

CG-CLI: Graph-Guided Context Optimization for AI Coding Agents

Husna Ansari*, Maya Nair**

*Department of Computer Science, SIES College of Arts, Science & Commerce (Empowered Autonomous), Mumbai, India

Email: Husnafmcs2526097@ascs.sies.edu.in

**Department of Computer Science, SIES College of Arts, Science & Commerce (Empowered Autonomous), Mumbai, India

Email: mayan@sies.edu.in

Abstract:

Large language models used for software engineering often require repository context distributed across files, functions, tests, configuration, and dependency paths. Supplying broad code context can increase input overhead, whereas aggressive reduction can omit relevant evidence. This paper presents CG-CLI, a local and model-agnostic command-line framework for graph-guided context optimization in JavaScript and TypeScript repositories. The system constructs a lightweight structural code graph, identifies seed nodes from a developer question, expands through statically detected relationships, and extracts compact source slices. A controlled benchmark of 150 cases across 10 locally maintained fixture projects and 15 task categories evaluates retrieval recall and context reduction against a controlled whole-file baseline. The primary one-hop configuration achieved 66.4% average retrieval recall with 16.2% average context reduction. Two-hop expansion improved recall to 75.1% but exceeded the baseline context by 2.8%, demonstrating a recall–context trade-off. CG-CLI also includes a local visualization dashboard and an optional structured response protocol for manual response-length comparison. The work is a retrieval and context-selection layer; it does not control external AI-agent tool loops or guarantee answer correctness.

Keywords — AI coding agents, code retrieval, code graph, context optimization, repository-level software engineering, static analysis.

I. INTRODUCTION

Large language models are increasingly used for code generation, completion, debugging, explanation, and repository-level development. Unlike isolated programming tasks, repository-level questions may require information distributed across functions, files, imports, tests, configuration, and data-access modules. Retrieval-augmented code-generation research and repository-level benchmarks show that selecting relevant cross-file context is an important part of effective code assistance [1]–[3].

A common workflow supplies large files or directories to an external coding model. This may include irrelevant source code, increase input overhead, and make useful evidence harder to locate. Conversely, selecting too little code can omit callers, imports, error handling, tests, or configuration needed to answer a developer question. The central problem is therefore to balance **context efficiency** with **retrieval coverage**.

This paper presents CG-CLI, a local command-line framework for graph-guided source-context selection in JavaScript and TypeScript repositories. CG-CLI constructs a lightweight structural graph containing files, functions, methods, classes, symbols, and statically detected relationships. Given a natural-language developer question, it identifies relevant seed nodes, expands through structural links,

and extracts exact source slices for an optional external AI coding workflow.

The research question is:

Can graph-guided code retrieval reduce the amount of repository context supplied to an AI coding agent while preserving relevant information needed for diverse developer tasks?

The system evaluates retrieval recall against manually defined expected symbols and compares selected source context with a controlled whole-file baseline. A 150-case controlled benchmark evaluates seed-only, one-hop, two-hop, and adaptive graph-expansion strategies.

The contributions of this work are:

1. A local graph-guided context-selection framework for JavaScript and TypeScript repositories.
2. A controlled evaluation that measures retrieval recall, selected nodes, source-context size, and estimated context reduction.
3. A retrieval-depth ablation demonstrating the trade-off between expected-symbol coverage and context efficiency.
4. A local dashboard and optional concise response protocol for visualization and manual response-length comparison.

CG-CLI is a retrieval and context-optimization layer, not an autonomous coding agent or a replacement for an external LLM. It does not control an external model's internal tool-use loop, hidden reasoning, or answer correctness.

II. RELATED WORK

A. Repository-Level Code Retrieval

Repository-level code completion and generation require relevant context beyond the active file. RepoBench, CodeRAG-Bench, and recent survey work examine repository-aware retrieval and the role of retrieved context in code generation [1]–[3]. Other approaches investigate graph-guided, dependency-aware, semantic, and selective retrieval, including RepoGraph, RANGER, RepoHyper, GraphCoder, DraCo, GRACE, and Repoformer [4], [5], [7]–[9], [13], [22]. Related work also explores repository generation and retrieval for local projects [18], [23], [24], as well as partial-dependency retrieval [29].

