DevFlow: An AI-Powered Online IDE with Intelligent Code Debugging and Optimization

1st Prof. Anita Dombale
Department of Computer Science with Artificial Intelligence
Vishwakarma Institute of Technology
Pune, India
Sangita.jaybhaye@vit.edu

2nd Aparna Nimishakavi
Department of Computer Science with Artificial Intelligence
Vishwakarma Institute of Technology
Pune, India
Aparna.nimishakavi23@vit.edu

3rd Avinash Birajdar
Department of Computer Science with Artificial Intelligence
Vishwakarma Institute of Technology
Pune, India
Avinash.birajdar23@vit.edu

4th Ayush Chavan
Department of Computer Science with Artificial Intelligence
Vishwakarma Institute of Technology
Pune, India
Ayush.Chavan23@vit.edu

5th Atharva Bhakre
Department of Computer Science with Artificial Intelligence
Vishwakarma Institute of Technology
Pune, India
Atharva.bhakre23@vit.edu

***Abstract*—Modern software development requires efficient coding environments with intelligent assistance capabilities. This paper presents DevFlow, an AI-powered online integrated development environment that combines real-time code execution with advanced debugging and optimization features. DevFlow leverages Google Gemini 2.5 Flash AI model integrated with Retrieval-Augmented Generation and ChromaDB vector database to provide context-aware debugging assistance. The system supports multiple programming languages including JavaScript, Python, C, C++, and Java, offering seamless code execution through a microservices architecture. Our novel approach integrates three key components: a robust code execution backend, an AI-powered debugging service using RAG technology, and an intelligent code optimizer that analyzes algorithmic complexity and suggests optimizations. Performance evaluation demonstrates that DevFlow reduces debugging time by providing accurate error analysis and corrective suggestions, while the optimizer successfully identifies and resolves complexity bottlenecks from $O(n^2)$ to $O(n)$ in tested scenarios. The system achieves an average response time of under 3 seconds for AI-assisted debugging and optimization tasks.**

***Index Terms*—online IDE, artificial intelligence, code optimization, debugging assistance, retrieval-augmented generation, microservices architecture, compiler integration**

I. Introduction

The landscape of software development has evolved significantly with the advent of cloud-based integrated development environments (IDEs) and artificial intelligence. Traditional IDEs require local installation, configuration, and maintenance, creating barriers for beginners and limiting accessibility. Online IDEs address these concerns by providing browser-based development environments, but most lack intelligent assistance for debugging and code optimization.

Recent advances in large language models (LLMs) and retrieval-augmented generation (RAG) have opened new possibilities for intelligent code assistance. However, integrating these technologies into a cohesive, production-ready IDE presents several challenges: maintaining low latency, providing accurate context-aware suggestions, and supporting multiple programming languages within a unified architecture.

This paper introduces DevFlow, a comprehensive online IDE that addresses these challenges through three main contributions:

- A microservices architecture that separates code execution, AI debugging, and optimization into independent, scalable services
- An AI-powered debugging system utilizing RAG with ChromaDB vector database to provide context-aware error analysis and corrective code suggestions
- An intelligent code optimizer that performs algorithmic complexity analysis and generates optimized implementations using Google Gemini 2.5 Flash

The system supports JavaScript, Python, C, C++, and Java, providing real-time code execution with comprehensive error handling. Our evaluation demonstrates that DevFlow significantly reduces debugging time while maintaining high accuracy in error identification and resolution suggestions.

II. Related Work

A. Online IDEs and Code Execution Platforms

Several online IDEs exist, including Replit, CodeSandbox, and JDoodle. These platforms provide browser-based code execution but typically lack advanced AI-powered debugging and optimization features. Replit offers collaborative features but limited intelligent code assistance. JDoodle focuses on simple code execution without contextual debugging support.

B. AI-Assisted Development Tools

GitHub Copilot and Amazon CodeWhisperer provide code completion using large language models. However, these tools primarily focus on code generation rather than debugging and optimization of existing code. They lack the retrieval-augmented generation approach that provides context-specific debugging assistance based on documented error patterns and solutions.

III. System Architecture

DevFlow employs a microservices architecture comprising four main components as illustrated in Fig. **??**. This design ensures scalability, maintainability, and independent deployment of services.

A. Frontend Layer

The frontend is built using Next.js 15 with React 19, providing a modern, responsive user interface. Key features include:

- Monaco Editor integration for syntax highlighting and code editing
- Resizable panels for file explorer, editor, and terminal views
- Tab management for multiple file editing
- Real-time terminal output display

The frontend communicates with backend services through RESTful APIs, maintaining separation of concerns and enabling independent scaling.

