International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 2, Mar-Apr-2026, ISSN (Online): 2395-566X

# CodeFox: A Modular Platform for Repository Insights

**Utsav R. Hirapra**
Dept. of Information Technology
Parul Institute of Engineering and Technology Parul University, Vadodara, Gujarat, India.

**Abstract- As the scope of software projects increases, it becomes increasingly difficult to ensure proper code quality and conduct efficient code reviews, not because the required tools lack, but because all relevant information tends to become buried under unnecessary noise. In an attempt to combat that issue, CodeFox is presented as a lightweight, modular solution for discovering valuable insights by leveraging modern AI algorithms. In terms of functionality, CodeFox consists of a developer-friendly user interface, various automated review tools, and a meaning- based code search module. The platform collects metadata such as commits, reviews, comments, as well as ownership information and uses semantic embeddings to index critical elements within the code. By leveraging vector search algorithm, CodeFox allows its users to easily discover similar code, discussion threads related to the code, as well as reviewers who were working on this code. The use of the platform at an early stage of development in our company allowed us to shorten the review cycle process as well as reveal the reasoning behind code changes more efficiently. In this paper, we discuss CodeFox architecture, key aspects of integration, namely webhooks, background workers, and persistent storage, as well as share our experience of implementing a convenient yet lightweight platform to facilitate further development. We plan to conduct a user study in order to evaluate efficiency in the context of the problem and implement more advanced data gathering features and intelligent suggestions in the future.**

**Keywords- Repository insights, Code review automation, Vector search, Embeddings, Developer productivity, Next.js, Pinecone, Prisma, Inngest.**

## I. INTRODUCTION

With software development evolving to become much more collaborative and data-intensive, the problem of having access to a lot more information also becomes very practical. Commits, pull requests, review comments, informal discussions in issue trackers, or chat tools — all of those pieces of communication become trivial to generate, but not necessarily easy to look up or correlate with one another. It is common practice to spend hours looking into the history to learn the rationale behind a particular decision or to determine an appropriate reviewer, which negatively affects productivity and makes knowledge loss more probable.

CodeFox has been developed in order to help address such everyday friction points. Rather than creating just another new feature in the developer tool belt, the idea of CodeFox was to consolidate repository signals in a centralized, lightweight platform that allows one to efficiently find useful context. The platform aggregates all relevant historical information, augments it with semantic embeddings, and enables users to query the context via a dashboard and search functionality.

Among other things, the platform provides contextual search by meaning (rather than by file name), suggests possible reviewers, summarizes discussions, and allows for efficient background indexing that keeps the UI performant. This paper covers the motivation behind the application design, implementation details, and performance measurement ideas.

## II. LITERATURE REVIEW

Previous work on software engineering and developer tools focused on solutions which help engineers discover relevant data from code repository. Historically, lexical and structural searches for code were used in open source tools like OpenGrok and ripgrep. Though they were able to provide literal results in code, they weren't capable of discovering semantically relevant code snippets. This problem was addressed by recent developments in embeddings and machine learning approaches such as CodeBERT [1] and Code2Vec [4].

Code change reviews are costly due to two problems. First, it's necessary to find suitable code reviewers. Second, it is required to search for past discussions concerning the code. Recent research into reviewer recommender systems [3] showed that these approaches are useful for developers. Nonetheless, they are concerned mostly with recommending reviewers, rather than with discovering related discussions.

NLP algorithms can be applied to software engineering use cases such as automated summarization of pull requests, extraction of intent from commit messages, and thread clustering. The approaches prove that the combination of code and discussion content leads to better search/recommendation quality, but are rarely packaged and made accessible for smaller teams.

Technological progress makes it possible to perform semantic lookups using embeddings in production-quality systems such as Faiss, Annoy, and Pinecone [5]. The background processing in frameworks such as Inngest [6] takes off-the-path heavy computations and improves overall UX.

Despite existing achievements, there are still several gaps to fill. Most of the systems analyze code or discussion separately and only few are capable of including review discussions into the semantic search process. There aren't many efficient solutions for smaller teams who need less complex tools for managing their repositories. CodeFox claims to solve these problems.

## III. MATERIALS AND METHODS

The approach is mainly concerned with the design and development of the CodeFox system.

### A. Tech Stack and Data Sources

- Next.js with TypeScript utilized to render client pages and UI elements.
- The application's backend consists of Node.js server routes and API handlers, as well as Inngest to run background workflows and asynchronous jobs.
- We will use Prisma ORMs with Postgres to store structured data and to migrate it.
- Pinecone is a hosted search index used for "Approximate Nearest Neighbour" Search.

