The Journal of Open Source Software

Code Explainer: A Multi-Method AI-Powered Code Analysis Tool

Piyush Yadav [1]

1 Department of Computer Science and Engineering, Institute of Engineering and Management

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 15 October 2025
Published: unpublished

Summary

Code Explainer is an open-source tool that generates natural language explanations of source code in Python, JavaScript, Java, and C++. It offers two complementary analysis methods: rule-based parsing for fast structural analysis, and neural language models for semantic understanding. This dual approach addresses a common challenge in software development and education—understanding unfamiliar code quickly and accurately. The tool provides a web interface for interactive use and a REST API for programmatic access, making it accessible to developers, students, educators, and researchers.

Statement of Need

Understanding existing code consumes significant developer time, particularly when reviewing unfamiliar codebases, onboarding new team members, or maintaining legacy systems. Students learning programming also struggle to comprehend code examples without clear explanations. While proprietary AI tools like GitHub Copilot exist, they lack transparency and require subscriptions. Open-source alternatives typically offer only static analysis without semantic understanding, or require machine learning expertise to deploy.

Code Explainer fills this gap by providing an easy-to-install, well-documented system that combines both fast rule-based analysis and advanced neural models. **Developers** benefit from rapid code comprehension during reviews and maintenance. **Students** and **educators** gain an interactive learning tool that explains code in natural language. **Researchers** can use it as a baseline for program comprehension studies or to evaluate new code-to-text models. The tool's dual methodology allows users to choose between speed (rule-based) and depth (neural) depending on their needs.

Key Features

Code Explainer provides:

- **Dual Analysis Methods**: Rule-based parsing using language-specific AST parsers (Python's `ast`, JavaScript's Acorn, JavaParser, Clang for C++) for fast structural analysis, and neural models (CodeBERT (Feng et al., 2020), CodeGen (Nijkamp et al., 2022), Gemini API) for semantic understanding.
- **Multi-Language Support**: Analyzes Python, JavaScript, Java, and C++ with language-specific pattern recognition.
- **Web Interface**: React-based UI with syntax-highlighted code editor (CodeMirror), method/model selection, and formatted explanation display.
- **REST API**: Flask backend with `/explain/` endpoint for programmatic access.

- **Performance**: Rule-based analysis completes in under 0.5 seconds per snippet for fast structural insights, while neural models provide richer, context-aware explanations with processing times of 1-2 seconds.

The tool allows users to choose between speed and depth depending on their needs.

Installation and Usage

Code Explainer requires Python 3.8+ and Node.js 14+. Installation is straightforward:

```
git clone https://github.com/piyushrajyadav/code-explainer.git
cd code-explainer

# Backend
cd backend && pip install -r requirements.txt && python run.py

# Frontend (new terminal)
cd frontend && npm install && npm start
```

Access the web interface at `http://localhost:3000` or use the REST API:

```
import requests

response = requests.post('http://localhost:8000/explain/', json={
    'code': 'def fibonacci(n): return n if n <= 1 else fibonacci(n-1) + fibonacci(n-2)',
    'language': 'python',
    'analysis_method': 'nlp'
})
print(response.json()['explanation'])
```

Detailed documentation is available in the repository's README.

Research and Educational Applications

Code Explainer can be used in education for teaching programming concepts, in software engineering for code review and onboarding, and in research on program comprehension and code-to-text generation.

Community and Contribution

The project welcomes contributions including bug reports, feature requests, documentation improvements, and new language analyzers. Detailed guidelines are in `CONTRIBUTING.md`. The software is actively maintained and licensed under MIT.

Acknowledgements

This work builds on the Hugging Face Transformers library (Wolf et al., 2020), PyTorch (Paszke et al., 2019), and pre-trained models from the research community.

References

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020). CodeBERT: A pre-trained model for programming and

natural languages. *Findings of the Association for Computational Linguistics: EMNLP 2020*, 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., Savarese, S., & Xiong, C. (2022). A conversational paradigm for program synthesis. *arXiv Preprint arXiv:2203.13474*. https://doi.org/10.48550/arXiv.2203.13474

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., Platen, P. von, Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., … Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. https://doi.org/10.18653/v1/2020.emnlp-demos.6