Retrieval-Augmented Code Generation: A Survey with Focus on Repository-Level Approaches

Anonymous authors
Paper under double-blind review

Abstract

Recent advances in large language models (LLMs) have significantly improved automated code generation. While existing approaches have achieved strong performance at the function and file levels, real-world software engineering requires reasoning over entire repositories, including cross-file dependencies, evolving execution environments, and global semantic consistency. This challenge has led to the emergence of Repository-Level Code Generation (RLCG), where models must retrieve, organize, and utilize repository-scale context to generate coherent and executable code changes. To address these challenges, Retrieval-Augmented Generation (RAG) has become an increasingly important paradigm for repository-level code intelligence. In this survey, we present a comprehensive review of Retrieval-Augmented Code Generation (RACG), with a particular focus on repository-level approaches. Rather than viewing RACG as a static "retrieve-then-generate" pipeline, we characterize it as a coupled and evolving process involving context construction, retrieval optimization, generation, and environment interaction. We organize existing methods through a unified analytical framework spanning retrieval substrate, control regime, and evaluation setting. Based on this framework, we systematically examine retrieval strategies, graph-based and non-graph-based retrieval paradigms, training-driven optimizations, and autonomous agent architectures. We further summarize widely used datasets, benchmarks, and system configurations, and discuss key challenges including scalability, reliability, efficiency, and the necessity boundary between RACG and long-context LLMs. Through this survey, we aim to provide a structured understanding of the rapidly evolving RACG landscape and highlight promising directions for future AI-powered software engineering research.

Contents

1 Introduction **4**

2 Preliminary and Methodological Foundations **5**

2.1 Repository-Level Code Generation (RLCG) 5

2.2 Retrieval-Augmented Code Generation (RACG) 6

3 Methodology **6**

3.1 Data Collection Process 7

3.2 Inclusion and Exclusion Criteria 7

3.3 Quality Assessment and Final Corpus 8

3.4 Topic Categorization and Analysis 8

3.5 Manuscript Organization 9

4 Core RACG Strategies **9**

4.1 Non-Graph-based RACG 11

4.1.1 Retrieval Content Construction 11

4.1.2 Retrieval Strategy Optimization 12

4.2 Graph-based RACG 13

4.2.1 Retrieval Content Construction 14

4.2.2 Retrieval Strategy Optimization 15

4.2.3 Observed Design Patterns 15

5 System Enhancements & Workflows **16**

5.1 Training-Driven Optimizations 17

5.1.1 Retrieval module training 17

5.1.2 Generation module training 17

5.2 Autonomous Agent Architectures 18

5.2.1 Level 0: Non-agent Systems 19

5.2.2 Level 1: Partial Agent Systems 19

5.2.3 Level 2: Fully Autonomous Agent Frameworks 20

6 Downstream Tasks and Benchmarks for RLCG **21**

6.1 Downstream Tasks 21

6.2 Benchmarks 21

7 Configurations for RACG Systems **23**

7.1 Data Source 23

7.2 Programming Language Support 24
7.3 Backbone Models for Retrieval and Generation 24

8 Discussion and Future Directions **27**
8.1 Security and Reliability 27
8.2 The Necessity Boundary: RAG vs. Long-Context LLMs 27
8.2.1 Limitations of Existing Approaches 27
8.2.2 Future Directions 28

9 Conclusion **29**

1 Introduction

> "If, as we confront some task, a part of the world functions as a process which, were it done in the head, we would have no hesitation in recognizing as part of the cognitive process, then that part of the world is part of the cognitive process." (Clark & Chalmers, 1998)

Recent advances in large language models (LLMs) are transforming software development from code suggestion to broader software engineering capabilities (Jimenez et al., 2024). Despite the inherent complexity of real-world development, characterized by large codebases, cross-file dependencies, and diverse tooling, agentic systems such as OpenClaw[1], Claude Code[2] and Codex[3] demonstrate that LLMs can operate effectively in these environments and make progress. These advances mark a shift from isolated code generation to more integrated and practical software engineering workflows, as shown in Figure 1. In practice, development is repository-centric, where functionality arises from interactions across files. Consequently, models must not only produce locally correct code, but also maintain global consistency and adapt to an evolving execution environment. These requirements define the core challenges of modern code generation.

Challenges. The first challenge is *long-range dependency modeling*: critical information is distributed across files, documents, and dependencies, while changes in one file may affect distant modules, making it difficult to identify minimal but sufficient context and maintain cross-file consistency (Assogba & Ren, 2025). The second is the *efficiency–correctness trade-off*: issue resolution prioritizes accuracy, whereas real-time completion demands low latency. This trade-off is further complicated by model limitations—strong models may memorize common libraries, while weaker ones suffer from context overload (Wang et al., 2025g). Scaling model size or context length only partially alleviates these issues and is often impractical for local deployment. Finally, *practical constraints* further limit applicability: privacy and compliance restrict cloud-based use of proprietary code (Chen et al., 2025b; God of Prompt, 2025), and most LLMs are trained in outdated public repositories (Otterly.AI Blog, 2024), reducing adaptability in real-world and enterprise settings.

The above challenges indicate that, in real-world settings, code generation is no longer a simple problem, but a complex process involving context selection and external information acquisition. In this context, Retrieval-Augmented Generation (RAG) offers an effective way to address context limitations by dynamically incorporating external knowledge during reasoning. In the code domain, this paradigm has evolved into **Retrieval-Augmented Code Generation (RACG)**, which enhances generation quality by retrieving relevant code and structural information.

However, in repository-level scenarios, RACG is no longer a simple two-stage "retrieve-then-generate" pipeline. Instead, it becomes a dynamic process that can be coupled with customized indexing and structure-aware retrieval. We characterize RACG as a unified process in which the model (i) constructs and retrieves context from large-scale code repositories, and (ii) generates or modifies code based on the retrieved information and current system state. In addition to this core process, various enhancements—such as iterative refinement, multi-source context acquisition, and tool-assisted reasoning—can be incorporated to improve performance (Zhang et al., 2023a; Su et al., 2024; Ma et al., 2025c). This formulation naturally aligns with **Repository-Level Code Generation (RLCG)**, which represents a new and challenging problem.

Existing surveys have reviewed general RAG systems (Gao et al., 2024b; Fan et al., 2024), LLM-based code generation (Jiang et al., 2025a; Dong et al., 2025c), and code generation agents for software engineering (He et al., 2025). However, these works do not systematically examine how retrieval is used to construct, select, and integrate the repository-level context. This survey fills this gap by providing a systematic overview of Retrieval-Augmented Code Generation in repository-level scenarios, organized around three questions: (i) how RACG methods are designed; (ii) how they are formulated and evaluated; and (iii) concerns and necessity boundary. Table 1 summarizes the positioning of this survey relative to previous work.

Key Contributions.

[1] https://github.com/OpenClaw/OpenClaw
[2] https://www.anthropic.com/claude-code
[3] https://openai.com/index/openai-codex/

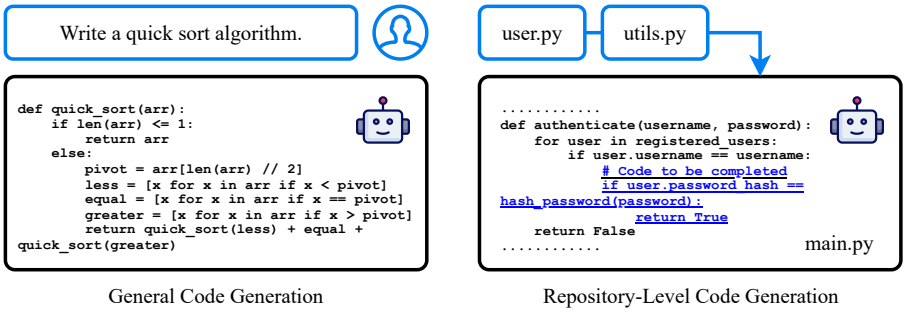

Figure 1: Comparison between General Code Generation and Repository-Level Code Generation

Table 1: Comparison with related surveys. ✓ = fully covered, ∼ = partially covered, × = not covered.

Survey	RAG Focus	Repo-Level	Code Gen	Agent Arch.
Gao et al. (Gao et al., 2024b)	✓	×	×	×
Fan et al. (Fan et al., 2024)	✓	×	×	×
Jiang et al. (Jiang et al., 2025a)	×	×	✓	×
Dong et al. (Dong et al., 2025c)	∼	×	✓	✓
He et al. (He et al., 2025)	×	×	∼	✓
This survey	✓	✓	✓	✓

- We establish Repository-Level Code Generation as a unified problem formulation, explicitly capturing cross-file dependencies, dynamic context, and system-level constraints.
- We provide the first survey of Retrieval-Augmented Code Generation, reframing RAG as a coupled, evolving process rather than a static pipeline.
- We introduce a unified analytical framework that organizes RACG along retrieval, control, and integration dimensions, covering both model design and agentic workflows.
- We systematically evaluate existing methods, benchmarks, and system configurations, revealing key trade-offs between efficiency, scalability, and correctness.
- We identify critical gaps in current research and outline future directions toward better RACG systems.

2 Preliminary and Methodological Foundations

This section formalizes the key concepts underlying Retrieval-Augmented Code Generation in repository-level settings. We focus on minimal yet sufficient abstractions that align with the paradigm introduced in Section 1. RLCG defines the task setting, while RACG describes a family of retrieval-augmented methodologies to solve repository-level code generation problems.

2.1 Repository-Level Code Generation (RLCG)

Repository-Level Code Generation (RLCG) refers to the generation or modification of code in the context of an entire software repository, rather than isolated functions or files.

Formally, let a repository be denoted as $\mathcal{R} = \{f_1, f_2, \ldots, f_n\}$, where each f_i represents a file containing code, documentation, or configuration. Given a task specification q (e.g., a bug report, feature request, unfinished code snippet), the objective of RLCG is to produce a modification $\Delta\mathcal{R}$ such that:

$$\mathcal{R}' = \mathcal{R} \oplus \Delta\mathcal{R} \tag{1}$$

satisfies both local correctness (syntactic and semantic validity) and global consistency across the repository.

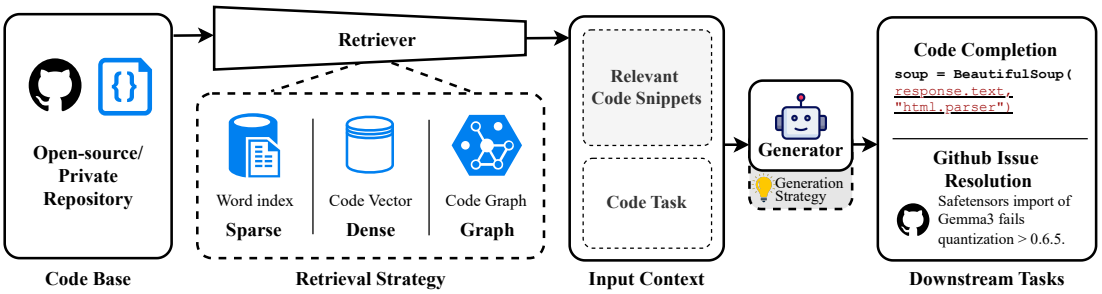

Figure 2: Retrieval-Augmented Code Generation, focusing on Repository-Level approaches.