### B. Design and Implementation Steps

- **Solicitation of Requirements:** Colleagues provided some recommendations on an informal level regarding the necessary features. This includes searching based on context, suggested reviewers, and short summaries of previous conversations.
- **Modeling the System:** Make sketches about high-level architecture. Also, sequence diagrams show that data flows from webhooks to results returned after a search.
- **Data Entered into the System:** Webhooks are tasked with recording events from the repository. The event handler is to normalize the payload for the persistence via prisma. Then, the handler pushes a job in a queue for further processing (ibid).
- **Enrichment of Data:** Artifacts like diffs, PR descriptions/notes, and comments are retrieved by background processes. These are in turn embedded, and the embeddings inserted into Pinecone. Jobs are idempotent and rate-limited.
- **Indexing and Retrieval:** There is synchronization of the PostgreSQL database and the Pinecone Vector database. The databases are linked with each other. The data processing flow uses lightweight-filtering; This is then followed by the use of an Approximate Nearest Neighbor (ANN) for conducting a semantic search with the returned data.
- **Recommended Reviewers:** Hybrid heuristics include historical records of file ownership, collaborative editing graphs, and embedding similarity to rank a sample of reviewers. The approach is needed to recommend on who should be reviewing a change.
- **User Interface and User Experience:** The frontend is then capable of retrieving the data based on ranked order and providing the information in context with reference (links) to the original artifacts. Any of the heavy operations are performed asynchronously.

### C. Reproducibility and Deployment

The code regarding infrastructure together with environment variables (URL to database, Pinecone API key, and Embedding API key) are elaborated in the README file of the repository. The application can be deployed using Vercel on the frontend, as well as a micro-instance or serverless runner for the background tasks.

There are unit tests that attest to parsing functionality, integration tests that attest to the proper functioning of ingestion, and linting via CI.

## IV. MODELING AND ANALYSIS

To understand how CodeFox is to be employed in a typical developers workflow few key modeling techniques were used. We started with use case analysis to get a general idea of what the actors involved try to achieve.

### A. Use Case Analysis

So let us explore the components of the system and how do they function together. The Use Case Diagram visualising this is presented in Fig. 1. The main actors, or the key players in this system are:

- **The developer:** they connect with GitHub, submit changes to review, execute automated checks and analyze the outputs of those checks.
- he person whose job it is to review: receives reports, reads summaries for context and adds comments on requests for changes.
- Repository Bot (Ingestion Service): watches webhooks, normalizes event data and enqueues enrichment jobs.

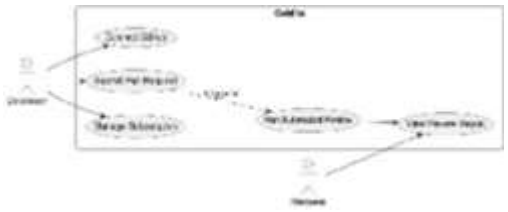

Fig. 1. Use Case Diagram of CodeFox

.

### B. Entity Relationship Diagram

Figure 2 shows the Entity-Relationship Diagram for CodeFox

- **Repository —** id, name, owner_id → User, created_at.
- **PullRequest —** id, repo_id → Repository, author_id → User, title, status, created_at.
- **Review — id**, pr_id → PullRequest, reviewer_id → User, verdict, created_at.
- **Comment —** id, review_id → Review, author_id → User, body, created_at.

We looked at how it helped us find old discussions, suggested people to review our work, and showed us relevant bits of information.

### A. Results (Informal)

- **Quicker context retrieval:** In this way, developers found it much faster to locate the relevant pull requests and conversations than when they were doing it manually by searching through history. Thanks to semantic search, one teammate said, she could "find the discussion from months ago in under a minute."
- **Reviewer recommendation result:** The hybrid ranking heuristic (ownership + co-edit signals + embedding similarity) get useful candidates. Second, the recommended reviewers were at least as well rated among the first three compared to those manually selected by the developer.

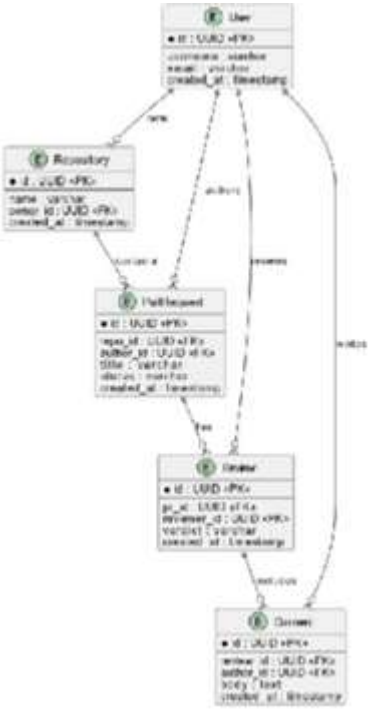

Fig. 2. Entity Relationship Diagram (ERD) of CodeFox.