B. Code Graphs and Structural Navigation

Code graphs represent relationships between program entities and can support code navigation, localization, and context selection. Existing work includes code knowledge graphs, semantic graph matching, graph databases, and programming knowledge graphs [14], [16], [17], [20], [21]. Further approaches investigate code summarization, repository-level graph generation, and graph-guided agents [6], [10]–[12], [15], [19]. Repository Intelligence Graph also explores deterministic architectural representations for code assistants [28].

CG-CLI is related to these approaches but focuses on a lightweight, local context-selection layer. It builds a structural graph from JavaScript and TypeScript repositories and evaluates how graph-expansion depth affects expected-symbol recall and selected-context size. It does not train a code model or implement an autonomous coding agent.

C. Repository-Level Evaluation and Research Gap

Repository-level evaluation extends beyond isolated functions to examine code completion, reasoning, and longer-horizon coding tasks. Existing benchmarks and studies include RepoBench, R2C2-Coder, SWE-PolyBench, RepoReason, and NL2Repo-Bench [2], [23], [25]–[27]. Research on repository intelligence and graph-enhanced retrieval further motivates examining how context is selected before it reaches a coding model [5], [11], [28], [29].

Unlike evaluations of generated answers or completed software tasks, this study measures local retrieval behavior using manually defined expected symbols and a controlled whole-file baseline. Therefore, CG-CLI's recall values indicate retrieval coverage, not answer correctness. Its contribution is an empirical comparison of retrieval depth, context size, and retrieval coverage in a lightweight local framework.

III. SYSTEM DESIGN AND METHODOLOGY

A. System Overview

CG-CLI is a local Node.js and TypeScript command-line application for indexing JavaScript and TypeScript repositories, constructing a structural graph, retrieving graph-related source slices, calculating metrics, and visualizing results through a local dashboard. The CLI remains the core research system; the dashboard provides visualization and manual evaluation support.

Given a developer question, CG-CLI indexes or loads repository structure, identifies relevant seed nodes, expands through graph relationships, extracts source slices, and produces an optimized context packet. The system separately records selected nodes, characters, local estimated tokens, retrieval recall, context reduction, and retrieval operations. The resulting packet can optionally be supplied to an external AI coding model, but CG-CLI does not call or control external model APIs.

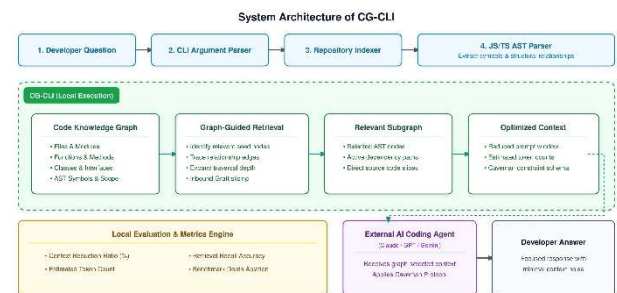


Fig. 1. Architecture of CG-CLI showing repository indexing, structural code-graph construction, graph-guided retrieval, optimized context generation, optional external model usage, and local evaluation metrics.

B. Repository Indexing

CG-CLI processes JavaScript and TypeScript source files locally. It uses source-analysis infrastructure based on Tree-sitter-compatible parsing components to identify structural entities and relationships. The indexer records file locations and source ranges so that the system can retrieve exact code slices later without duplicating complete file contents in each graph node.

The index excludes irrelevant generated or dependency artifacts where configured by the project, such as dependency directories and build output. The purpose of indexing is not to execute repository code. CG-CLI reads and parses source files for static analysis and graph construction.

C. Structural Code Graph

The structural graph stores source entities as nodes and statically detectable relationships as directed edges. The graph is intentionally lightweight and designed for retrieval rather than full semantic program verification.