Typical applications include

- *Cross-file Code Completion:* Predicting or synthesizing missing code segments by leveraging repository-wide context, from single lines to full functions.
- *Issue or PR Resolution:* Automatically fixing reported problems by locating relevant files and generating patches consistent with project conventions.

We discuss a broader range of applications in Section 6.1.

Unlike function-level generation, RLCG requires reasoning over long-range dependencies, cross-file interactions, and system-level constraints. The generation process is inherently state-dependent, as intermediate changes may alter the repository structure and influence subsequent decisions.

2.2 Retrieval-Augmented Code Generation (RACG)

Retrieval-Augmented Code Generation enhances LLMs with external software knowledge via retrieval. Unlike pure LLM generation, RACG dynamically retrieves relevant information such as code, documentation, or graphs to construct context-aware prompts.

In its basic form, RACG (Figure 2) consists of two components: (i) **Retriever** $\mathcal{M}_r$, which selects the relevant context $C \subseteq \mathcal{R}$ given a query or intermediate state. It typically operates through several structured strategies, including *identifier matching* based on exact names or signatures, *sparse retrieval* using lexical or sparse-vector methods such as TF-IDF (Manning et al., 2008) or BM25 (Robertson & Walker, 1994), *dense retrieval* using neural embeddings such as CodeBERT (Feng et al., 2020) or UniXcoder (Guo et al., 2022) with nearest-neighbor search, *graph-based retrieval* leveraging structures such as abstract syntax trees and call graphs via traversal or subgraph matching, and *hybrid retrieval* that integrates lexical, embedding, and structural signals to balance precision and recall. (ii) **Generator** $\mathcal{M}_g$ is a language model such as GPT-5.5 (OpenAI, 2025a) or CodeLlama (Rozière et al., 2024) that produces code conditioned on the context retrieved.

Traditional RAG follows a static pipeline:

$$C = \mathcal{M}_r(q), \quad y = \mathcal{M}_g(q, C) \tag{2}$$

However, in repository-level settings, RACG departs from this simple paradigm. Instead of treating retrieval as a static preprocessing step, RACG operates as a coupled process where retrieval and generation are interdependent and evolve over time. Recent work also explores iterative or agent-style RACG frameworks (Zhang et al., 2023a; Yu et al., 2025).

3 Methodology

This section outlines the methodology used to conduct a systematic literature review. We adopt a structured review approach inspired by established guidelines in software engineering literature (Petersen et al., 2008;

He et al., 2025; Jiang et al., 2025a). The methodology encompasses research question formulation, data collection, inclusion and exclusion filtering, quality assessment, snowballing, and topic categorization.

- **RQ1: How does RACG evolve from simple retrieval paradigms to complex autonomous architectures?**
  This question investigates the methodological shift from semantic retrieval to structure-aware graph retrieval. It further explores how training-based optimizations and agentic architectures address complex task resolution.
- **RQ2: What are the core configurations, task environments, and evaluation practices for RACG systems?**
  This question examines the ecosystem and implementation of RACG, covering the impact of diverse data sources and programming language paradigms. It also evaluates how downstream tasks and current benchmark metrics reflect the demands of practical software engineering.
- **RQ3: What are the concerns of RACG systems and where is its necessity boundary?**
  This question analyzes key system-level bottlenecks, including reliability, scalability, and structural complexity. It further examines when retrieval remains necessary versus when long-context modeling is sufficient, and outlines directions for bridging this gap toward real-world deployment.

3.1 Data Collection Process

To address our research questions and construct a comprehensive corpus on RACG, we adopted a systematic two-stage data collection pipeline, with the primary focus placed on **methodological advances** in retrieval-augmented and repository-level code generation.

In the first stage, we determined our core search keywords by analyzing accepted papers related to this topic. This initial step allowed us to derive and refine a precise set of search terms strongly associated with RACG. Specifically, the search keywords include `retrieval-augmented code generation`, `repository-level code generation`, `code retrieval`, `code search`, `code RAG`, `code completion`, `issue resolution`, and `issue localization`.

In the second stage, we collected all publications from leading artificial intelligence and software engineering venues and journals (`ICLR`, `NeurIPS`, `ICML`, `IJCAI`, `AAAI`, `JMLR`, `ACL`, `EMNLP`, `ICSE`, `FSE`, `ASE`, `TOSEM`, `TSE`, and `ISSTA`). We then conducted keyword similarity searches across multiple bibliographic platforms, including `ACM Digital Library`, `IEEE Xplore`, `arXiv`, `OpenReview`, and the `ACL Anthology`. For journals and platforms with anti-crawling mechanisms, we supplemented automated retrieval with manual scanning. For preprints on `arXiv`, we implemented a hybrid approach: we first queried the arXiv API using the keywords, and subsequently applied similarity filtering to isolate the most relevant papers.

The search period spanned from January 1, 2023 to March 31, 2026. This process resulted in 243 candidate papers from the targeted conferences and journals, along with 610 papers from `arXiv`. After consolidating these sources and removing duplicates, the resulting candidate pool was screened according to the criteria detailed in Section 3.2.

3.2 Inclusion and Exclusion Criteria

A paper was included if it met all the following:

- The task involves retrieval-augmented, cross-file, or repository-level settings that enhance code generation.
- The study proposes a novel RAG-based approach or analyzes the impact of retrieval on generation quality.
- The full text is accessible and written in English.

A paper was excluded if it:

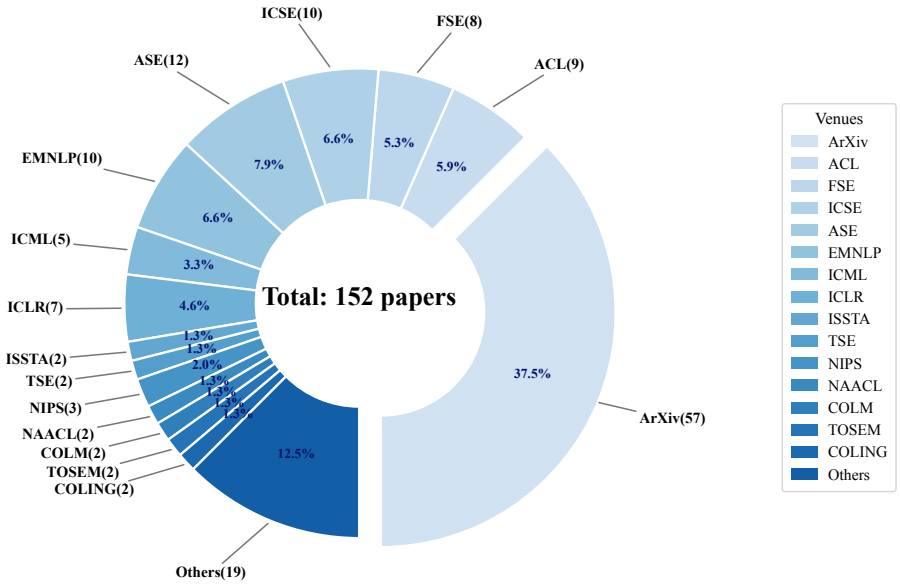

Figure 3: Distribution of selected RACG papers across publication venues.

- Focuses solely on standalone code generation or static analysis tools without pretrained Language Models.
- Uses RAG only as a minor component without methodological contribution.
- Target tasks such as vulnerability or clone detection, or purely retrieval model training without generation (except for issue localization tasks, which are retained despite being purely retrieval-based due to their high relevance to repository-level settings).
- Presents only conceptual discussions or ethical commentary without technical contributions.

3.3 Quality Assessment and Final Corpus

To ensure the rigor and completeness of our review, each candidate article was evaluated for: (1) relevance to RACG, (2) methodological clarity, and (3) reproducibility. Works that did not meet these criteria were excluded. For preprints sourced from `arXiv`, we additionally required a detailed method section and solid experimental results to compensate for the lack of formal peer review.

Furthermore, backward snowballing was applied to the reference lists of included papers to identify foundational and related works potentially missed by the initial keyword searches.

After completing these rigorous screening and quality checks, a final corpus of 152 papers was retained, comprehensively covering the intersections of software engineering, natural language processing, and machine learning.

3.4 Topic Categorization and Analysis

We manually annotated each paper along multiple dimensions, including task type, system architecture, retrieval method, evaluation setting, and target domain. These annotations support both the quantitative distribution analysis and the taxonomy developed in the following sections.

Figure 3 shows the publication venue distribution of the 152 selected papers. `arXiv` constitutes the largest category, with 57 papers, followed by major software engineering and NLP venues such as `ASE`, `ICSE`, `EMNLP`, `ACL`, and `FSE`. This distribution reflects the rapid iteration cycle of RACG research, where many works first appear as preprints while increasingly entering established venues in software engineering, natural language processing, and machine learning.

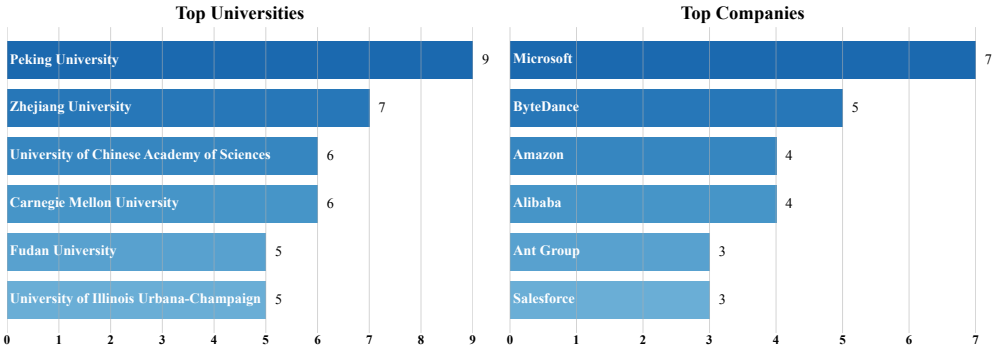

Figure 4: Top contributing universities and companies in RACG research.

