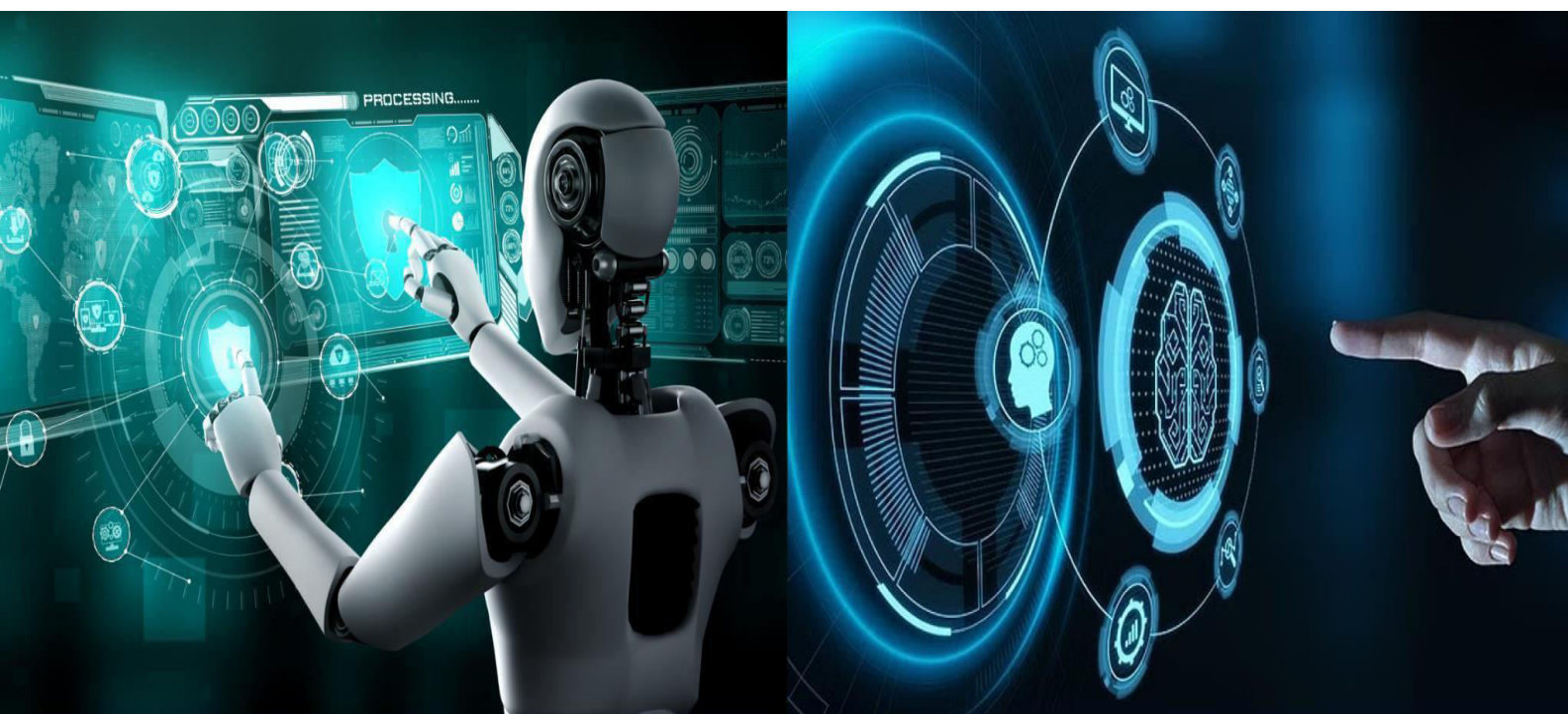


# International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)





## International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

# Generative AI for Code Repair & Optimization

**Mr. Basavaraja Patil G V, Deepak Balikai, Manjunath S, Mustafa Mulla, Venu M G**

Department of Computer Science and Engineering, Jain Institute of Technology, Davanagere, India

