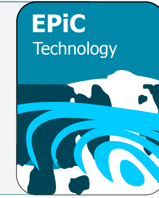
EPiC Series in Technology

Volume 3, 2026, Pages 89–92

Proceedings of AI for Accelerated Research Symposium



A Vision for Sustainable AI-Assisted Development: Semantic Retrieval for Efficient AI Code Generation

Krishiv Piduri[1*]

Henry M. Gunn High School, Palo Alto, California, United States
krishiv@krishivpiduri.com

Abstract

Recent advancements in Artificial Intelligence (AI) code generation have given rise to a practice called vibe-coding where a code-generation tool writes the code while the developer primarily provides feedback and corrections. Tools like Cursor and GitHub Copilot have popularized it by directly integrating AI code generation into the IDE, thus making it easier to use. However, users of these tools often err on the side of providing excessive content; often providing whole dependency trees and code bases despite the decreasing accuracy and increasing energy consumption. In the paper that follows, we outline a vision for a system that extracts minimal, but sufficient context for these code generation systems. We argue that current context management systems are a critical bottleneck and propose a research agenda based on semantic retrieval to address this issue. We explore open questions and future directions that could make AI-assisted development more efficient and cost-effective.

1 Introduction

The generation of AI code is rapidly changing software development. Developers increasingly rely on AI tools to generate and debug code. This new process, commonly referred to as vibe-coding, inverts the traditional coding process such that the developer becomes more like an editor while the LLM writes the code. As these Large Language Models (LLMs) become more advanced and capable, the bottleneck increasingly shifts from generation quality to context management and optimization as well as deciding what code the model should and should not see in order to respond effectively.

While RAG has transformed NLP [4], applying it to code presents unique challenges. Developers often err on the side of providing excessive context to maximize quality, but this creates two critical problems. First, the monetary and computational cost of processing redundant tokens is prohibitive. Second, effectiveness often degrades as irrelevant code dilutes the context, making it harder for models to discern pertinent information [6, 7, 2].

These challenges highlight a need for a fundamental shift in how AI tools interact with codebases. We envision a future where intelligent semantic retrieval systems provide targeted, sufficient context. This paper explores core research challenges, architectural pathways, and evaluation methodologies required to realize this vision.

[*] Conceived the research direction, designed the system architecture, conducted all experiments, and wrote the paper.

2 A Conceptual Architecture to Frame Research Challenges

Unlike recent graph-based approaches [5, 1] or simple retrieval agents [3], our vision focuses on optimizing the ”return on context” by leveraging semantic summaries to retrieve only the minimal, sufficient code chunks necessary for generation.

2.1 Overview

The following section presents a conceptual architecture created to highlight key research solutions. This model is not intended to serve as a definitive solution. Rather, this is used as a medium to discuss key research challenges and opportunities in code summarization, retrieval, and system integration.

Figure 1 illustrates the complete system architecture. The system consists of three major components. In the **Indexing Phase (offline)**, the codebase is parsed into semantically meaningful chunks and stored with metadata; this is performed once per initialization. During the **Prompt Phase (online)**, a context retrieval agent identifies relevant chunks based on the user's prompt. Finally, the **Update Phase (maintenance)** ensures that when code changes are detected, only the modified components are re-processed and updated in the database.

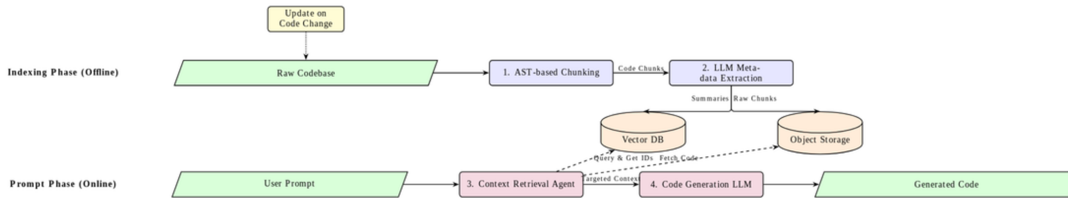


Figure 1: System overview

2.2 Indexing phase

The indexing phase processes the codebase and creates a searchable semantic index. This phase consists of three sequential steps:

2.2.1 Code chunking

The process begins by segmenting the codebase into semantically meaningful chunks. This step requires careful calibration: overly large chunks may overwhelm the backend coding agent with unnecessary context, leading to increased computational overhead; conversely, excessively small chunks may lack sufficient context for accurate interpretation. We optimize for chunks that minimize two external dependancies.