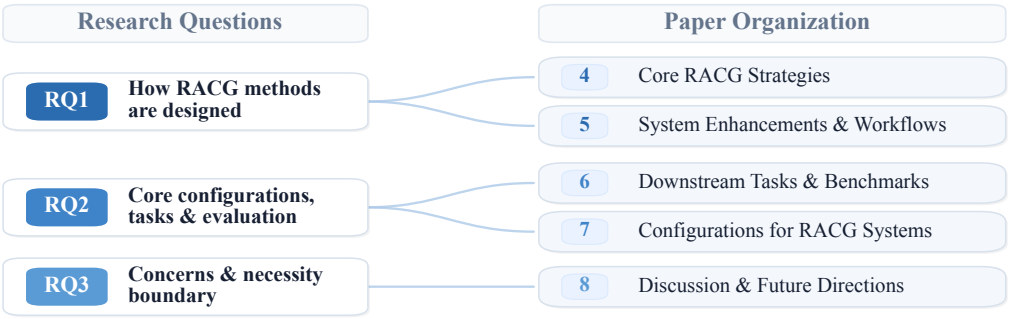

Figure 5: Mapping between the research questions and the organization of this survey.

Figure 4 further summarizes the leading institutions contributing to RACG research. Among universities, representative contributors include `Peking University`, `Zhejiang University`, `University of Chinese Academy of Sciences`, `Carnegie Mellon University`, `Fudan University`, and `University of Illinois Urbana-Champaign`. Among industry labs, major contributors include `Microsoft`, `ByteDance`, `Amazon`, `Alibaba`, `Ant Group`, and `Salesforce`. The distribution indicates that RACG has attracted sustained attention from both academia and industry, reflecting its relevance to practical code intelligence systems.

3.5 Manuscript Organization

This section constitutes the main analytical body of our survey, synthesizing 152 selected papers on Retrieval-Augmented Code Generation with a particular emphasis on repository-level settings. To distinguish our work from prior general-purpose RAG surveys (Jiang et al., 2025a; He et al., 2025), we organize our findings around three coupled analytical dimensions: (1) **retrieval substrate** (*how* context is represented and retrieved), (2) **control regime** (*when* and *how often* retrieval is invoked), and (3) **evaluation setting** (*what tasks and benchmarks* reflect real-world demands). As illustrated in Figure 5, these dimensions are reflected in the survey structure: Sections 4–5 focus on retrieval substrate and control regime, while Sections 6–7 extend the discussion to downstream tasks, benchmarks, backbone models, and system configurations commonly adopted in RACG research. Section 8 further provides a cross-cutting analysis of system-level limitations and the necessity boundary of RACG. These dimensions also guide our three research questions. **RQ1** is addressed in Sections 4–5, **RQ2** in Sections 6–7, and **RQ3** (core limitations and necessity boundary of RACG) in Section 8.

4 Core RACG Strategies

To systematically organize the RACG literature, we construct a taxonomy of RAG strategies through an iterative analysis of the selected studies. Specifically, we examine the retrieval substrate, context representation,

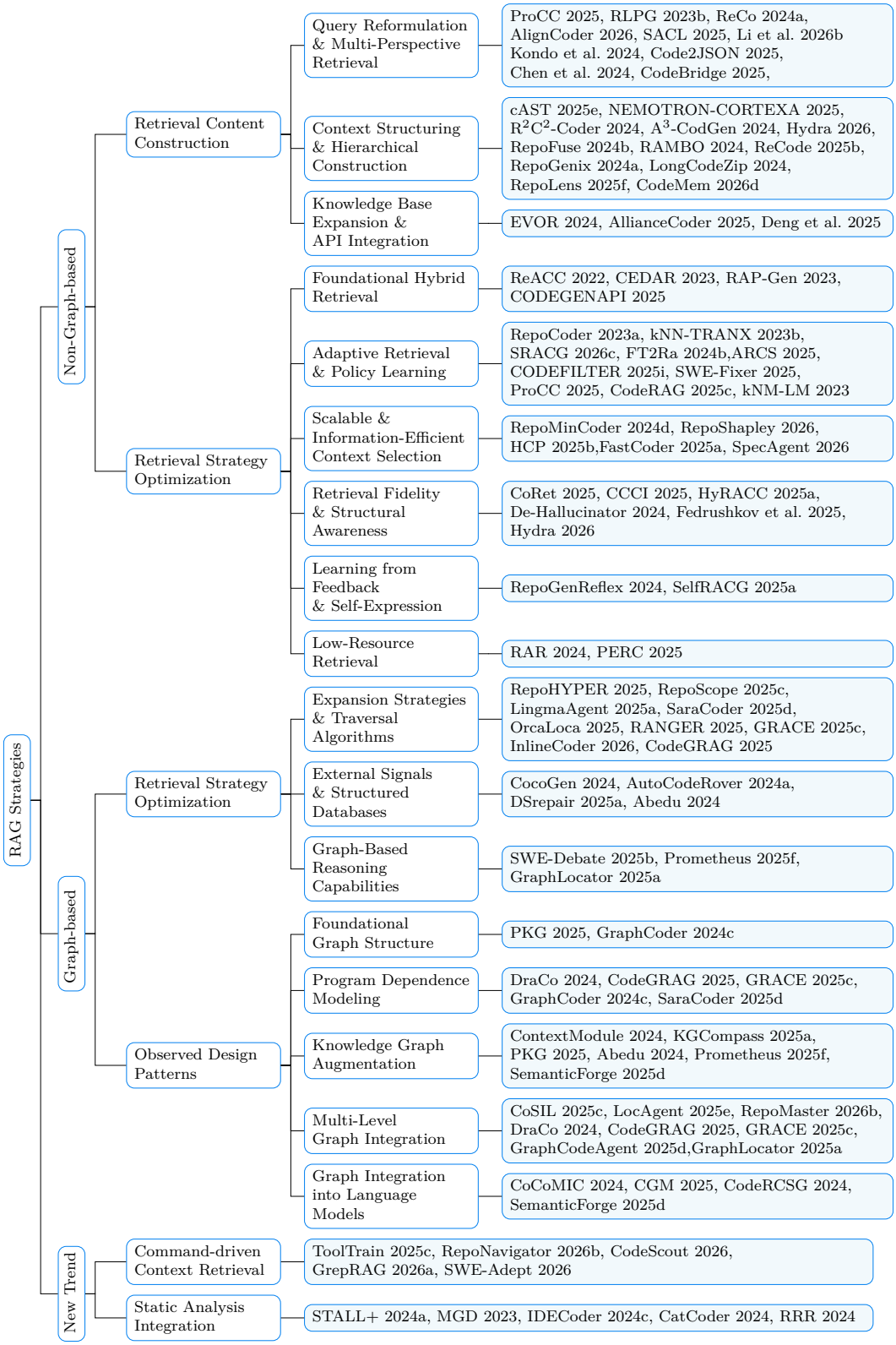

Figure 6: Taxonomy of RAG Strategies.

and retrieval mechanisms adopted by prior work, and group methods according to their dominant retrieval characteristics. As illustrated in Figure 6, Retrieval-Augmented Code Generation methods can be broadly categorized into two major paradigms: **non-graph-based** and **graph-based**. We classify methods by their *primary* retrieval substrate: a method is considered graph-based if its core retrieval mechanism depends on explicitly constructed graph structures, and non-graph-based otherwise. Methods that inject lightweight structural signals but retrieve over flat text collections remain in the non-graph category. Hybrid approaches are grouped according to their predominant component and noted explicitly. The two paradigms are detailed in turn below.

4.1 Non-Graph-based RACG

Non-graph-based RAG methods retrieve code without explicit graph structures, treating repositories as flat text collections and identifying relevant context via similarity matching. Early approaches rely on lexical signals (e.g., BM25 (Robertson & Walker, 1994), Jaccard similarity (Jaccard, 1901)) to match identifiers or keywords, but struggle with deeper semantics. Recent systems shift to semantic retrieval using representation learning. Models such as GraphCodeBERT (Guo et al., 2021) and UniXcoder (Guo et al., 2022) encode code into embeddings, enabling similarity matching that better captures functional and contextual relationships.

Despite their simplicity, these methods remain widely used due to efficiency and ease of integration. Modern RACG systems extend them along two axes: **retrieval strategies** (how to search) and **retrieval content construction** (what to retrieve and how to organize it). Emerging directions further incorporate **static analysis** for structural signals and **command-driven retrieval** for dynamic, interactive context acquisition.

4.1.1 Retrieval Content Construction

This line of work asks ***what to retrieve***. Rather than modifying retrieval algorithms, it improves performance by reformulating queries, restructuring content, and enriching the knowledge base.

Query Reformulation and Multi-Perspective Retrieval. A key issue is the mismatch between query form and code semantics. To address this, methods enhance query expressiveness in two ways. First, *multi-perspective enrichment* generates complementary query views. ProCC (Tan et al., 2025) combines lexical, hypothesis, and summary views with bandit selection. RLPG (Shrivastava et al., 2023b) samples candidate contexts and learns selection. ReCo (Li et al., 2024a) rewrites queries and code for consistency. AlignCoder (Jiang et al., 2026) further leverages multiple candidate completions to construct richer queries. Second, *conversion-based retrieval* bridges modality gaps. SACL (Gupta et al., 2025) augments code with semantic descriptions. Kondo et al. (Kondo et al., 2024) convert code to natural language. Code2JSON (Singhal et al., 2025), Chen et al. (Chen et al., 2024), and CodeBridge (Liang et al., 2025) explore structured or alternative matching. Li et al. (Li et al., 2026b) introduce reasoning-based intent synthesis via docstring-like generation.

Context Structuring and Hierarchical Construction. Another challenge is organizing retrieved content. Methods improve usability via structuring, hierarchy, and compression. For *structure-aware chunking*, cAST (Zhang et al., 2025e) uses AST-based partitioning, while Hydra (Le-Anh et al., 2026) preserves function-level integrity. For *multi-level organization*, R^2C^2-Coder (Deng et al., 2024) builds abstract and fragment pools, A^3-CodGen (Liao et al., 2024) integrates multi-granularity context, and NEMOTRON-CORTEXA (Sohrabizadeh et al., 2025) emphasizes solution diversity. For *fusion and compression*, RepoFuse (Liang et al., 2024b), RAMBO (Bui et al., 2024), and RepoGenix (Liang et al., 2024a) compose compact prompts, while LongCodeZip (Bogomolov et al., 2024) applies coarse (function-level) and fine (block-level) pruning. Search-space reduction is also explored: ReCode (Zhao et al., 2025b) predicts task type, RepoLens (Wang et al., 2025f) clusters conceptual concerns, and CodeMem (Wang et al., 2026d) maintains API-level and session-level memory.

Knowledge Base Expansion and API Integration. When required knowledge is missing, the corpus itself must be expanded. EVOR (Su et al., 2024) dynamically updates both queries and the knowledge base for evolving domains. AllianceCoder (Gu et al., 2025) generates API-level descriptions from queries,

Table 2: Static analysis integration.