TABLE I. STRUCTURAL GRAPH NODES AND EDGE TYPES

Category	Type	Description
Node	File	A JavaScript or TypeScript source file
Node	Function	A top-level function or function-like declaration
Node	Method	A class or object method where extraction is supported
Node	Class	A class declaration
Node	Symbol	A named exported, imported, or referenced symbol
Edge	CONTAINS	Connects a file or class to its contained entity
Edge	IMPORTS	Connects a file to an imported module or symbol
Edge	EXPORTS	Connects a file to an exported symbol
Edge	CALLS	Connects a caller to a statically identified callee
Edge	REFERENCES	Connects a source entity to a referenced symbol

Node identifiers follow patterns such as:

file:<path>

fn:<path>:<name>

method:<path>:<Class.method>

class:<path>:<Name>

sym:<path>:<name>

The graph stores source locations and structural links. This enables the retrieval stage to recover only the relevant lines of source code after node selection.

D. Graft Engine and Controlled Baseline

The Graft Engine is the inbound graph-guided retrieval component. It identifies seed nodes by matching question terms against indexed symbols, file paths, and source metadata. It then expands through statically detected graph relations to retrieve related code.

Four configurations are evaluated: Depth 0 retrieves seed nodes only; Depth 1 includes directly related nodes; Depth 2 includes second-hop relations; and Adaptive begins with limited expansion and applies further heuristic expansion when required. Selected nodes are converted into exact source slices with file paths, symbols, line ranges, and relationship metadata.

The selected context is compared with a controlled whole-file baseline containing complete source files associated with the selected nodes under the benchmark policy. This baseline is not equivalent to the entire repository or to an external agent's actual file-reading behavior.

E. Metrics

Retrieval recall measures the proportion of manually defined expected symbols included in the retrieved set:

$$\text{Recall} = \frac{\text{Expected symbols retrieved}}{\text{Total expected symbols}}$$

It measures retrieval coverage rather than LLM answer correctness.

Context reduction compares selected context C_s against baseline context C_b :

$$\text{Context Reduction} = \frac{C_b - C_s}{C_b} \times 100$$

A negative value indicates that selected context exceeded the controlled baseline.

CG-CLI estimates token volume locally using:

$$\text{Estimated Tokens} = \frac{\text{Characters}}{4}$$

This heuristic supports consistent relative comparison but is not an exact tokenizer count for Claude, GPT, Gemini, or another provider. The system additionally records selected-node counts and retrieval operations. These local operations must not be interpreted as actual external AI-agent tool calls.

F. Caveman Filter: Outbound Compact Response Protocol

The Caveman Filter is an optional, model-neutral concise response protocol. It instructs an external model to answer using four fields: **Answer**, **Evidence**, **Action**, and **Risk**. The protocol discourages greetings, repeated explanations, and unnecessary introductory text.

CG-CLI does not run an LLM locally or automatically rewrite a model response. Its dashboard supports manual pasting of baseline and compact responses, then measures characters, estimated tokens, word count, required-section presence, and compactness warnings. Response-length reduction does not establish semantic equivalence, technical correctness, or functional preservation; these require manual or separately validated evaluation.

IV. EXPERIMENTAL SETUP

A. Controlled Benchmark

CG-CLI was evaluated using a controlled benchmark comprising 150 cases across 10 locally maintained JavaScript/TypeScript fixture projects and 15 developer-task categories. Each case uses a manually specified set of expected symbols. The benchmark evaluates retrieval behavior and context size; it does not directly evaluate the correctness of an external LLM's answer.

The benchmark outcome was:

- Total cases: 150
- Successful cases: 145
- No-match cases: 5
- System execution errors: 0
- Fixture projects: 10
- Task categories: 15

The five no-match cases are retained in the reported benchmark status rather than removed from the evaluation. This preserves visibility into cases where the retrieval system could not identify sufficient matching evidence.