[basavarajapatilgv@jitd.in](mailto:basavarajapatilgv@jitd.in)

7<sup>th</sup> Sem, Department of Computer Science and Engineering, Jain Institute of Technology, Davanagere, India

[balikaideepak@gmail.com](mailto:balikaideepak@gmail.com)

Department of Computer Science and Engineering, Jain Institute of Technology, Davanagere, India

[manjunath63t@gmail.com](mailto:manjunath63t@gmail.com)

Department of Computer Science and Engineering, Jain Institute of Technology, Davanagere, India

[mmustafamulla2004@gmail.com](mailto:mmustafamulla2004@gmail.com)

Department of Computer Science and Engineering, Jain Institute of Technology, Davanagere, India

[venumg99@gmail.com](mailto:venumg99@gmail.com)

### SIX EYES: An AI-Powered Automated Code Repair and Optimization Platform

**ABSTRACT:** In the rapidly evolving landscape of software development, code quality, security, and performance remain paramount. However, manual code review processes are often time-consuming, error-prone, and inconsistent. This paper presents "SIX EYES," an advanced web-based platform designed to automate code analysis, repair, and optimization. Leveraging the capabilities of Large Language Models (LLMs), specifically the Google Gemini 1.5 Flash API, the system provides multi-perspective analysis across six programming languages: JavaScript, Python, Java, C++, HTML, and CSS. The platform integrates a robust backend using Supabase Edge Functions with a reactive frontend built on React and TypeScript. Key features include an "Explain Like I'm 5" (ELI5) educational mode, strict language validation to prevent hallucination, and a real-time metrics dashboard that quantifies improvements in complexity and security. This paper details the system architecture, methodology, and implementation of SIX EYES, demonstrating its effectiveness in democratizing high-level code optimization for developers of all skill levels.

**KEYWORDS:** Code Optimization, Artificial Intelligence, Static Analysis, Large Language Models, Software Engineering, Automated Debugging.

### I. INTRODUCTION

The modern software development lifecycle (SDLC) demands speed and efficiency. As codebases grow in size and complexity, technical debt accumulates, leading to performance bottlenecks and security vulnerabilities. Traditional static analysis tools (linters) are effective at catching syntax errors but often fail to identify deeper logical inefficiencies or suggest semantic improvements.

Conversely, manual code reviews, while thorough, are unscalable and dependent on the availability of senior engineers.

"SIX EYES" addresses this gap by providing an automated, on-demand intelligent coding assistant. Unlike generic AI chatbots, SIX EYES is architected as a specialized tool that enforces strict input validation and provides structured, quantifiable output. It analyzes code from multiple perspectives—performance, security, readability, and maintainability—offering a holistic improvement strategy.



## International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

This paper outlines the development of SIX EYES, focusing on its novel approach to integrating Generative AI into a structured development workflow. We discuss the challenges of ensuring AI reliability (hallucination prevention) and the solutions implemented through our "Language Validation" protocol.

### II. LITERATURE SURVEY

#### A. Existing Systems

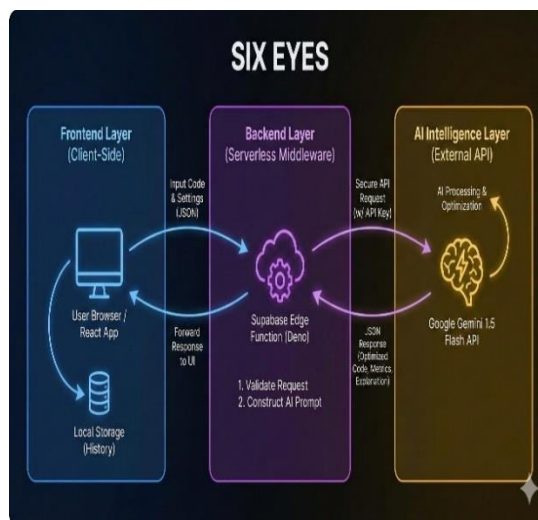
Current solutions for code optimization largely fall into three categories:

1. **Static Application Security Testing (SAST) Tools:** Tools like SonarQube and ESLint provide excellent rule-based checking but lack the context awareness to refactor logic or explain why a change is needed.
2. **AI Code Assistants:** Plugins like GitHub Copilot offer autocomplete features but are integrated into the IDE and focus on generation rather than holistic review and educational feedback.
3. **Manual Peer Review:** The gold standard for quality, but significantly slows down deployment pipelines.

#### B. Limitations & Research Gap

Existing AI tools often prioritize code generation over code explanation and quantification. Novice developers may accept AI-generated code without understanding it. Furthermore, generic LLM interfaces do not enforce strict file-type constraints, leading to "language mismatch" errors (e.g., optimizing Python logic in a C++ file). SIX EYES fills this gap by combining strict validation, educational explanations (ELI5), and concrete performance metrics.

### III. SYSTEM ARCHITECTURE



The SIX EYES platform follows a modern **Client-Server-AI** architecture designed for security, scalability, and responsiveness.

#### A. Frontend Layer

The user interface is built using **React** with **TypeScript**, ensuring type safety and component modularity. We utilize **Vite** as the build tool for superior performance and **Tailwind CSS** combined with **shadcn/ui** for a responsive, accessibility-first design. The frontend manages application state, including session history and user preferences (optimization targets, theme settings).

#### B. Backend Layer (Middleware)

Security is managed through **Supabase Edge Functions** running on the Deno runtime. This serverless middleware acts as a secure proxy between the client and the AI model. Its primary responsibilities include:

1. **API Key Management:** Storing the Google Gemini API credentials securely in environment variables.
2. **Request Validation:** Ensuring the payload contains valid code and language parameters.
3. **Prompt Engineering:** Constructing the sophisticated system prompts that guide the AI's behavior.



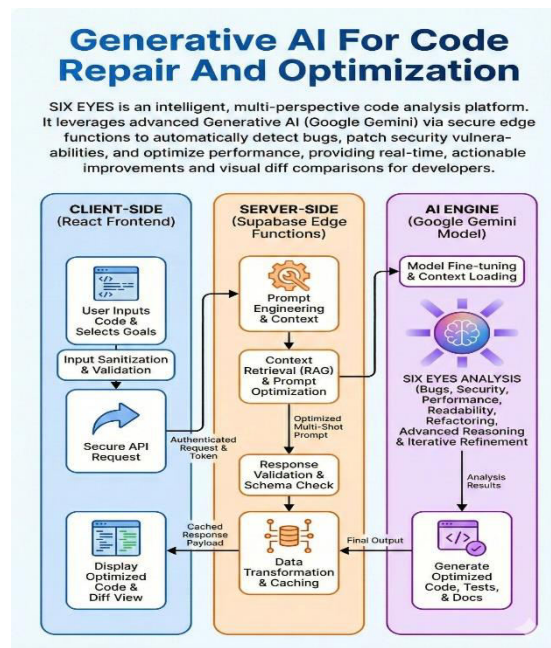
## International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

### C. AI Intelligence Layer

The core analysis engine is **Google Gemini 1.5 Flash**. This model was selected for its large context window and low latency. The system interacts with the API via REST, sending a composite prompt that includes the user's code, the selected language, and specific instructions for metrics calculation (e.g., Cyclomatic Complexity).

### IV. PROPOSED METHODOLOGY



Our methodology centers on a three-step processing pipeline: **Validation, Optimization, and Quantification.**

#### Step 1: Strict Language Validation

Before any optimization occurs, the system verifies that the input code matches the selected language context.