System	Usage
MGD (2023)	Feedback loop ensures consistent generation of types, identifiers, API protocol, and enum values.
STALL+ (2024a)	Cross-file dependent contexts used in prompt formulation, decoding control, and post-processing.
IDECoder (2024c)	Retrieves contexts of code elements, project structure, developer intention, and error feedback.
CatCoder (2024)	Builds a type dependency graph to guide prompts within scope for practicality.
RRR (2024)	Iteratively refines context for undefined symbols and dependencies, class and member metadata.

while Deng et al. (Deng et al., 2025) infer implicit API dependencies from drafts to retrieve relevant APIs. Together, these methods improve retrieval by aligning queries with intent, structuring context for usability, and enriching the underlying knowledge base.

4.1.2 Retrieval Strategy Optimization

Retrieval strategy optimization studies ***what* to fetch, *when* to fetch it, and *how much* to use**. These choices determine whether retrieved context helps or harms generation. Foundational work, like ReACC (Lu et al., 2022), CEDAR (Nashid et al., 2023), and RAP-Gen (Wang et al., 2023), combine BM25 with dense embeddings, leveraging lexical matching for exact identifiers and dense retrieval for semantic similarity. CODEGENAPI (Zan et al., 2025) extends this to private codebases via API signature retrieval.

Adaptive Retrieval and Policy Learning. Static strategies fail to adapt to query complexity and uncertainty. RepoCoder (Zhang et al., 2023a) iteratively refines queries using intermediate generations, while kNM-LM (Tang et al., 2023) injects retrieval selectively for poorly predicted tokens. Other methods learn adaptive policies. kNN-TRANX (Zhang et al., 2023b) applies syntax constraints with adaptive weighting. ProCC (Tan et al., 2025) selects retrievers via LinUCB (Li et al., 2010), and FT2Ra (Guo et al., 2024b) uses Δlogits to trigger retrieval. ARCS (Bhattarai et al., 2025) decomposes queries, while CODEFILTER (Li et al., 2025i) and SWE-Fixer (Xie et al., 2025) refine context usage. CodeRAG (Zhang et al., 2025c) introduces log-probability-guided probing and multi-path retrieval. SRACG (Wang et al., 2026c) adds necessity-aware retrieval, multi-objective selection, and preference consistency filtering.

Scalable and Information-Efficient Context Selection. Large repositories exceed context limits, making selection critical. HCP (Zhang et al., 2025b) prunes context via structural reachability, while RepoMinCoder (Li et al., 2024d) optimizes information retention under length constraints. RepoShapley (Huo et al., 2026) models selection as a cooperative game using Shapley values. Efficiency-oriented methods include FastCoder (Zhao et al., 2025a), which adopts speculative decoding with draft-and-verify stages, and SpecAgent (Ma et al., 2026), which precomputes speculative contexts via future-edit prediction.

Retrieval Fidelity and Structural Awareness. Reducing context size does not ensure correctness. CoRet (Fehr et al., 2025) incorporates call-graph structure, CCCI (Jin & Hamdaqa, 2025) filters inconsistent files, and HyRACC (Li et al., 2025a) adaptively fuses model and datastore outputs. To reduce hallucinations, De-Hallucinator (Eghbali & Pradel, 2024) verifies API usage, while Fedrushkov et al. (Fedrushkov et al., 2025) improve retriever robustness via syntactic hierarchy and contrastive learning. Hydra (Le-Anh et al., 2026) further learns dependency-aware retrieval through pairwise classification.

Learning from Feedback and Self-Expression. Retrieval is often decoupled from generation. RepoGenReflex (Wang et al., 2024) introduces a Reflector that generates feedback to guide subsequent retrieval. SelfRACG (Dong et al., 2025a) enables models to express information needs explicitly before retrieval.

Low-Resource Retrieval. In low-resource settings, weak corpora limit retrieval. RAR (Dutta et al., 2024) combines multiple weak sources (e.g., examples, grammar) via dual retrievers, while PERC (Yoo et al., 2025) converts tasks into pseudocode to enable cross-language retrieval.

Emerging Direction: Static Analysis Integration

A complementary direction incorporates *static analysis* to provide explicit structural signals (e.g., types, dependencies, execution constraints), overcoming limitations of similarity-based retrieval.
These signals can be integrated throughout the pipeline. Representative systems and their integration strategies are summarized in Table 2. STALL+ (Liu et al., 2024a) applies static analysis in prompting, decoding, and post-processing. MGD (Agrawal et al., 2023) enforces type consistency via decoding-time masking. IDECoder (Li et al., 2024c) leverages IDE-level analysis for precise context retrieval. They also support prompt refinement. CatCoder (Pan et al., 2024) uses type-dependency graphs to constrain prompts, while RRR (Deshpande et al., 2024) iteratively refines context with analysis and execution feedback.
Overall, static analysis improves robustness and correctness by adding explicit structural constraints.

Emerging Direction: Command-driven Context Retrieval

A recent trend shifts retrieval from one-shot context selection to *agent-driven search*, where LLMs actively decide what to query, which files or symbols to inspect, and when to stop. ToolTrain (Ma et al., 2025c), RepoNavigator (Zhang et al., 2026b), and CodeScout (Sutawika et al., 2026) train agents to conduct multi-step repository exploration through tool calls or terminal commands. GrepRAG (Wang et al., 2026a) shows that LLM-generated `ripgrep` commands can serve as effective retrieval queries, while SWE-Adept (He & Roy, 2026) uses agent-directed traversal to follow relevant dependency paths. Overall, command-driven retrieval makes context acquisition more adaptive, reduces irrelevant context, and turns retrieval into a test-time decision process.

Takeaways

The two dimensions correspond to core components of the RACG process. Retrieval strategy optimization controls the retriever $\mathcal{M}_r$ and its interaction with state s_t, enabling dynamic context selection. Retrieval content construction defines $\mathcal{C}(\mathcal{R}, s_t)$, determining how repository information is formatted into model inputs.
Together, they form a unified design space balancing adaptability, informativeness, and correctness.

4.2 Graph-based RACG

Graph-based RAG incorporates explicit structural information from code repositories by representing code as graphs rather than flat text. Although designs vary, they share the principle of leveraging structured representations to enhance retrieval. In practice, parsing tools such as Tree-sitter (Brunsfeld et al., 2013) are commonly used to extract Abstract Syntax Trees (ASTs), which form the basis for graph construction.

Formally, a repository is modeled as a graph $G = (V, E)$, where nodes V represent code entities at different granularities, and edges E encode structural, syntactic, or semantic relationships. This graph acts as a structured index, allowing retrieval to consider both textual similarity and relational context.

Table 3 provides a systematic comparison of recent graph-based RACG approaches. When analyzing a graph-based system, two primary factors determine its effectiveness: **how the graph G is constructed**, and **how retrieval is performed over the graph structure**. The following sections examine these two dimensions and summarize design patterns in current research.

Table 3: Comparison of Graph-Based RAG Methods

Method	Edge Types						Node Types				
	Contain	Import	Inherit	Invoke	Data Flow	Ctrl Flow	Dir.	Module	Class	Func.	Line
AutoCodeRover (2024a)	✓	×	×	×	×	×	×	✓	✓	✓	×
CocoGen (2024)	✓	✓	✓	×	×	×	×	✓	✓	✓	×
CoCoMIC (2024)	✓	✓	×	×	×	×	×	✓	✓	✓	×
CodePlan (2024)	✓	✓	✓	✓	×	×	×	✓	✓	✓	×
ContextModule (2024)	✓	×	×	✓	×	×	×	✓	✓	✓	×
DraCo (2024)	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	×
GraphCoder (2024c)	×	×	×	✓	✓	✓	×	×	×	×	✓
PKG (2025)	✓	×	×	×	×	×	×	×	×	✓	✓
KG-RLCG (2025)	✓	×	×	×	×	×	×	✓	✓	✓	×
CGM (2025)	✓	✓	✓	✓	×	×	✓	✓	✓	✓	×
CodeGRAG (2025)	✓	×	×	✓	✓	✓	×	×	×	✓	×
GraphCodeAgent (2025d)	✓	✓	✓	✓	×	×	×	✓	✓	✓	×
CodexGraph (2025b)	✓	×	✓	✓	×	×	×	✓	✓	✓	×
CoSIL (2025c)	✓	✓	✓	✓	×	×	✓	✓	✓	✓	×
GRACE (2025c)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×
GraphLocator (2025a)	✓	✓	✓	✓	×	×	✓	✓	✓	✓	×
HCGS (2025)	✓	✓	✓	✓	×	×	×	✓	✓	✓	×
KGCompass (2025a)	✓	×	×	✓	×	×	×	✓	✓	✓	×
LingmaAgent (2025a)	✓	×	✓	✓	×	×	✓	✓	✓	✓	×
LocAgent (2025e)	✓	✓	✓	✓	×	×	✓	✓	✓	✓	×
OrcaLoca (2025)	✓	×	×	✓	×	×	✓	✓	✓	✓	×
Prometheus (2025f)	✓	×	×	×	×	×	✓	✓	✓	✓	×
RepoGraph (2025b)	✓	×	×	✓	×	×	×	×	✓	✓	✓
RepoHYPER (2025)	✓	✓	✓	✓	×	×	×	✓	✓	✓	×
RepoScope (2025c)	✓	✓	✓	✓	×	×	×	×	✓	✓	×
SaraCoder (2025d)	✓	✓	×	×	✓	✓	×	✓	✓	✓	×
SemanticForge (2025d)	✓	✓	✓	✓	×	×	×	✓	✓	✓	×
SWE-Debate (2025b)	✓	✓	✓	✓	×	×	×	×	✓	✓	×
SynFix (2025)	✓	✓	×	✓	×	×	✓	✓	✓	✓	×
InlineCoder (2026)	×	×	×	✓	×	×	×	×	×	✓	×
RepoMaster (2026b)	✓	✓	×	✓	×	×	✓	✓	✓	✓	×
RPG (2026)	✓	✓	×	✓	✓	×	✓	✓	✓	✓	×
SWE-Adept (2026)	✓	×	×	✓	×	×	×	✓	✓	✓	×

4.2.1 Retrieval Content Construction

Edge Types Edges define typed relations between code entities. Each edge can be represented as a tuple (v_i, v_j, r), where $v_i, v_j \in V$ and r denotes the relation type. We consider six common types:

- **Contain**: Captures structural inclusion, such as classes containing functions or modules containing classes. It may also represent hierarchical containment across multiple code blocks.
- **Import**: Encodes static dependencies introduced by language-level import or include statements across modules or packages.
- **Inherit**: Represents inheritance relationships between classes, reflecting object-oriented structure.
- **Invoke**: Captures call relations between functions or methods, modeling execution-level dependencies.
- **Data Flow (Data F.)**: Tracks how data propagates between variables, parameters, or return values.
- **Control Flow (Ctrl F.)**: Represents control dependencies induced by conditionals, loops, and branching constructs.

