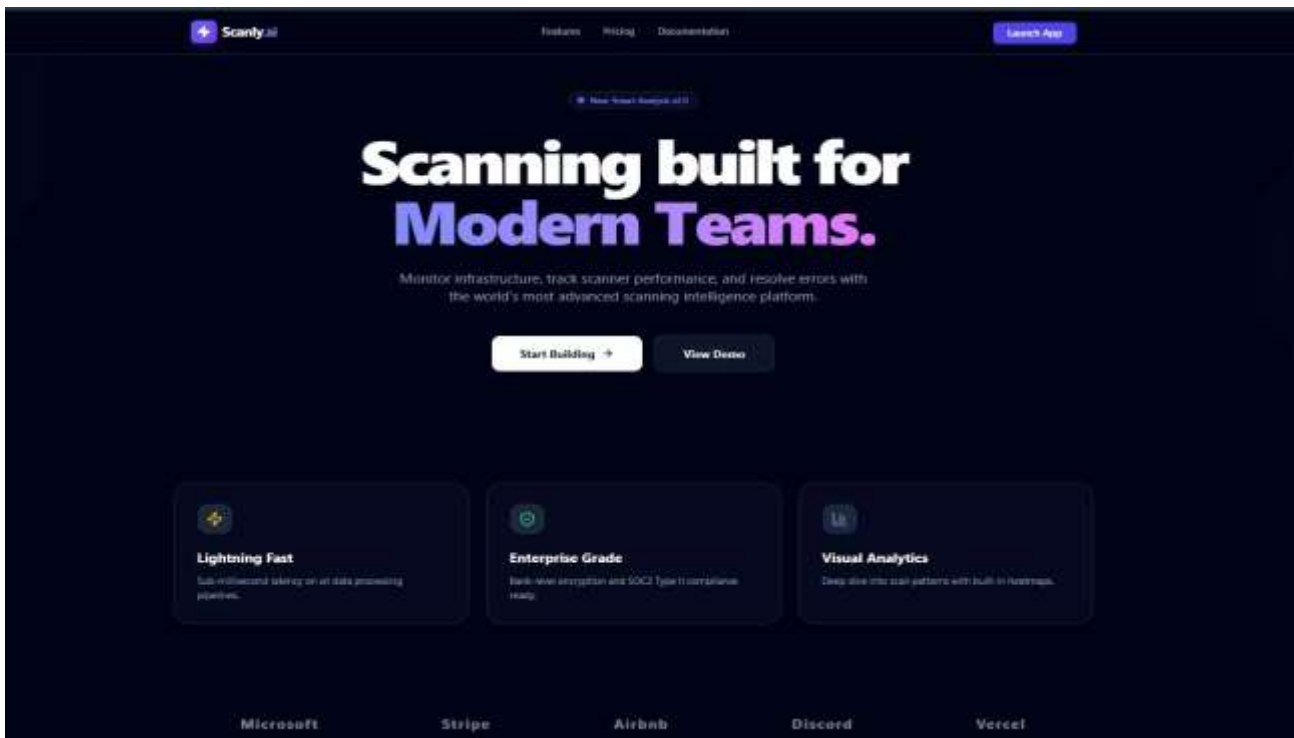


Figure 2: Sign-in Page

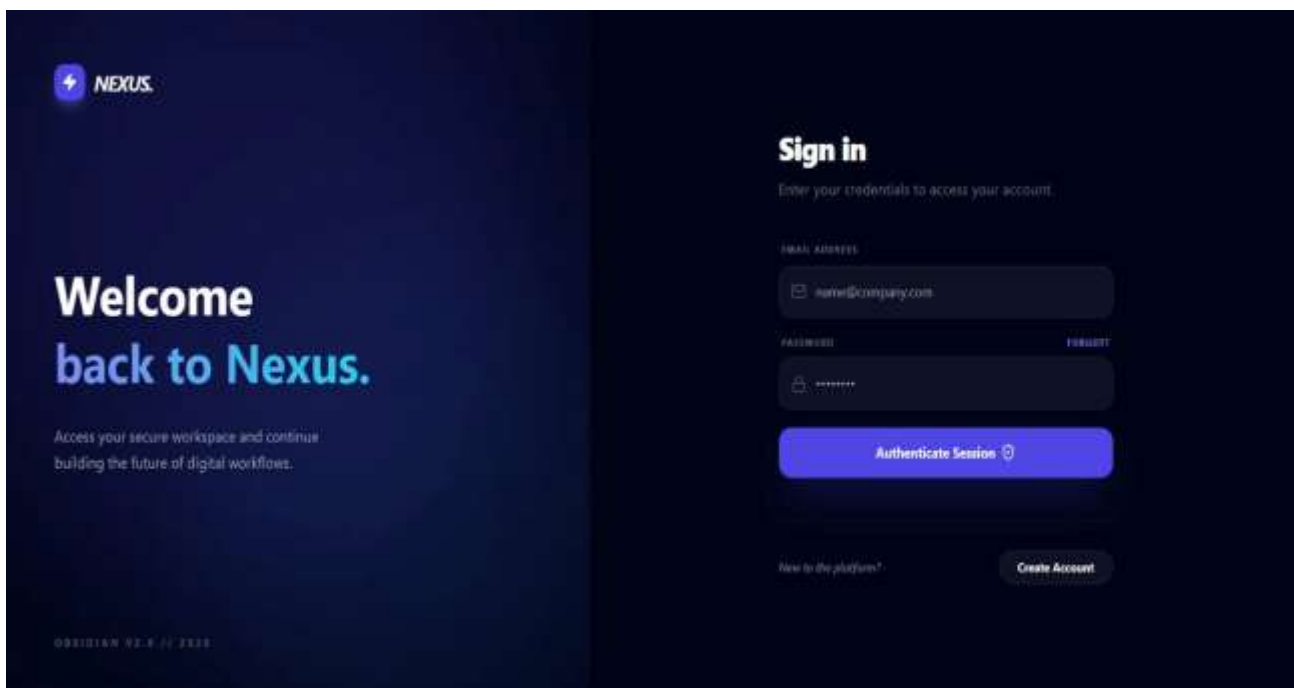