B. Task Categories

The benchmark evaluates questions from the following categories:

1. Bug Diagnosis
2. Code Navigation
3. Function Understanding

4. Class/Object Understanding
5. Dependency Tracing
6. Error Path
7. Data Flow
8. Impact Analysis
9. Test Relationship
10. Configuration
11. API/Route
12. Database/Data Access
13. Security/Validation
14. Performance Investigation
15. Feature/Change

Representative questions include:

- “How does inspect find a symbol?”
- “What functions does TaskService.rename call?”
- “Which methods call validateTaskInput?”
- “What does the db module import?”
- “What happens after validateTaskInput throws?”
- “Where is task data passed from the service to the database?”
- “Which imports are affected if I rename the task model?”
- “What production code does testCreateTask call?”
- “Where are authentication settings configured?”
- “Where are tokens or API keys verified?”

C. Retrieval Variants

Four variants were evaluated:

1. Depth 0: Seed nodes only.
2. Depth 1: Seed nodes plus one graph hop.
3. Depth 2: Seed nodes plus two graph hops.
4. Adaptive: Heuristic expansion from Depth 1 toward Depth 2 where additional context was considered necessary.

The primary variant is Depth 1 because it represents the main configuration of the benchmark and provided the strongest observed balance between recall and context reduction.

V. RESULTS AND DISCUSSION

A. Controlled Benchmark Summary

The primary Depth 1 configuration achieved 66.4% average retrieval recall and 16.2% average context reduction. It selected 7.8 nodes and 3,202 characters on average, compared with a 3,787-character controlled baseline. Using the local character-based heuristic, this corresponds to approximately 801 selected estimated tokens versus 947 baseline estimated tokens.

The result indicates that direct graph-neighbor expansion can reduce selected source context while retaining a measurable proportion of manually defined expected symbols.

However, retrieval recall should not be interpreted as answer accuracy.

TABLE II. CONTROLLED BENCHMARK SUMMARY FOR THE PRIMARY DEPTH 1 CONFIGURATION

Metric	Result
Controlled benchmark cases	150
Benchmark fixture projects	10
Task categories	15
Successful cases	145
No-match cases	5
System execution errors	0
Average retrieval recall	66.4%
Average context reduction	16.2%
Average selected nodes	7.8
Average selected context	3,202 characters
Average selected estimated tokens	approximately 801
Average baseline context	3,787 characters
Average baseline estimated tokens	approximately 947

The estimated token figures are calculated through the local characters-divided-by-four heuristic:

$$\frac{3202}{4} \approx 801$$

$$\frac{3787}{4} \approx 947$$

Thus, the primary configuration selected approximately 145 fewer estimated tokens per benchmark case on average relative to the controlled baseline. This is an estimated comparison metric, not an actual count from OpenAI, Anthropic, Google, or another provider tokenizer.

B. Retrieval-Depth Ablation

TABLE III. RETRIEVAL-DEPTH ABLATION RESULTS

Variant	Average Recall	Context Reduction	Average Selected Characters	Average Selected Nodes
Depth 0: Seed only	40.9%	7.9%	945	2.8
Depth 1: Seed + one hop	66.4%	16.2%	3,202	7.8
Depth 2: Seed + two hops	75.1%	-2.8%	4,990	12.5
Adaptive Depth 1→2	74.5%	5.8%	4,294	10.6

Depth 0 produced the smallest selected context but only 40.9% recall, showing that seed nodes alone frequently omit relevant dependencies. Depth 1 increased recall to 66.4% while producing the largest observed positive context reduction, 16.2%.

Depth 2 achieved the highest recall, 75.1%, but selected 4,990 characters on average and produced -2.8% context reduction. Therefore, the additional graph expansion increased coverage but removed the context-efficiency advantage. Adaptive expansion achieved 74.5% recall and 5.8% reduction,

indicating that the current heuristic improves coverage but requires further calibration.

Overall, Depth 1 provided the best observed trade-off under the current controlled benchmark policy. This result is not a claim that one-hop retrieval is universally optimal.

The observed trade-off motivates evaluating retrieval coverage together with context size rather than treating either metric as a sufficient indicator of retrieval quality. Related work on selective repository retrieval also identifies the usefulness of retrieved context as an important consideration [22].