Node Types Nodes correspond to code entities at different levels of granularity, forming a hierarchical representation of the repository:

- **Directory**: Represents the top-level physical organization of a repository.

- **Module**: Corresponds to a single source file (e.g., `.py`, `.java`, `.cpp`). Although often referred to as *File* in prior work, we use *Module* to emphasize its role as a logical layer between Directory and Class.
- **Class / Function**: Core semantic units in most programming languages and common targets for retrieval.
- **Line**: The smallest addressable unit in code, enabling fine-grained retrieval and alignment.

4.2.2 Retrieval Strategy Optimization

Given a repository graph $G = (V, E)$, retrieval can be formulated as selecting a subgraph $G' = (V', E')$ conditioned on a query q. A common workflow first identifies seed nodes via lexical or semantic matching, then expands them to incorporate structural context.

Expansion strategies determine how structure is utilized. Local methods such as k-NN or n-hop subgraphs capture nearby dependencies; Breadth-First Search emphasizes immediate neighbors, while Depth-First Search explores deeper paths. Some agent-based systems further define specialized primitives over ASTs (e.g., `search_node`, `traverse_graph` (Chen et al., 2025e)). Pruning is typically applied to control context size.

Expansion Strategies and Traversal Algorithms. Existing methods can be viewed as different instantiations of expansion and traversal strategies, including redesigned relevance scoring and diverse search algorithms. RepoHYPER (Phan et al., 2025) follows a search-expand-refine paradigm based on k-NN with a learned link predictor for re-ranking. RepoScope (Liu et al., 2025c) leverages K-means to predict call chains. LingmaAgent (Ma et al., 2025a) and RANGER (Shah et al., 2025) cast traversal as a search problem using Monte Carlo Tree Search. OrcaLoca (Yu et al., 2025) lets LLM schedule retrieval order via relevance scores. SaraCoder (Chen et al., 2025d) improves retrieval diversity using Decaying Subgraph Edit Distance and Maximal Marginal Relevance. InlineCoder (Hu et al., 2026) retrieves callees after drafting code, while CodeGRAG (Du et al., 2025) and GRACE (Wang et al., 2025c) employ Graph Neural Networks to predict node relevance.

Integration of External Signals and Databases. Some approaches incorporate external signals or structured databases. CocoGen (Bi et al., 2024) refines retrieval using compiler feedback, and AutoCodeRover (Zhang et al., 2024a) combines hierarchical search with fault localization. DSrepair (Ouyang et al., 2025a) models code as RDF triples for SPARQL queries, while Abedu et al. (2024) translate natural language into Cypher queries.

Graph-Based Reasoning Capabilities. Other work enhances reasoning over graphs. SWE-Debate (Li et al., 2025b) uses multi-agent debate over fault propagation paths. Prometheus (Chen et al., 2025f) introduces a working memory to avoid redundant reasoning and traversal, and GraphLocator (Liu et al., 2025a) constructs causal graphs for step-by-step root cause analysis.

4.2.3 Observed Design Patterns

Foundational Graph Structure. Most methods adopt a three-tier abstraction of `Module`, `Class`, and `Function`, connected by `Contain` edges for hierarchy and `Invoke` edges for call relations. This design preserves structure while remaining tractable for language model reasoning. A few approaches use finer granularity for higher precision: PKG (Saberi & Fard, 2025) employs block- and function-level units with pruning, while GraphCoder (Liu et al., 2024c) builds statement-level graphs with control-flow and data-dependence edges, improving local accuracy at the cost of coverage and overhead.

Program Dependence Modeling. Only a subset of methods explicitly model fine-grained dependencies. DraCo (Cheng et al., 2024) captures variable dependencies via extended dataflow analysis, while CodeGRAG (Du et al., 2025), GraphCoder (Liu et al., 2024c), GRACE (Wang et al., 2025c), and SaraCoder (Chen et al., 2025d) incorporate both control flow and data flow. Adoption remains limited due to reliance on language-specific analysis, cross-function complexity, and high construction cost, although such graphs provide the most precise semantics without execution.

Knowledge Graph Augmentation. Some methods extend graphs with repository metadata and external context. ContextModule (Guan et al., 2024) integrates user behavior and code signals, KGCompass (Yang et al., 2025a) links code with issues and pull requests for multi-hop reasoning, and PKG (Saberi & Fard, 2025) unifies code and text into a programming knowledge graph with multi-granularity retrieval. Abedu (Abedu et al., 2024) supports question answering over repository graphs, while Prometheus (Chen et al., 2025f) augments knowledge graphs with working memory for multi-step reasoning. SemanticForge (Zhang et al., 2025d) further applies knowledge graphs to repository-level code generation with correctness guarantees.

Multi-Level Graph Integration. A clear trend is the integration of multiple structural levels within a single system, where different graph representations are assigned complementary roles for context resolution. CoSIL (Jiang et al., 2025c) adopts a two-stage retrieval pipeline, first locating relevant files via a coarse module-level call graph and then refining context using a function-level graph. LocAgent (Chen et al., 2025e) organizes repositories into a hierarchical dependency graph exposed as an interactive reasoning tool. Beyond structural hierarchies, several frameworks bridge modalities: GraphCodeAgent (Li et al., 2025d) links requirement-level semantics with code structure, while DraCo (Cheng et al., 2024) and CodeGRAG (Du et al., 2025) combine structural topology with semantic data flow to improve retrieval effectiveness. For autonomous exploration, RepoMaster (Wang et al., 2026b) integrates a Hierarchical Code Tree, Function Call Graph, and Module Dependency Graph. GRACE (Wang et al., 2025c) further advances this direction with a multi-level, multi-semantic graph and hierarchical fusion. Finally, GraphLocator (Liu et al., 2025a) constructs a Causal Issue Graph over a Repository Dependency Fractal Structure Graph to explicitly model symptom-to-root-cause paths.

Graph Integration into Language Models. Several works study how to inject graph information into language models. CoCoMIC (Ding et al., 2024) encodes code entities as tokens via causal attention, CGM (Tao et al., 2025) integrates graph structure into attention with dedicated adapters, and CodeRCSG (Jiang et al., 2024) aligns cross-lingual semantic graphs with model representations. SemanticForge (Zhang et al., 2025d) incorporates graph constraints into beam search with formal verification.

Takeaways

Taken together, graph-based RACG can be understood as operating over a structured retrieval space, where graph construction defines the search space, retrieval strategies determine how this space is explored, and integration mechanisms control how structural information influences generation.
Despite these advances, graph-based methods face a fundamental limitation: their construction pipelines depend on language-specific parsers and static analyzers, which reduces portability across programming languages.

5 System Enhancements & Workflows

This section explores the methodologies for optimizing the end-to-end RACG pipeline. While modular RAG can be implemented in a lightweight, zero-shot manner, the inherent complexity often requires a tighter coupling between retrieval signals and the generation process. We categorize works into two primary paradigms:

First, **Training-Driven Optimizations** (Section 5.1) focus on parameter-level adaptations designed to align the retriever and generator modules. We primarily examine contrastive learning frameworks, architectural fusion techniques, and the emerging trend of using reinforcement learning.

Second, **Autonomous Agent Architectures** (Section 5.2) shift the paradigm from passive pipelines to dynamic workflows. To analyze these systems, we propose a three-tier classification framework based on autonomy. This allows RACG systems to refine decisions based on execution feedback.

5.1 Training-Driven Optimizations

RACG systems can be implemented in a lightweight manner, as work such as DraCo (Cheng et al., 2024) and RepoGraph (Ouyang et al., 2025b) already showed promising results. Some research, however, increasingly treats training as essential for performance. Most approaches are self-supervised or unsupervised, leveraging the inherent structure of code in the absence of labeled data.

5.1.1 Retrieval module training

Contrastive learning and representation alignment. Most methods train retrievers to align queries and code in a shared space. ReACC (Lu et al., 2022) combines BM25 with a dense retriever and applies contrastive pre-training with code augmentations. InferFix (Jin et al., 2023) aligns buggy code with fixes, while CodeGenAPI (Zan et al., 2025) pairs natural language queries with API documentation. CodeXEmbed (Liu et al., 2025d) unifies multiple tasks in a large-scale contrastive framework. SweRank (Reddy et al., 2025) and SweRank+ (Wang et al., 2025f) optimize retrieval with InfoNCE loss and hard negatives. NEMOTRON-CORTEXA (Sohrabizadeh et al., 2025) generates synthetic query-code pairs, while CONAN-R (Li et al., 2025h) enhances alignment with code–document and another mask entity prediction task.

Structure-aware optimization. Some methods incorporate code structure during training. kNN-TRANX (Zhang et al., 2023b) maps natural language to abstract syntax trees with auxiliary networks for reliability. CodeGRAG (Du et al., 2025) jointly encodes textual and structural views using graph neural networks with structure-preserving learning.

Generation-driven and likelihood-based objectives. Other approaches optimize retrieval based on its impact on generation. CoRet (Fehr et al., 2025) introduces a likelihood-based loss using call graph and file path signals. CodeFilter (Li et al., 2025i) scores retrieved blocks with special tokens. RepoShapley (Huo et al., 2026) models interactions among retrieved contexts via Shapley values. RLPG (Shrivastava et al., 2023b) selects contexts with a trained classifier, while SelfRACG (Dong et al., 2025a) integrates retrieval directly into the model's hidden states.

Multi-stage pipelines and re-ranking. To balance scale and precision, several methods adopt coarse-to-fine retrieval. SWE-Fixer (Xie et al., 2025) combines BM25 with a fine-tuned LLM model, SweRank (Reddy et al., 2025) adds LLM re-ranking, and CodeRAG (Zhang et al., 2025c) distills re-rankers from stronger models.

5.1.2 Generation module training

Context augmentation and supervised fine-tuning. The most direct approach concatenates retrieved content with the input prompt. ReACC (Lu et al., 2022) and InferFix (Jin et al., 2023) apply supervised fine-tuning (SFT) on augmented inputs, while R^2C^2-Coder (Deng et al., 2024) and RepoFusion (Shrivastava et al., 2023a) extend this idea by incorporating richer structural contexts and task alignment objectives.

Architectural fusion and attention mechanisms. Beyond prompt concatenation, several methods modify architectures for deeper integration. Compression-based approaches such as CoCoMic (Bi et al., 2024), CoRoVA (Cherniuk et al., 2026), and HEF (Sorokin et al., 2026) reduce retrieved content into compact representations for efficient attention. Graph-based methods, including CodeRCSG (Jiang et al., 2024), GRACE (Wang et al., 2025c), and CGM (Tao et al., 2025), jointly model structural relationships via GNNs or subgraph pre-training. Other designs, such as CONAN-G (Li et al., 2025h), leverage dual-view representations and FiD architectures to enhance semantic fusion.