**System Architecture**

The architecture of CodeFox consists of four different modular layers, as depicted in Fig.

- **Presentation Layer:** Dashboard, Search, and Review UI realized on Next.js and React framework. Returns relevant search results in the form of snippets of context with links to PRs/commits.
- **Application Layer:** The Next.js API / App router handles server routes and authentication, using JWT/OAuth as the basis. Background processing/indexing, embedding generation done by Inngest workers.
- **Data Layer:** PostgreSQL database and Prisma ORM for structured metadata, Pinecone for storing semantic embeddings. Keeps metadata and GitHub for events ingestion, OpenAI (or similar) for embeddings, Pinecone for vectors. The connectors can be replaced with another one without modifying any core logic.
- It was generally well received by the public. These summaries were particularly useful as users could refer back to the primary source if uncertain about something rather than needing to search for it. Others said that they were also able to ask less fundamental questions while reviewing, as they no longer had to stop and interrogate what was going on quite so often.

**Discussion**

That's a positive development for smaller teams. There are simple, lightweight semantic tools that ensure less time wasted going in circles with iterations and triage without additional process overhead.

And in many cases, the real value doesn't come from perfect automation—it comes from helping people quickly understand the context behind things.

Results are mostly anecdotal and biased due to early adopters influencing tool evolution. No quantitative benchmarks have been established yet, and embedding quality relies both on the choice of model and on text purity. Costs of API usage, limitations of rate limits, noise in embeddings generated for smaller pieces of code, and privacy issues associated with such solutions are known problems.

**Lessons learned:** perform expensive operations in the background, maintain vector-metadta synchronization, and carefully cleanse input data to prevent secrets leakage through embedding APIs.

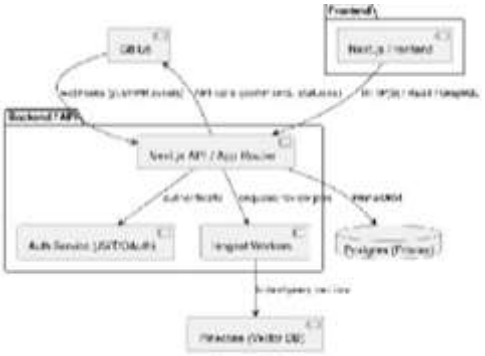

Fig. 3. System Architecture of CodeFox.

## V. RESULTS AND DISCUSSION

We tried out the CodeFox prototype in our own workplace and kept track of how it was used in a few different situations.

## VI. CONCLUSION

These outcomes look encouraging for smaller groups; light semantic tools would minimize pointless exchanges while increasing triaging velocity without high operational costs. The highest gain comes from presenting the proper context rather than building a perfectly automated solution.

The results we've seen so far are based on real-world experience, and they might be influenced by the people who started using the tool early on. Even so, the results are really encouraging. At this point, we can't share any hard numbers, but it's worth noting that getting a good embedding depends a lot on the model you use and how clean your input data is. We also discovered some potential challenges we will have to consider, including the cost of calling an API, the fact that embeddings can be noisy for small code fragments and privacy issues.

The key takeaways include (a) offloading costly computations from the request time by indexing in the background, (b) synchronising your metadata and your vectors, and (c) cleaning up the inputs to avoid leaking sensitive information to the external embedding provider.

**Future Work**

- Proposed future directions show that the systems functionality can be improved even further. Take care not to comment out (IE, include HTML comment tags) code.
- User study is a significant direction refers to controlled evaluation through triage time, precision@k for document retrieval and reviewer recommend accuracy. Future work also incorporate taking aggregation over repositories to identify commonality themes and specialized reviewers on more than one repository
- The analysis of embedding techniques, which do not involve any external API, and token deletion that prevents the leak of sensitive information will help differential privacy improve a lot. One of the prospects lies in the supervised ranking methods that is labeling reviewer data and using algorithms based on supervised learning along with the existing heuristic based algorithms.

**Acknowledgment**

We want to thank the people who helped us with CodeFox, especially those who worked on the open source libraries that made it possible. We also appreciate the feedback from our teammates, who tried out early versions and gave us ideas for what features to focus on.

## REFERENCES

1. M. A. Baxter et al., “CodeBERT: A Pre-Trained Model for Programming and Natural Languages,” arXiv:2002.08155, 2020.
2. A. Alon et al., "Code2Vec: Learning Distributed Representations of Code," Proc. ACM POPL, 2019.
3. J. Zhao, Y. Zhang, and X. Yu, “Who should review this pull request?” in Proc. ICSE, 2019, pp. 1–12.
4. CodeRabbit Inc., "CodeRabbit: AI-Powered Code Review Platform," 2024. [Online]. Available: https://coderabbit.ai
5. OpenAI, "Large Language Models for Code Understanding and Generation," 2023.