Figure 3: Create Account Page

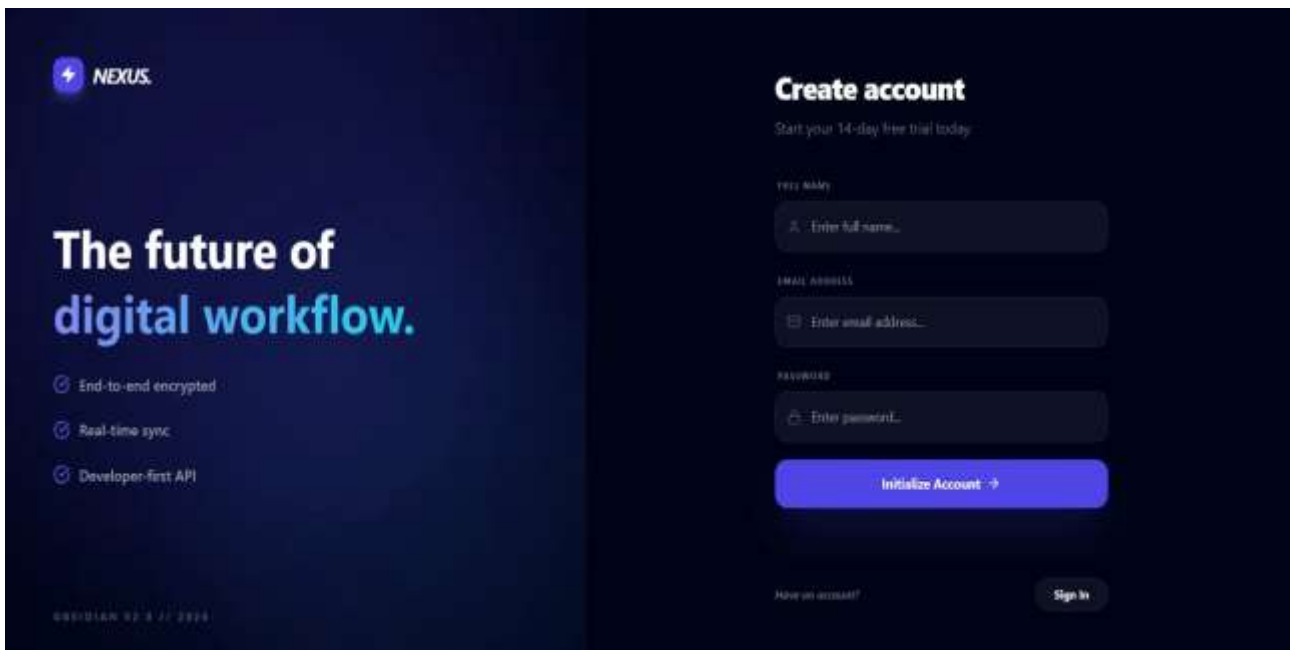