Reasoning trajectories and distillation. To improve performance on complex tasks, some works incorporate reasoning supervision. SWE-Fixer (Xie et al., 2025) uses chain-of-thought (CoT) training for patch generation, while LocAgent (Chen et al., 2025e) distills planning trajectories from larger models into smaller ones. Other approaches explicitly model developer intent by generating intermediate representations (e.g., docstrings) to guide reasoning (Li et al., 2026b).

Advanced training paradigms and domain adaptation. More advanced strategies refine training objectives and curricula. RepoFormer (Wu et al., 2024) learns when to do retrieval, while CMFT (Sagtani et al., 2025) adopts curriculum learning for robustness. SRI (Zhang et al., 2026a) reframes completion as dynamic editing, and domain-specific tuning is explored in DroidCoder (Yu et al., 2024b) and RTLRepoCoder (Wu et al., 2025).

Emerging Direction: Reinforcement learning for retrieval and generation alignment.

To bridge the gap between retrieval and generation objectives, reinforcement learning (RL) is increasingly adopted. Methods such as RLCoder (Wang et al., 2025e) and AlignCoder (Jiang et al., 2026) optimize retrievers using generator feedback (e.g., perplexity), while SemanticForge (Zhang et al., 2025d) trains query planners with REINFORCE. Two-stage pipelines combine SFT with preference optimization (e.g., PPO in RRG (Gao et al., 2024a), DPO in CoLA (Li et al., 2025e)) to improve alignment and reduce hallucination. RL also powers autonomous code agents such as RepoSearcher (Ma et al., 2025c), RepoNavigator (Zhang et al., 2026b), and CodeScout (Sutawika et al., 2026). Even outside RAG, works like SoRFT (Ma et al., 2025b) and CoLT (Li et al., 2025f) highlight RL's broader impact on improving code generation quality.

Takeaways

Overall, training is crucial for RACG performance, but must align with task requirements and supervision. Key principles include:

- **Separate signals for retrieval and generation.** Fine-tuning the generator alone is often insufficient; retrieval-specific or joint objectives are needed when retrieval quality is the bottleneck.
- **Contrastive learning dominates retrieval training.** It effectively exploits code structure (e.g., transformations, bug–fix pairs) without costly annotation.
- **Reinforcement learning enables alignment.** It helps models learn when and how to use retrieved context, especially in multi-hop or ambiguous scenarios.

5.2 Autonomous Agent Architectures

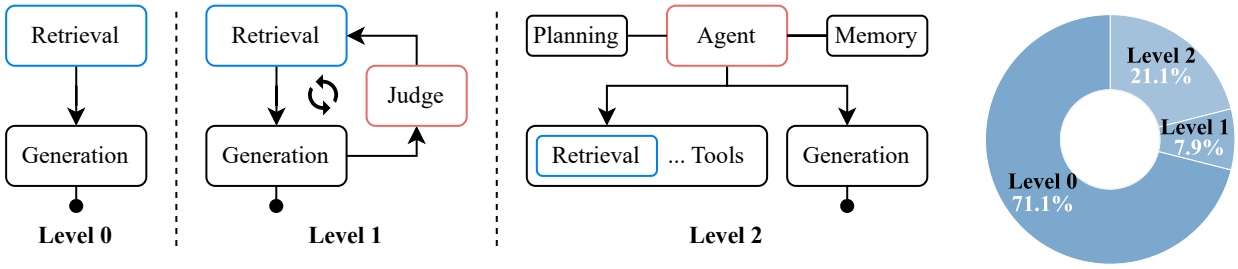

Figure 7: Three-tier classification of agent-based architectures in RACG (left), ranging from non-agent systems (Level 0) to partial agent systems (Level 1) and fully autonomous agents (Level 2). The distribution of systems across levels is illustrated in the pie chart (right).

Although recent RACG methods improve retrieval and context construction, many still operate as static, one-shot pipelines that cannot revise decisions based on intermediate outcomes. This limitation is especially evident in repository-level tasks (e.g., bug fixing and multi-file editing), which require iterative exploration, error correction, and interaction with external signals.

Agent-based architectures address this by enabling iterative decision-making, structured planning, and environment interaction, transforming RACG into dynamic, feedback-driven workflows. This allows models to decompose tasks, adapt retrieval and generation, and recover from errors. Despite broad adoption across

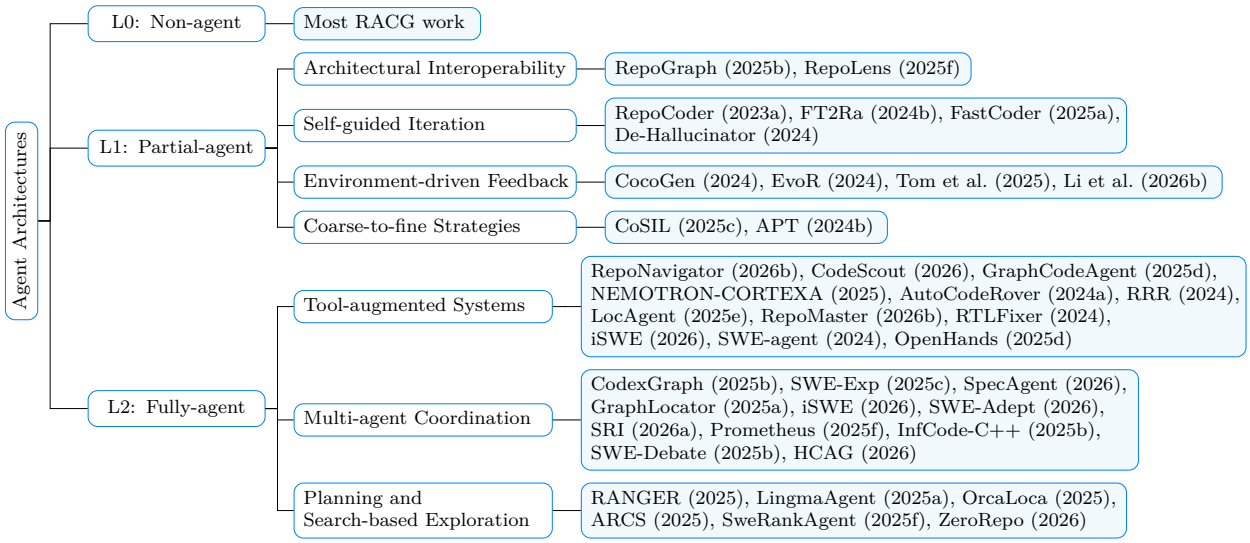

Figure 8: Taxonomy of Agent Architectures.

domains, the definition of an "agent" remains inconsistent. In this work, *we define an agent as a system that perceives its environment, makes decisions via planning or learned policies, and iteratively executes actions to achieve goals without rigid control flow.*

To analyze agent-based RACG systems, we propose a three-tier taxonomy (Figure 7 and 8) based on autonomy:

- **Level 0: Non-agent systems**—Static pipelines without iterative feedback, where retrieval and generation are performed in a single pass.
- **Level 1: Partial agent systems**—Systems with an explicit iterative feedback loop, where intermediate outputs (e.g., generated code, execution signals, or retrieved context) are used to refine subsequent decisions, but without autonomous planning or tool orchestration.
- **Level 2: Fully autonomous agents**—Systems that perform goal-directed reasoning with autonomous planning, multi-step decision-making, and interaction with external tools or environments.

5.2.1 Level 0: Non-agent Systems

Most RACG approaches fall under Level 0, relying on fixed pipelines without dynamic decision-making. While effective for goal-driven tasks, this design lacks adaptivity for complex, multi-step repository problems. Agentless (Xia et al., 2025) exemplifies this paradigm with a three-phase pipeline: hierarchical localization (repository $\rightarrow$ file $\rightarrow$ lines), patch generation, and regression-based validation. By avoiding agentic behavior, it achieves strong interpretability and low computational cost, while still reaching competitive performance (e.g., 32% on SWE-bench Lite).

Notably, most training-intensive RACG systems also remain at this level, as retrievers and generators are often optimized independently and cannot adapt during inference, creating a gap between training-time optimization and runtime flexibility.

5.2.2 Level 1: Partial Agent Systems

Level 1 systems introduce iterative feedback loops, where intermediate signals (e.g., partial outputs, compiler feedback, or retrieved context) refine subsequent steps. Unlike Level 2, they do not require full planning or autonomous control. Architectural interoperability with agent frameworks, such as RepoGraph (Ouyang et al., 2025b) and RepoLens (Wang et al., 2025f), does not, by itself, constitute agentic behavior; such systems are therefore more appropriately classified as Level 1.

The other Level 1 systems can be grouped into three patterns. First, **self-guided iteration**, where model outputs guide retrieval, as in RepoCoder (Zhang et al., 2023a), FT2Ra (Guo et al., 2024b), and FastCoder (Zhao et al., 2025a). De-Hallucinator (Eghbali & Pradel, 2024) further filters hallucinated APIs during iteration. Second, **environment-driven feedback**, where external signals guide refinement. CocoGen (Bi et al., 2024) uses compiler feedback, EvoR (Su et al., 2024) incorporates execution and web signals, and other works refine outputs using domain constraints or human feedback (Tony et al., 2025; Li et al., 2026b). Third, **coarse-to-fine strategies** progressively narrow the search space. CoSIL (Jiang et al., 2025c) moves from file-level to function-level reasoning, while APT (Zhang et al., 2024b) prioritizes simpler dependencies before complex ones.

A key challenge is stability: iterative loops may introduce error accumulation or semantic drift, requiring better control mechanisms or human-in-the-loop designs (Zhang et al., 2023a).

5.2.3 Level 2: Fully Autonomous Agent Frameworks

A growing number of RACG systems fall under Level 2, featuring fully agentic architectures with autonomous planning, tool usage, and multi-step reasoning. Unlike Level 1 systems that rely on predefined feedback loops, these systems act as goal-directed agents that dynamically explore and operate within repository environments. Existing approaches can be grouped into three main design patterns.

Tool-augmented systems. These systems rely on structured tools to navigate repositories, spanning multiple levels of abstraction. At the command level, RepoNavigator (Zhang et al., 2026b) uses a *jump* tool for symbol retrieval, while CodeScout (Sutawika et al., 2026) operates purely via Unix terminal commands. At the software engineering level, tools based on Abstract Syntax Trees (ASTs) and the Language Server Protocol (LSP) enable precise static analysis, as seen in RRR (Deshpande et al., 2024), NEMOTRON-CORTEXA (Sohrabizadeh et al., 2025), and AutoCodeRover (Zhang et al., 2024a). Graph-based systems such as GraphCodeAgent (Li et al., 2025d), LocAgent (Chen et al., 2025e), and RepoMaster (Wang et al., 2026b) introduce traversal tools for repository graphs. Domain-specific frameworks include RTLFixer (Tsai et al., 2024) (Verilog compiler) and iSWE (Ganhotra et al., 2026) (CLDK for Java analysis). At the most general level, SWE-agent (Yang et al., 2024) defines an Agent–Computer Interface, while OpenHands (Wang et al., 2025d) provides a unified environment for command execution, browsing, and editing.