C. Category-Level Findings

TABLE IV. CATEGORY-LEVEL RETRIEVAL RESULTS

Task Category	Retrieval Recall	Context Reduction
Bug Diagnosis	60.0%	13.7%
Code Navigation	60.0%	17.4%
Function Understanding	97.5%	9.9%
Class/Object Understanding	60.3%	13.4%
Dependency Tracing	93.3%	12.6%
Error Path	47.5%	18.0%
Data Flow	63.3%	-0.6%
Impact Analysis	92.0%	23.0%
Test Relationship	86.7%	24.4%
Configuration	68.3%	46.1%
API/Route	80.0%	7.2%
Database/Data Access	49.0%	12.9%
Security/Validation	63.3%	13.0%
Performance Investigation	33.3%	28.4%
Feature/Change	41.7%	9.7%

Function Understanding, Dependency Tracing, and Impact Analysis produced the highest recall values: 97.5%, 93.3%, and 92.0%, respectively. These tasks align well with statically detectable functions, imports, references, and direct relationships.

Performance Investigation, Feature/Change, Error Path, and Database/Data Access showed lower recall. Such questions may depend on indirect calls, runtime behavior, framework conventions, broader semantics, or relationships not represented in the current static graph. Data Flow produced a small negative context reduction of -0.6%, illustrating that some multi-module tasks require enough connected code to match or exceed the controlled baseline.

D. Dashboard and Manual Response Comparison

The local dashboard visualizes graph-selected nodes, source slices, baseline and optimized context metrics, benchmark evidence, and manual response comparison. Users may label a manually pasted response as originating from Claude, GPT, Gemini, or another model; this label does not represent an API integration.

The Caveman response protocol can reduce response length and encourage structured evidence reporting. However, a shorter response is not automatically more correct. The dashboard therefore reports structural compactness only and

provides manual fields for correctness, information coverage, omissions, and functional-equivalence review.

VI. LIMITATIONS AND FUTURE WORK

CG-CLI has several limitations. Its token values are local estimates based on characters divided by four rather than provider-specific tokenizer counts. The static graph may miss dynamic imports, reflection, runtime-generated relations, indirect invocation, instance-method calls, JSX/CommonJS patterns, and some TypeScript path aliases. Retrieval recall measures expected-symbol coverage rather than answer correctness, patch quality, or model behavior.

The controlled fixture benchmark may not represent the scale and diversity of real open-source repositories. Manually specified expected-symbol sets may also introduce benchmark bias. In addition, adaptive expansion is heuristic and has not been statistically calibrated. CG-CLI does not measure hidden reasoning, actual external-agent tool calls, external model cost, latency, patch correctness, or semantic equivalence.

Future work should evaluate the framework on pinned real-world repositories, incorporate provider-specific tokenizers and cost measurement, improve static and dynamic relationship analysis, combine structural and semantic retrieval, integrate with real coding-agent workflows, and conduct human studies of answer usefulness and developer productivity.

VII. CONCLUSION

CG-CLI is a local graph-guided context-selection framework for AI coding workflows. It indexes JavaScript and TypeScript repositories, constructs a lightweight structural graph, retrieves related source slices, and compares the selected context with a controlled whole-file baseline.

Across 150 controlled cases, the primary one-hop configuration achieved 66.4% average retrieval recall with 16.2% average context reduction. The ablation demonstrated a clear trade-off: broader graph expansion improved recall, but two-hop retrieval increased selected context beyond the baseline. Under the current benchmark policy, Depth 1 provided the best observed balance between expected-symbol coverage and context efficiency.

CG-CLI should be understood as a context-retrieval layer rather than an autonomous coding agent or an external LLM replacement. The results support further study using real repositories, improved structural analysis, provider-specific token measurements, and direct answer-quality evaluation.