B. Code Execution Service

The execution backend runs on Node.js with Express, handling code compilation and execution for five programming languages. The service architecture includes:

- Isolated execution environment using temporary file system
- Language-specific compiler/interpreter invocation (GCC for C/C++, Python3, Node.js, Java)
- Rate limiting to prevent abuse
- Automatic cleanup of temporary files

Each code execution request generates a unique identifier, creates temporary files, compiles (if necessary), executes the code, captures output, and performs cleanup—all within 3-5 seconds for typical programs.

C. AI Debugging Service (RAG)

The RAG service integrates Google Gemini 2.5 Flash with ChromaDB vector database to provide intelligent debugging assistance. The architecture consists of:

- **Document Seeding**: Error patterns, solutions, and best practices are embedded into ChromaDB using sentence transformers
- **Retrieval Phase**: When debugging is requested, the service queries ChromaDB with error logs and code context to retrieve relevant documentation (TOP_K=3)
- **Generation Phase**: Retrieved context is combined with the code and error logs in a structured prompt sent to Gemini API
- **Response Structuring**: AI-generated response includes root cause analysis, explanation, corrective steps, and corrected code

This RAG approach significantly improves accuracy compared to zero-shot LLM queries by grounding responses in documented error patterns.

D. Code Optimization Service

The optimizer service analyzes code for algorithmic inefficiencies and generates optimized implementations. The process follows these steps:

1) **Language Detection**: Automatically identifies programming language from file extension
2) **Complexity Analysis**: Uses Gemini 2.5 Flash to analyze time and space complexity
3) **Algorithm Identification**: Detects inefficient patterns (nested loops, redundant operations)
4) **Optimization Generation**: Produces optimized code with improved algorithmic complexity
5) **Structured Output**: Returns JSON containing original complexity, optimized complexity, analysis, optimized code, and improvement summary

The optimizer successfully identifies common anti-patterns such as $O(n^2)$ bubble sort and suggests $O(n \log n)$ alternatives or built-in language optimizations.

IV. Implementation Details

A. Technology Stack

DevFlow utilizes modern web technologies and AI frameworks:

- **Frontend**: Next.js 15, React 19, TypeScript, Monaco Editor, Tailwind CSS
- **Backend Services**: Node.js, Express.js (ESM modules)
- **AI Integration**: Google Generative AI SDK (Gemini 2.5 Flash)
- **Vector Database**: ChromaDB 1.8.1 with sentence transformers
- **Compilers/Interpreters**: GCC, G++, Python3, Node.js, Java JDK

B. RAG Pipeline Implementation

The RAG debugging service implements a two-phase approach. During initialization, documentation is processed and stored:

1) Error documentation is chunked into semantic segments
2) Each chunk is embedded using sentence transformers
3) Embeddings are stored in ChromaDB with metadata

At query time, the system retrieves relevant context and generates responses:

1) User code and error logs are embedded
2) ChromaDB performs similarity search (TOP_K=3)
3) Retrieved documents provide context to Gemini

4) Structured prompt ensures consistent JSON output format

This approach reduces hallucination and improves response accuracy by grounding AI responses in verified documentation.

C. Security and Error Handling

DevFlow implements multiple security measures:

- Rate limiting (10,000 requests per 15 minutes per IP)
- Temporary file isolation with UUID-based naming
- Automatic cleanup of execution artifacts
- CORS configuration for cross-origin requests
- Environment variable management for API keys

Error handling is comprehensive, capturing compilation errors, runtime exceptions, and API failures. The system distinguishes between successful output and error messages using keyword detection ("error:", "exception") to provide appropriate user feedback.

V. Evaluation and Results

A. Experimental Setup

We evaluated DevFlow using 50 deliberately buggy code samples across five programming languages (10 per language). Each sample contained common errors such as syntax mistakes, logic errors, undefined variables, and algorithmic inefficiencies. Performance metrics included:

- Debugging accuracy: Percentage of correctly identified root causes
- Response time: Average latency for AI assistance
- Optimization effectiveness: Complexity improvement rate
- User satisfaction: Qualitative feedback from 20 developers

B. Debugging Performance

The RAG-powered debugging service achieved 94% accuracy in identifying root causes across test cases. Table I shows accuracy breakdown by error type.

Average response time was 2.8 seconds, including ChromaDB retrieval (0.4s), Gemini API call (2.2s), and response formatting (0.2s). This performance meets real-time interaction requirements for developer workflows.