Figure 4: Dashboard Page

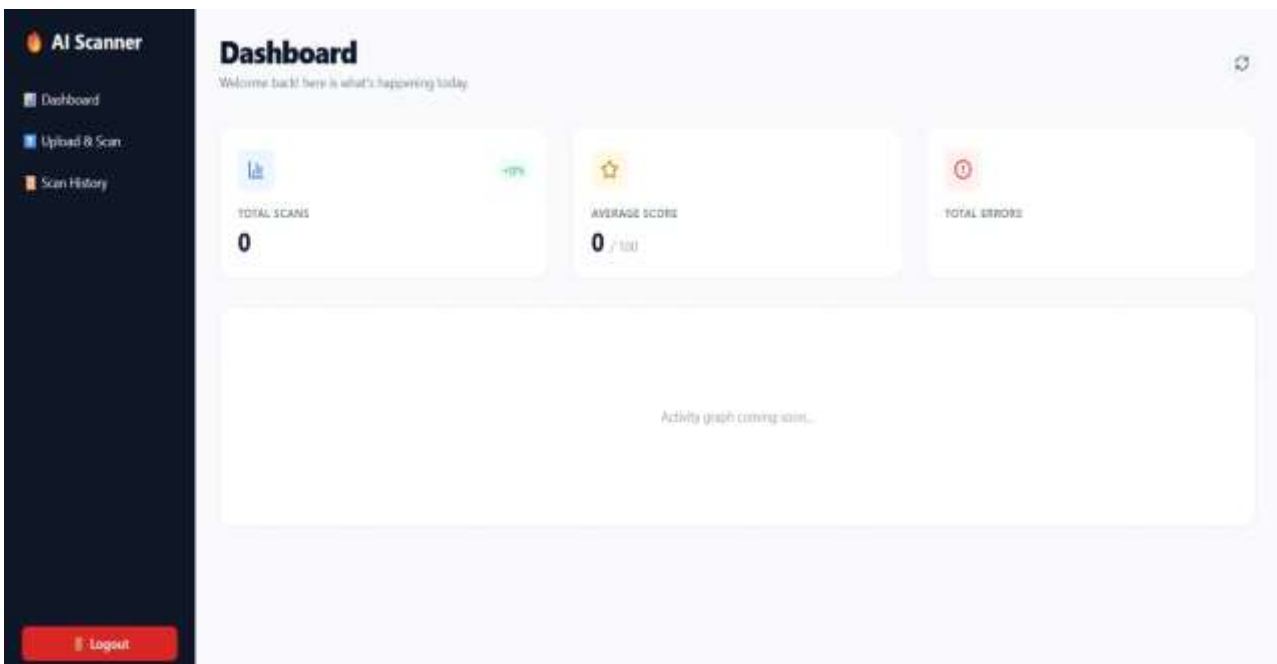
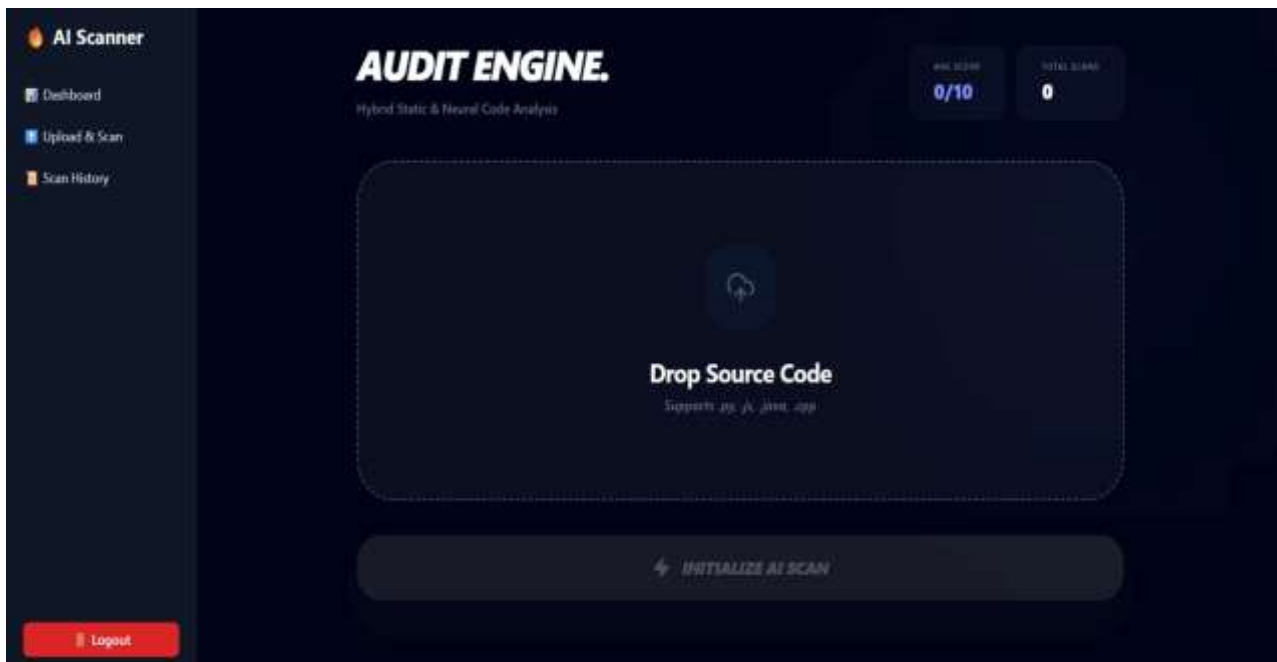