REFERENCES

- [1] Y. Tao, Y. Li, Y. Qin, and Y. Liu, "Retrieval-Augmented Code Generation: A Survey with Focus on Repository-Level Approaches," arXiv preprint arXiv:2510.04905, 2025. [Online]. Available: <https://arxiv.org/abs/2510.04905>
- [2] T. Liu, C. Xu, and J. McAuley, "RepoBench: Benchmarking Repository-Level Code Auto-Completion Systems," in *Proc. International Conference on Learning Representations (ICLR)*, 2024. [Online]. Available: <https://arxiv.org/abs/2306.03091>
- [3] Z. Z. Wang, A. Asai, X. V. Yu, F. F. Xu, Y. Xie, G. Neubig, and D. Fried, "CodeRAG-Bench: Can Retrieval Augment Code Generation?," arXiv

- preprint arXiv:2406.14497, 2024. [Online]. Available: <https://arxiv.org/abs/2406.14497>
- [4] S. Ouyang, W. Yu, K. Ma, Z. Xiao, Z. Zhang, M. Jia, J. Han, H. Zhang, and D. Yu, "RepoGraph: Enhancing AI Software Engineering with Repository-level Code Graph," in *Proc. International Conference on Learning Representations (ICLR)*, 2025. [Online]. Available: <https://arxiv.org/abs/2410.14684>
- [5] P. Shah, R. Ghosh, A. Singhal, and D. Dutta, "RANGER: Repository-Level Agent for Graph-Enhanced Retrieval," arXiv preprint arXiv:2509.25257, 2025. [Online]. Available: <https://arxiv.org/abs/2509.25257>
- [6] Z. Chen et al., "LocAgent: Graph-Guided LLM Agents for Code Localization," arXiv preprint arXiv:2503.09089, 2025. [Online]. Available: <https://arxiv.org/abs/2503.09089>
- [7] H. N. Phan, H. N. Phan, T. N. Nguyen, and N. D. Q. Bui, "RepoHyper: Search-Expand-Refine on Semantic Graphs for Repository-Level Code Completion," arXiv preprint arXiv:2403.06095, 2024. [Online]. Available: <https://arxiv.org/abs/2403.06095>
- [8] W. Liu, A. Yu, D. Zan, B. Shen, W. Zhang, H. Zhao, Z. Jin, and Q. Wang, "GraphCoder: Enhancing Repository-Level Code Completion via Coarse-to-fine Retrieval Based on Code Context Graph," in *Proc. 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2024, pp. 570–581, doi: 10.1145/3691620.3695054.
- [9] W. Cheng, Y. Wu, and W. Hu, "Dataflow-Guided Retrieval Augmentation for Repository-Level Code Completion," in *Proc. Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. [Online]. Available: <https://arxiv.org/abs/2405.19782>
- [10] M. Vogel, F. Meyer-Eschenbach, S. Kohler, E. Grünwald, and F. Balzer, "Codebase-Memory: Tree-Sitter-Based Knowledge Graphs for LLM Code Exploration via MCP," arXiv preprint arXiv:2603.27277, 2026. [Online]. Available: <https://arxiv.org/abs/2603.27277>
- [11] Y. Hu, T. Su, L. Zhao, B. Zhu, and H. Haque, "LARGER: Lexically Anchored Repository Graph Exploration and Retrieval," arXiv preprint arXiv:2605.16352, 2026. [Online]. Available: <https://arxiv.org/abs/2605.16352>
- [12] T. Paipuru, "CodeCompass: Navigating the Navigation Paradox in Agentic Code Intelligence," arXiv preprint arXiv:2602.20048, 2026. [Online]. Available: <https://arxiv.org/abs/2602.20048>
- [13] X. Wang, B. Wang, C. Zhi, J. Han, X. Zhao, J. Yin, and S. Deng, "GRACE: Graph-Guided Repository-Aware Code Completion through Hierarchical Code Fusion," arXiv preprint arXiv:2509.05980, 2025. [Online]. Available: <https://arxiv.org/abs/2509.05980>
- [14] X. Liu et al., "CodexGraph: Bridging Large Language Models and Code Repositories via Code Graph Databases," arXiv preprint arXiv:2408.03910, 2024. [Online]. Available: <https://arxiv.org/abs/2408.03910>