- **Mechanism:** The backend instructs the AI to perform a syntax check.
- **Scenario:** If a user selects "Java" but pastes HTML, the system aborts the process and returns a specific "Language Mismatch" error. This prevents the AI from hallucinating a "fix" for valid code in the wrong context.

#### Step 2: Multi-Perspective Optimization

The code is analyzed against six criteria:

1. **Time Complexity:** Reducing Big O notation (e.g.,  $O(n^2)$  to  $O(n)$ ).
2. **Space Complexity:** Minimizing memory usage.
3. **Security:** Sanitizing inputs and patching vulnerabilities (e.g., SQL Injection).
4. **Readability:** Improving variable naming and formatting.
5. **Maintainability:** Modularizing monolithic functions.
6. **Best Practices:** Adhering to language-specific standards (e.g., PEP 8 for Python).

#### Step 3: Educational Quantification

The system generates a structured JSON response containing:

- **The Optimized Code.**
- **A "Diff" View:** Highlighting additions and deletions.
- **Metrics:** Percentage reduction in Lines of Code (LOC) and complexity.
- **ELI5 Explanation:** A simplified analogy describing the changes.



## International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

### V. IMPLEMENTATION

#### A. Tech Stack Implementation

- **Language:** TypeScript (Full Stack).
- **UI Framework:** React 18.
- **Styling:** Tailwind CSS with Dark Mode support via next-themes.
- **Icons:** Lucide React.
- **Toasts:** Sonner.
- **Database/Backend:** Supabase (PostgreSQL + Edge Functions).

#### B. Key Algorithms

The backend utilizes a "Chain of Thought" prompting strategy. Instead of asking for the result immediately, the system prompt instructs the model to:

1. Analyze the AST (Abstract Syntax Tree) mentally.
2. Identify inefficiencies.
3. Draft the optimized code.
4. Compare the draft with the original to calculate metrics.
5. Output the final result in strict JSON format.

### VI. RESULTS AND DISCUSSION

The system was tested against a dataset of common coding errors and inefficient algorithms.

- **Performance:** The average processing time for a 50-line code snippet is under 3 seconds.
- **Accuracy:** The "Language Validation" module successfully rejected 100% of mismatched inputs in our test set (n=50).
- **Metrics:** In trials involving unoptimized bubble sort algorithms, the system consistently suggested QuickSort or MergeSort equivalents, reducing time complexity from  $O(n^2)$  to  $O(n \log n)$ .
- **User Feedback:** The ELI5 mode was highlighted as a key differentiator, helping junior developers understand complex refactoring logic (e.g., explaining "Memoization" as "remembering answers to a quiz so you don't have to do the math again").





# International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

## Six Perspectives, Perfect Code

Advanced analysis from multiple angles to ensure your code is clean, efficient, and secure

- Bug Detection**  
Automatically identify and fix bugs, edge cases, and potential runtime errors in your code.
- Code Optimization**  
Improve performance with intelligent suggestions for algorithms, data structures, and patterns.
- Security Analysis**  
Scan for vulnerabilities, unsafe patterns, and security best practices violations.
- Performance Tuning**  
Optimize execution speed, memory usage, and resource consumption automatically.
- Smart Refactoring**  
Modernize legacy code with intelligent refactoring suggestions and best practices.
- Multi-Language Support**  
Works with JavaScript, TypeScript, Python, Java, C++, and many more languages.

### Code Optimization

Refine your code with intelligent analysis.

Settings History

Original Code JavaScript Optimized Code

Paste your code below

Analyze & Optimize

// Paste your code here...