Multi-agent coordination. These systems decompose tasks across multiple specialized agents. Role-based designs separate responsibilities, e.g., CodexGraph (Liu et al., 2025b) (natural language vs. Cypher query generation) and SWE-Exp (Chen et al., 2025c) (Instructor vs. Assistant). SpecAgent (Ma et al., 2026) combines Retriever and Forecaster agents. Task-level decomposition is also common, particularly separating localization and editing, as in GraphLocator (Liu et al., 2025a), iSWE (Ganhotra et al., 2026), SWE-Adept (He & Roy, 2026), and SRI (Zhang et al., 2026a). More complex pipelines include Prometheus (Chen et al., 2025f) (knowledge graph integration) and InfCode-C++ (Dong et al., 2025b) (Reproducer, Patch, Selector agents). Beyond pipelines, SWE-Debate (Li et al., 2025b) introduces structured debate, while HCAG (Wu & Deng, 2026) employs cooperative game-inspired collaboration.

Planning and search-based exploration. These systems explicitly explore large search spaces. Tree-based methods such as RANGER (Shah et al., 2025) and LingmaAgent (Ma et al., 2025a) apply Monte Carlo Tree Search (MCTS). Efficiency-oriented designs include OrcaLoca (Yu et al., 2025) with action scheduling and context pruning and ARCS (Bhattarai et al., 2025) with plan-conditioned retrieval with tiered control. SweRankAgent (Wang et al., 2025f) structures exploration into search, reasoning, reformulation, and aggregation, while ZeroRepo (Luo et al., 2026) applies an explore–exploit strategy over a feature tree for scalable codebase generation.

A defining feature of Level 2 systems is their ability to enable open-ended exploration and adaptive decision-making. However, this flexibility introduces challenges in efficiency, controllability, and reliability at scale.

Emerging Direction: Agentic Search Instead of Traditional RAG

The shift from traditional RAG to Agentic Search reflects a move from passive similarity matching to active exploration. Claude Code (Anthropic, 2025a) exemplifies this paradigm by replacing static embeddings with dynamic tool usage. Instead of querying a fixed index, it leverages terminal utilities such as `grep`, `ls`, and `cat` within a continuous Plan–Act–Observe loop. This grounds reasoning in the live filesystem state, improving precision and enabling real-time dependency tracking. As discussed in Section 4.1.2, this aligns with command-driven context retrieval, and is increasingly adopted in systems such as CodeScout (Sutawika et al., 2026).

Takeaways

RACG agent architectures span a broad spectrum: Level 0 systems offer simplicity and strong baselines; Level 1 introduces iterative feedback for moderate adaptivity; and Level 2 enables full autonomy through planning, tools, and collaboration. Bridging the gap between Level 0 efficiency and Level 2 capability remains a key research challenge.

6 Downstream Tasks and Benchmarks for RLCG

Since RACG is a broad topic, the benchmark landscape is highly fragmented, with many heterogeneous and loosely defined tasks. To ensure clarity and reliability, we focus only on formally defined benchmarks that are directly relevant to RLCG, centering on retrieval-augmented code generation rather than loosely related problems.

6.1 Downstream Tasks

Before the emergence of RLCG, code intelligence research had already explored retrieval-centric tasks such as **code search** (Yin et al., 2018), which evaluates a model's ability to retrieve or synthesize code from natural language queries, as well as **bug or vulnerability detection** (Li et al., 2018), which emphasizes identifying issues using retrieved evidence. With the rise of large language models, RLCG shifts the focus toward stronger *generation* capabilities, integrating retrieval to support reasoning and synthesis rather than serving as the primary objective.

RLCG systems have since been applied to several key downstream tasks. The most prominent are **cross-file code completion** and **GitHub issue resolution**, both discussed in Section 2.1. These tasks reflect realistic software engineering scenarios that require reasoning beyond a single file: the former predicts missing code using repository context, while the latter automates bug fixing or feature implementation based on issue descriptions.

Another important setting is **coding ability evaluation**, represented by benchmarks such as HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021). Although less grounded in real-world workflows, these benchmarks provide standardized environments to assess whether RLCG improves reasoning and code synthesis under external knowledge augmentation, and enable controlled comparisons with other techniques such as instruction tuning or chain-of-thought prompting.

Beyond these, RLCG has also been applied to **program repair** (Berabi et al., 2021) and **test generation** (Alshahwan et al., 2024), which focuses on synthesizing unit tests or assertions. Emerging directions further include **theory-to-code translation** (Wu & Deng, 2026) and **repository-level monitoring, editing, and refactoring** (Bairi et al., 2024). Together, these tasks highlight the versatility of RLCG across multiple dimensions of software engineering.

6.2 Benchmarks

We now turn to benchmarks for evaluating RLCG systems. Many works rely on self-constructed benchmarks, as they are easy to build and can highlight specific model strengths. However, this practice hinders fair and

consistent comparison across systems. To ensure rigor, we include only benchmarks adopted by multiple works and distinguish two roles:

- **RLCG-direct benchmarks**: Designed for repository-level or cross-file generation, where retrieval over a codebase is intrinsic. These provide the primary evidence for evaluating RLCG systems.
- **General-purpose coding benchmarks**: Standalone tasks (e.g., algorithmic problems, docstring-to-code) that do not require repository context. These serve as *auxiliary evidence* for assessing base LLM capability, but not RLCG effectiveness.

Representative benchmarks used in RLCG evaluation are summarized in Table 4.

Table 4: Overview of representative benchmarks.

Category	Benchmark	Size	Programming Language	Date	Link
Line Completion	RepoEval (Zhang et al., 2023a)	3573	Python	2023-03	link
	RepoBench (Liu et al., 2024b)	49684[3]	Python, Java	2024-01	link
	CrossCodeEval (Ding et al., 2023)	9928	Python, Java, TypeScript, and C#	2023-11	link
Function Generation	CoderEval (Yu et al., 2024a)	460	Python, Java	2024-02	link
	DevEval (Li et al., 2024b)	1874	Python	2024-05	link
	EvoCodeBench (Li et al., 2025c)	275	Python	2024-03	link
Real-World Resolution	SWE-bench (Jimenez et al., 2024)	2294	Python	2023-10	link
	Long Code Arena (Bogomolov et al., 2024)	—[4]	Python, Java, Kotlin	2024-06	link
General Purpose	Aider Polyglot (Team, 2024)	225	Multi-language	2024-12	link
	LiveCodeBench (Jain et al., 2025)	300+	Python	2024-03	link
	HumanEval (Chen et al., 2021)	164	Python	2021-07	link
	MBPP (Austin et al., 2021)	974	Python	2021-08	link
	CodeXGLUE (Lu et al., 2021)	—[5]	Multi-language	2021-03	link

Repository-Level Line Completion. **RepoEval** (Zhang et al., 2023a) (from RepoCoder) includes 1,600 line completions, 1,600 API calls, and 373 function completions from high-quality Python repositories (2022+). It evaluates functional correctness using repository-native unit tests and snapshots (Jan 2023) across line, API, and function levels. **RepoBench** (Liu et al., 2024b) supports Python and Java, decomposing RLCG into retrieval (RepoBench-R), completion (RepoBench-C), and pipeline (RepoBench-P) tasks. It uses newly crawled GitHub data and evaluates retrieval with *Accuracy@k* and completion with *EM/ES*. **CrossCodeEval** (Ding et al., 2023) provides 10,000 cross-file completion examples across Python, Java, TypeScript, and C#, using static analysis for import resolution and evaluating in-file, cross-file retrieval, and combined prompt settings.

Repository-Level Function Generation. **CoderEval** (Yu et al., 2024a) contains 460 real-world problems (Python/Java) across six context levels, focusing on non-standalone functions. It standardizes inputs as "signature + doc + context" and evaluates with Docker-based `Pass@k` and `Acc@k`. **EvoCodeBench** (Li et al., 2025c) offers 275 Python samples aligned with evolving repositories, incorporating dependency structures and updates; tasks combine natural language requirements, signatures, references, and tests, evaluated by `Pass@k` and `Recall@k`. **DevEval** (Li et al., 2024b) includes 1,874 manually annotated samples from 117 repositories across 10 domains, requiring function generation with signatures, natural language requirements, and cross-file context.

Pragmatic and Real-World Issue Resolution. **SWE-bench** (Jimenez et al., 2024) contains 2,294 real-world issues from 12 Python repositories, where models generate patches validated by tests; variants

[3]Calculated based on data from RepoBench-Python and RepoBench-Java on Hugging Face.
[4]Long Code Arena encompasses 6 distinct tasks, making it inappropriate to compute the overall size of the benchmark.
[5]CodeXGLUE encompasses 10 distinct tasks from 14 datasets, making it inappropriate to compute the overall size.

include SWE-bench-lite, -verified, -Pro, -multilingual, and -multimodal (Yang et al., 2025b). **Long Code Arena** (Bogomolov et al., 2024) consists of six long-context benchmarks for project-level tasks such as bug localization and library-based generation, with curated datasets and standardized evaluation tools.

General-Purpose Coding Benchmarks (Auxiliary Evidence). These benchmarks are not designed for repository-level or cross-file tasks but are often used to assess base LLM performance or compare augmentation strategies. They provide *indirect evidence* for RACG effectiveness. **Polyglot Leaderboard** (Team, 2024) evaluates multilingual coding under a unified `pass@k` metric. **LiveCodeBench** (Jain et al., 2025) includes 300+ recent problems (2023–2024) for code generation, self-repair, and execution. **HumanEval** (Chen et al., 2021) offers 164 Python problems with signatures, docstrings, and tests for `pass@k` evaluation. **MBPP** (Austin et al., 2021) contains 974 beginner-level Python problems described in natural language. **CodeXGLUE** (Lu et al., 2021) is a suite of 10 tasks across 14 datasets, covering translation, completion, and more, with unified evaluation and baselines.

Takeaways

RLCG evaluation is shifting toward repository-level benchmarks that require integrating retrieval with generation (e.g., cross-file completion and issue resolution). General-purpose coding benchmarks offer only indirect evidence, making task-aligned, realistic evaluation increasingly important.

7 Configurations for RACG Systems

This section summarizes key configurations that shape RACG performance and scope. **Data Sources** (Section 7.1) highlight the shift from repository-local inputs to diverse, task-specific signals. **Programming Language Support** (Section 7.2) reviews the Python-centric landscape and motivates cross-lingual frameworks for real-world use. **Backbone Models** (Section 7.3) discusses model choices for retrieval and generation, along with their trade-offs.