Figure 5: Audit Engine



6. Discussion

6.1. Strengths of the System

- **Automated Code Analysis:** AI takes over the heavy lifting of code analysis, so developers spend less time hunting for errors and more time building features.
- **Improved Code Quality:** The system offers smart suggestions, spots bugs, and gives tips for optimization. All of this means cleaner, better code.
- **Faster Review Process:** It flags issues right away, which keeps the review cycle moving fast and helps teams stay productive.
- **Consistency in Reviews:** By following set standards and removing human bias, it makes code reviews fair and uniform every time.
- **Real-Time Feedback:** Developers get instant advice and guidance, so they can tackle problems without waiting.
- **Scalability:** No matter how much code comes in, the system handles it without slowing down.
- **User-Friendly Interface:** The interface is simple and straightforward, so everyone—developers, reviewers, or admins—can jump in and use it without a struggle.
- **Reliability:** Users can trust the system thanks to its steady performance and accurate code analysis.
- **Integration Capability:** It connects easily with version control tools and CI/CD pipelines, keeping everything running smoothly.

6.2. Challenges and Limitations

- AI doesn't always get business logic. Sometimes it glosses over the details, so its suggestions can miss the mark.
- There's the issue of false alarms. The system might call out good code as a problem or overlook real bugs. It still needs a person to double-check.
- Getting started isn't easy. Training the AI and fitting it into the system takes time and real technical know-how.
- If your codebase is huge, expect a slowdown. Analyzing lots of code can bog down processing and hit system performance.
- Results depend on training data. If you feed the AI limited or narrow examples, its reviews lose accuracy.
- Security's a big deal. When AI touches sensitive source code, you need strong protections so nothing slips out to the wrong hands.
- Trust is earned, not given. Developers often need time before they really rely on AI suggestions during their work.
- Making everything fit together can get tricky. Connecting the AI with existing dev tools, repositories, and CI/CD setups isn't always straightforward.

6.3. Future Scope:

Here's where this system can go next. Imagine it handling several programming languages, making it useful for a lot more people. Plugging it into IDEs like VS Code could give developers instant feedback as they code. If AI models get smarter, they'll spot tougher bugs and security problems nobody wants. A mobile app version could let folks review code anywhere, anytime. Adding analytics is a game changer—it'll help users see their progress and sharpen their skills as they go.

7. References:

1. Feng, Z., Guo, D., Tang, D., and others. "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," EMNLP, 2020.
2. Chen, M., Tworek, J., Jun, H., and others. "Evaluating Large Language Models Trained on Code," arXiv preprint, 2021.
3. Allamanis, M., Barr, E. T., Devanbu, P., and Sutton, C. "A Survey of Machine Learning for Big Code and Naturalness," ACM Computing Surveys, 2018.
4. Tufano, M., Watson, C., Bavota, G., and others. "Learning Bug-Fixing Patches using Neural Machine Translation," ACM Transactions on Software Engineering, 2019.
5. Bird, C., Rigby, P., Barr, E., and others. "The Promises and Perils of Mining GitHub," IEEE MSR Conference, 2013.
6. Johnson, B., Song, Y., Murphy-Hill, E., and Bowdidge, R. "Why Developers Don't Use Static Analysis Tools," ICSE, 2013.
7. Vaswani, A., Shazeer, N., Parmar, N., and others. "Attention Is All You Need," NeurIPS, 2017.
8. DeepSeek AI. "DeepSeek-Coder: AI Model for Code Intelligence," Technical Report, 2024.