The optimal chunking strategy respects the program's existing organizational structure. Specifically, natural units like functions, classes, and modules should remain intact. We leverage the existing cAST system [8], which performs precisely this type of semantically-aware code segmentation.

2.2.2 Extracting metadata

After grouping code into semantic chunks, we generate natural language summaries to enable searchable storage. These summaries are embedded as vectors in a database, allowing natural language queries to retrieve relevant code chunks.

The quality of these summaries is paramount to system performance. This step represents a key area for ongoing research, as summary quality directly impacts retrieval accuracy.

2.2.3 Data storage

The final step stores processed data in a two-tier architecture where the raw code chunks are stored in an object store and the embeddings to the summaries of those code chunks are stored in a vector database. This separation optimizes both cost and performance: expensive vector operations are performed only on compact summaries, while bulk code storage uses cheaper object storage solutions.

2.3 Prompt phase

This phase transforms user requests into targeted code generation. First, a *context retrieval agent* parses the prompt to identify key concepts and queries the vector database for semantically similar summaries. The system then fetches the corresponding raw code from object storage and passes this targeted context to the backend coding agent for generation. The number of retrieved chunks (k) is a critical hyperparameter requiring empirical optimization.

2.4 Update phase

Production systems require real-time synchronization to ensure the backend agent operates on current code versions. We implement incremental processing where file system watchers identify modified files (*Change Detection*), triggering **selective re-processing** of only the changed sections. Finally, updated chunks replace outdated entries in the vector store to maintain **database synchronization** without the cost of full re-indexing.

This approach minimizes computational overhead while ensuring data freshness, crucial for maintaining system accuracy in active development environments.

3 Research Agenda and Open Challenges

To realize the vision of sustainable, semantic-aware AI development, several key challenges must be addressed. We propose the following research agenda:

A. Challenges in Code Summarization. A critical open question is defining "summary fidelity." How can we ensure a natural language summary captures the strict functional constraints of code (e.g., edge cases, error handling) without retaining the full token cost of the original file? Furthermore, we need quantitative metrics that measure the quality of code summaries specifically for retrieval effectiveness, rather than just human readability.

B. Challenges in Retrieval Quality. Current metrics focus on generic accuracy (Pass@k), but we lack methodologies to measure if retrieved code is truly relevant to a developer's specific intent. Future work must create benchmark datasets for evaluating code retrieval across different programming languages and paradigms to standardize these measurements.

C. System and Integration Challenges. Codebases change rapidly. A static vector store becomes a liability if the code summaries are outdated. A major research challenge is developing strategies to detect when code changes invalidate existing summaries and managing cascading updates efficiently. Additionally, we must identify technical barriers to integrating persistent embeddings directly into existing IDEs to ensure seamless adoption.

References

[1] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, et al. Graphcodebert: Pre-training code representations with data flow, 2021.

[2] J. Hartman, H. Sagtani, J. Tibshirani, and R. Mehrotra. Ai-assisted coding with cody: Lessons from context retrieval and evaluation for code recommendations. In *Proceedings of the 18th ACM Conference on Recommender Systems (RecSys '24)*, pages 748–750, October 2024.

[3] S. Jain, A. Dora, K. S. Sam, and P. Singh. Llm agents improve semantic code search. *arXiv preprint arXiv:2408.11058*, 2024.

[4] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. *CoRR*, abs/2005.11401, 2020.

[5] J. Li, X. Shi, K. Zhang, L. Li, G. Li, Z. Tao, J. Li, et al. Coderag: Supportive code retrieval on bigraph for real-world code generation. *arXiv preprint arXiv:2504.10046*, 2025.

[6] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023.

[7] J. Walczak, P. Tomalak, and A. Laskowski. Impact of code context and prompting strategies on automated unit test generation with modern general-purpose large language models. *arXiv preprint arXiv:2507.14256*, 2025.

[8] Y. Zhang, X. Zhao, Z. Z. Wang, C. Yang, J. Wei, and T. Wu. cast: Enhancing code retrieval-augmented generation with structural chunking via abstract syntax tree. *arXiv preprint arXiv:2506.15655*, 2025.