C. Code Optimization Results

The optimizer successfully identified complexity issues in 46 out of 50 test cases (92%). Common optimizations included:

- Bubble sort ($O(n^2)$) to built-in sort ($O(n \log n)$)
- Nested loops to hash map lookups ($O(n^2)$ to $O(n)$)
- Redundant calculations moved outside loops

Table II presents complexity improvements achieved across different algorithmic patterns.

D. Comparison with Existing Solutions

Table III compares DevFlow with existing online IDEs and AI coding assistants. DevFlow uniquely combines real-time execution, RAG-powered debugging, and algorithmic optimization in a single platform.

TABLE I
Debugging Accuracy by Error Type

Error Type	Test Cases	Accuracy
Syntax Errors	15	100%
Logic Errors	12	91.7%
Runtime Errors	13	92.3%
Undefined Variables	10	90%
Overall	**50**	**94%**

TABLE II
Code Optimization Results

Pattern	Original	Optimized
Sorting Algorithms	$O(n^2)$	$O(n \log n)$
Nested Loop Search	$O(n^2)$	$O(n)$
Redundant Calculations	$O(n^2)$	$O(n)$
Array Operations	$O(n^2)$	$O(n)$

VI. Discussion and Future Work

DevFlow demonstrates that combining microservices architecture with AI-powered assistance creates a powerful development environment. The RAG approach significantly improves debugging accuracy compared to zero-shot LLM queries by grounding responses in documented patterns.

However, several areas warrant future investigation:

- **Expanded Language Support**: Adding support for additional languages such as Rust, Go, and Swift
- **Collaborative Features**: Real-time code sharing and pair programming capabilities
- **Performance Optimization**: Caching frequent queries and implementing edge computing for reduced latency
- **Enhanced Security**: Sandboxed execution environments and resource limits
- **Learning System**: Adaptive RAG that learns from user interactions and feedback

The modular architecture facilitates these enhancements without requiring fundamental system redesign.

VII. Conclusion

This paper presented DevFlow, an AI-powered online IDE that integrates real-time code execution with intelligent debugging and optimization services. By leveraging retrieval-augmented generation with ChromaDB and Google Gemini 2.5 Flash, DevFlow achieves 94% debugging accuracy and successfully optimizes algorithmic complexity in 92% of test cases. The microservices architecture ensures scalability and maintainability while supporting five programming languages.

TABLE III
Feature Comparison with Existing Platforms

Feature	DevFlow	Replit	JDoodle	Copilot
Multi-language	✓	✓	✓	✓
Real-time Execution	✓	✓	✓	-
RAG Debugging	✓	-	-	-
Code Optimization	✓	-	-	Partial
Complexity Analysis	✓	-	-	-

DevFlow demonstrates that combining traditional compiler technology with modern AI capabilities creates synergistic benefits for developers. The system reduces debugging time through accurate error analysis and provides actionable optimization suggestions, making it a valuable tool for both learning and professional development.

Future work will focus on expanding language support, implementing collaborative features, and enhancing the RAG system with adaptive learning capabilities. The open architecture enables continuous improvement and integration of emerging AI technologies.

ACKNOWLEDGMENT

We thank the open-source community for providing the foundational technologies that made this project possible, including Next.js, Express.js, ChromaDB, and Google Generative AI.

REFERENCES

Please number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

REFERENCES

[1] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Advances in Neural Information Processing Systems, 2020, pp. 9459–9474.

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

[3] Google, "Gemini: A Family of Highly Capable Multimodal Models," Technical Report, Google DeepMind, 2023.

[4] ChromaDB, "The AI-native open-source embedding database," https://www.trychroma.com/, accessed Dec. 2024.

[5] GitHub Copilot, "Your AI pair programmer," https://github.com/features/copilot, accessed Dec. 2024.

[6] Replit, "The collaborative browser based IDE," https://replit.com/, accessed Dec. 2024.

[7] A. Vaswani et al., "Attention is All You Need," in Advances in Neural Information Processing Systems, 2017, pp. 5998–6008.

[8] J. Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proceedings of NAACL-HLT, 2019, pp. 4171–4186.

[9] T. Brown et al., "Language Models are Few-Shot Learners," in Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 1877–1901.

[10] S. Iyer et al., "OPT-IML: Scaling Language Model Instruction Meta Learning through the Lens of Generalization," arXiv preprint arXiv:2212.12017, 2022.

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

[12] Next.js Documentation, "The React Framework for Production," https://nextjs.org/docs, accessed Dec. 2024.