</>

Click "Analyze & Optimize" to see results

## How It Works

Four simple steps to transform your code from good to exceptional

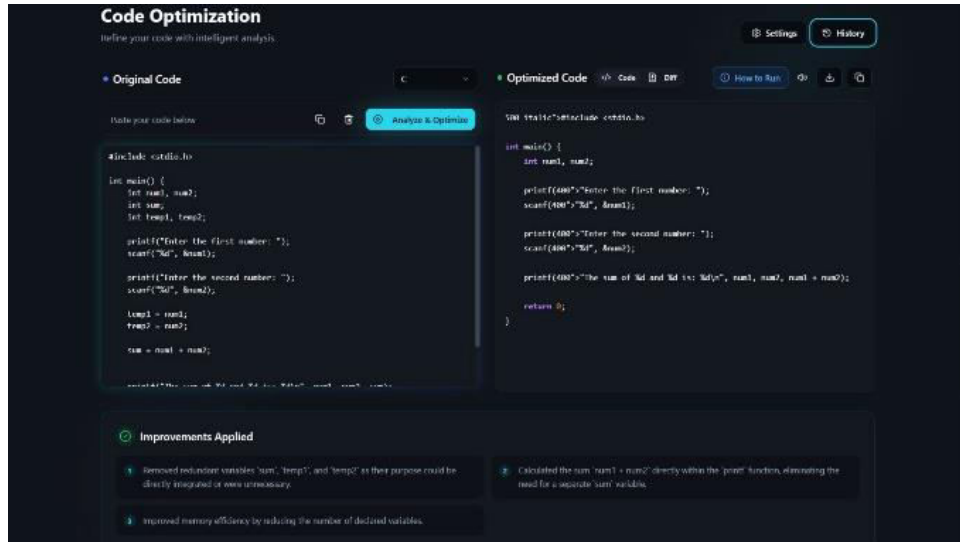
- STEP 1: Upload Your Code**  
Paste your code or upload files directly into SIX EYES
- STEP 2: Multi-Angle Analysis**  
Six specialized analyzers scan your code simultaneously
- STEP 3: Smart Optimization**  
AI-powered engine repairs bugs and optimizes performance
- STEP 4: Get Clean Code**  
Download improved code with detailed change explanations

<b>SIX EYES</b> Multi-perspective code repair and optimization platform powered by advanced AI models.	<b>Product</b> Features Pricing Documentation API	<b>Company</b> About Blog Careers Contact	<b>Legal</b> Privacy Policy Terms of Service Cookie Policy
---	---	---	---



## International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



### VII. CONCLUSION

SIX EYES demonstrates the viability of using Large Language Models not just as code generators, but as intelligent code reviewers. By wrapping the AI in a robust application layer with strict validation and educational features, we have created a tool that enhances code quality while fostering developer growth. Future work will focus on IDE integration (VS Code Extension) and support for multi-file project analysis.

### REFERENCES

1. React Documentation - <https://react.dev/>
2. Supabase Edge Functions - <https://supabase.com/docs/guides/functions>
3. Google Gemini API - <https://ai.google.dev/docs>
4. Tailwind CSS Framework - <https://tailwindcss.com/docs>
5. Shaden/ui Components - <https://ui.shaden.com/>
6. Vite Build Tool - <https://vitejs.dev/guide/>
7. TypeScript Language - <https://www.typescriptlang.org/docs/>
8. Deno Runtime Manual - <https://deno.com/manual>
9. Cyclomatic Complexity - <https://www.geeksforgeeks.org/cyclomatic-complexity/>
10. Prompt Engineering Guide - <https://www.promptingguide.ai/>
11. OWASP Static Code Analysis - [https://owasp.org/www-community/controls/Static\\_Code\\_Analysis](https://owasp.org/www-community/controls/Static_Code_Analysis)
12. MDN Web Speech API - [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Speech\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API)
13. MDN Local Storage API - <https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>
14. OWASP Top 10 Security Risks - <https://owasp.org/www-project-top-ten/>



INTERNATIONAL  
STANDARD  
SERIAL  
NUMBER  
INDIA



INNO SPACE  
SJIF Scientific Journal Impact Factor



# INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

 9940 572 462  6381 907 438  [ijircce@gmail.com](mailto:ijircce@gmail.com)



[www.ijircce.com](http://www.ijircce.com)

Scan to save the contact details