- [15] J. Li et al., "GraphCodeAgent: Dual Graph-Guided LLM Agent for Retrieval-Augmented Repo-Level Code Generation," arXiv preprint arXiv:2504.10046, 2025. [Online]. Available: <https://arxiv.org/abs/2504.10046>
- [16] I. Abdelaziz, J. Dolby, J. McCusker, and K. Srinivas, "A Toolkit for Generating Code Knowledge Graphs," arXiv preprint arXiv:2002.09440, 2020. [Online]. Available: <https://arxiv.org/abs/2002.09440>
- [17] X. Ling et al., "Deep Graph Matching and Searching for Semantic Code Retrieval," arXiv preprint arXiv:2010.12908, 2020. [Online]. Available: <https://arxiv.org/abs/2010.12908>
- [18] M. Athale and V. Vaddina, "Knowledge Graph Based Repository-Level Code Generation," arXiv preprint arXiv:2505.14394, 2025. [Online]. Available: <https://arxiv.org/abs/2505.14394>
- [19] D. Sounthiraraj, J. Hancock, Y. Kortam, A. Javvaji, P. Singh, and S. Shankar, "Code-Craft: Hierarchical Graph-Based Code Summarization for Enhanced Context Retrieval," arXiv preprint arXiv:2504.08975, 2025. [Online]. Available: <https://arxiv.org/abs/2504.08975>
- [20] M. R. Chinthareddy, "Reliable Graph-RAG for Codebases: AST-Derived Graphs vs LLM-Extracted Knowledge Graphs," arXiv preprint arXiv:2601.08773, 2026. [Online]. Available: <https://arxiv.org/abs/2601.08773>
- [21] S. Seddik, F. Seddik, I. Saberi, F. Fard, M. H. Huynh, and P. Thongtanunam, "Context-Augmented Code Generation Using Programming Knowledge Graphs," arXiv preprint arXiv:2601.20810, 2026. [Online]. Available: <https://arxiv.org/abs/2601.20810>
- [22] D. Wu, W. U. Ahmad, D. Zhang, M. K. Ramanathan, and X. Ma, "Repoformer: Selective Retrieval for Repository-Level Code Completion," in *Proc. International Conference on Machine Learning (ICML)*, 2024. [Online]. Available: <https://arxiv.org/abs/2403.10059>
- [23] K. Deng et al., "R2C2-Coder: Enhancing and Benchmarking Real-world Repository-level Code Completion Abilities of Code Large Language Models," arXiv preprint arXiv:2406.01359, 2024. [Online]. Available: <https://arxiv.org/abs/2406.01359>
- [24] M. Hostnik and M. Robnik-Šikonja, "Retrieval-augmented Code Completion for Local Projects Using Large Language Models," arXiv preprint arXiv:2408.05026, 2024. [Online]. Available: <https://arxiv.org/abs/2408.05026>
- [25] M. S. Rashid et al., "SWE-PolyBench: A Multi-language Benchmark for Repository Level Evaluation of Coding Agents," arXiv preprint arXiv:2504.08703, 2025. [Online]. Available: <https://arxiv.org/abs/2504.08703>
- [26] J. Li, Y. Su, and M. R. Lyu, "From Laboratory to Real-World Applications: Benchmarking Agentic Code Reasoning at the Repository Level," arXiv preprint arXiv:2601.03731, 2026. [Online]. Available: <https://arxiv.org/abs/2601.03731>
- [27] J. Ding et al., "NL2Repo-Bench: Towards Long-Horizon Repository Generation Evaluation of Coding Agents," arXiv preprint arXiv:2512.12730, 2025. [Online]. Available: <https://arxiv.org/abs/2512.12730>
- [28] T. Cherny-Shahar and A. Yehudai, "Repository Intelligence Graph: Deterministic Architectural Map for LLM Code Assistants," arXiv preprint arXiv:2601.10112, 2026. [Online]. Available: <https://arxiv.org/abs/2601.10112>
- [29] Z. Liu, Z. Jiang, Z. Ye, H. Wang, J. Liu, and X. Ren, "Effective and Efficient Context Retrieval via Partial Dependency Graph for Repository-Level Code Generation," arXiv preprint arXiv:2608.01927, 2026. [Online]. Available: <https://arxiv.org/abs/2608.01927>