7.1 Data Source

In RACG, retrieval typically operates over the target repository, sometimes augmented with API documentation (Zhang et al., 2023a; Cheng et al., 2024; Gu et al., 2025; Deng et al., 2025). Recent work broadens this scope with task-specific resources, which can be grouped into four categories.

General-purpose and web corpora. Some methods incorporate external programming-related text. PKG (Saberi & Fard, 2025) uses Q&A datasets and tutorials as general knowledge, while EvoR (Su et al., 2024) integrates heterogeneous web content (e.g., blogs, discussions, documentation, and execution feedback) into a unified, evolving knowledge base.

Domain knowledge. Other approaches leverage specialized corpora or constraints. DeepV (Ibnat et al., 2025) reuses VerilogDB (Calzada et al., 2025) for hardware tasks. RTLFixer (Tsai et al., 2024) retrieves expert explanations for syntax errors, while CodeGuarder (Lin et al., 2025) and SecGuide-based methods (Tony et al., 2025) incorporate vulnerability patterns and secure-coding rules.

Repository-level structural and relational knowledge. Several systems exploit relations within or across repositories. A^3-CodGen (Liao et al., 2024) links local context with global module structures and libraries. Abedu et al. (2024) construct repository knowledge graphs from commits, issues, and files. RepoMaster (Wang et al., 2026b) builds multi-level graphs (e.g., call graphs, dependency graphs) across repositories and retrieves task-relevant components.

Experience and behavior traces. Another line leverages past interaction traces. ContextModule (Guan et al., 2024) mines real editing behavior, SWE-Exp (Chen et al., 2025c) stores repair trajectories, and ReCode (Wang et al., 2025c) collects buggy–fixed pairs. Beyond code, Kranti et al. (2024) use human interaction traces (e.g., collaborative building actions) as retrieval signals.

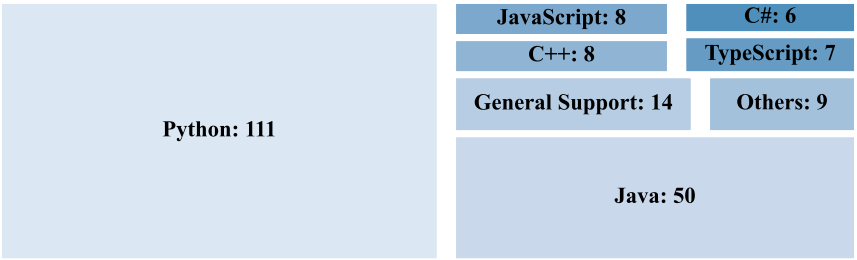

Figure 9: Programming Language Distribution

Takeaways

RACG data sources are evolving from repository-local context to diverse external signals, including web corpora, domain knowledge, structural graphs, and behavioral traces. This diversification improves task performance but also highlights data selection as a key, under-explored design factor (Wang et al., 2025g).

7.2 Programming Language Support

We analyze programming language usage in RACG systems based on explicit mentions in prior work, including *cross-lingual transfer*. As shown in Figure 9, **Python** dominates (111 occurrences), likely due to its prevalence in AI and data science, flexible syntax, and strong ecosystem. **Java** follows (50), reflecting its role in enterprise systems. **C++** and **JavaScript** appear less frequently (8 each), while **TypeScript** and **C#** indicate continued interest in statically typed languages. Emerging languages such as **Rust**, **Go**, and **Kotlin** remain underrepresented.

This distribution shows a strong Python-centric bias in RACG research. While Python facilitates rapid prototyping, limited coverage of languages like Rust or Go may restrict generalization to real-world, multi-language codebases. Statically typed languages introduce additional challenges in structure, tokenization, and generation fidelity. As a result, bridging the gap between academic settings and heterogeneous industrial repositories remains an open problem, further compounded by benchmark bias (e.g., SWE-bench is Python-only).

Emerging Direction: Universal Language Support

Cross-language capability is increasingly important for RACG. `Zhu et al.` (Zhu et al., 2025) show that cross-language retrieval can improve multilingual code generation, enhancing robustness and generalization.
Many approaches map different languages into unified representations. `Code2JSON` (Singhal et al., 2025) converts code into structured natural language features via zero-shot LLMs. `CodeXEmbed` (Liu et al., 2025d) unifies retrieval tasks across text and code. `CodeRCSG` (Jiang et al., 2024) builds cross-lingual semantic graphs using GNNs, while `Prometheus` (Chen et al., 2025f) constructs unified knowledge graphs across languages.
Other works leverage the Language Server Protocol (LSP) for standardized multi-language support. `HCGS` (Sounthiraraj et al., 2025) uses a modified `multilspy` client, and `ChatLSP` (Blinn et al., 2024) provides unified contextual services across languages.

7.3 Backbone Models for Retrieval and Generation

We categorize and list the commonly used backbone models in RACG systems. We note that the models included here are representative rather than exhaustive, aiming to cover widely adopted examples across academic literature.

Dense Retrieval Models. Dense retrievers map queries and code into continuous vector spaces using neural encoders. **UniXcoder** (Guo et al., 2022) leverages ASTs and comments with prefix-adapters and contrastive learning for accurate embeddings. **CodeT5** (Wang et al., 2021) incorporates identifier-aware pre-training to better capture semantic signals. **Voyage-Code** (Voyage AI, 2024) improves multilingual retrieval via Matryoshka learning and quantization-aware training. **Stella** (Zhang et al., 2025a) uses multi-stage distillation for strong semantic representations. **GraphCodeBERT** (Guo et al., 2021) models data flow dependencies beyond syntax, while **CodeBERT** (Feng et al., 2020) is effective for NL–PL tasks such as search and documentation. **Ada-Embedding-002** (OpenAI, 2025b) provides general-purpose text–code embeddings with strong retrieval performance.

Sparse Retrieval Models. Sparse retrievers rely on lexical matching and simple similarity metrics. **BM25** (Robertson et al., 1995) is a TF–IDF–based ranking function with length normalization, serving as a strong keyword-based baseline. **Jaccard Similarity** (Jaccard, 1901) measures token overlap and is useful for capturing structural similarity between queries and code.

Table 5: Overview of selected open-source code generation models, including details such as the developing organization, model size (with some MoE models in the format of "full parameters (activation parameters)"), vocabulary size, context window length, total number of training tokens, and release date.

Model	Organization	Size	Vocab	Context	Tokens	Date
CodeGen (Nijkamp et al., 2023b)	Salesforce AI	350M, 2B, 6B, 16B	50K	2048	577.2B	2022-03
CodeGen2 (Nijkamp et al., 2023a)	Salesforce AI	1B, 3.7B, 7B, 16B	50K	2048	400B-1.4T	2023-03
SantaCoder (Allal et al., 2023)	BigCode	1.1B	48K	2048	236B	2023-01
StarCoder(Base) (Li et al., 2023)	BigCode	1B, 3B, 7B, 15.5B	48K	8192	1T	2023-05
StarCoder2 (Lozhkov et al., 2024)	BigCode	3B, 7B, 15B	48K	16K	4T	2024-02
Code LLaMA (Rozière et al., 2024)	Meta	7B, 13B, 34B, 70B	31K	16K-100K	500B-1T	2023-08
DeepSeek-Coder (Guo et al., 2024a)	DeepSeek	1.3B, 5.7B, 6.7B, 33B	32K	16K	2T	2024-01
DeepSeek-Coder-V2 (DeepSeek-AI et al., 2024)	DeepSeek	16B(2.4B), 236B(21B)	100K	16K-128K	6T	2024-06
CodeQwen1.5 (Bai et al., 2023)	Alibaba	7B	90K	64K	3T	2024-04
Qwen2.5-Coder (Hui et al., 2024)	Alibaba	0.5B, 1.5B, 3B, 7B, 14B, 32B	149K	32K-128K	5.5T	2024-11
Qwen3-Coder (Team, 2025)	Alibaba	30B(3B), 480B(35B)	148K	256K-1M	7.5T	2025-07

Open-source Generation Models. Representative open-source code generation models and their configurations are summarized in Table 5. **CodeGen** (Nijkamp et al., 2023b) is a decoder-only Transformer family (up to ∼16B) trained on multilingual NL–PL data, showing strong zero-shot and multi-turn synthesis performance. **SantaCoder** (Allal et al., 2023) is a compact 1.1B model trained on The Stack, leveraging Multi-Query Attention and Fill-in-the-Middle (FIM) for efficient inference and accurate infilling. **StarCoder** (Li et al., 2023) (∼15.5B) is trained on ∼1T tokens across 80+ languages, supporting 8K context and achieving competitive HumanEval performance. **Code LLaMA** (Rozière et al., 2024) extends LLaMA 2 for code (7B–34B), offering strong long-context completion and instruction-following. **DeepSeek-Coder** (Guo et al., 2024a) is trained on ∼2T tokens (87% code), with 16K context and strong multilingual completion and infilling. **Qwen2.5-Coder** (Hui et al., 2024) supports up to 32B parameters, 128K context, and ∼92 languages, excelling in generation, repair, and reasoning.

Proprietary Generation Models. Representative proprietary code generation models and their capabilities are summarized in Table 6. **GPT and o Series (e.g., GPT-5, GPT-5 mini, GPT-4o, o3)** (OpenAI, 2025a) from OpenAI lead in code generation and reasoning, with GPT-5 targeting complex, long-context tasks and the o-series focusing on efficient multi-step reasoning. **Claude Series (e.g., Claude Opus 4, Sonnet 4.5)** (Anthropic, 2025b) from Anthropic excels in structured reasoning and coding, with Opus 4 supporting long-running agentic workflows and Sonnet 4 offering strong performance at lower cost. **Gemini Series (e.g., Gemini 2.5 Pro, Flash-Lite)** (Google DeepMind, 2025) from Google DeepMind provides strong coding and multimodal reasoning, with up to 1M-token context, while Flash-Lite emphasizes efficiency.

Table 6: Overview of selected proprietary code generation models, including details such as the developing organization, context window length, cost, release date and official SWE-bench score.

Model	Organization	Context	Cost (Input/Output)	Date	SWE-bench
GPT-4o	OpenAI	128K	\$2.50/\$10.00	2024-05	21.62%
GPT-5	OpenAI	400K	\$1.25/\$10.00	2025-08	65.00%
GPT-5 mini	OpenAI	400K	\$0.25/\$2.00	2025-08	59.80%
o3	OpenAI	200K	\$2.00/\$8.00	2025-04	58.40%
Claude Opus 4	Anthropic	200K	\$15.00/\$75.00	2025-05	67.60%
Claude Sonnet 4	Anthropic	200K	\$3.00/\$15.00	2025-05	64.93%
Claude Sonnet 4.5	Anthropic	200K	\$3.00/\$15.00	2025-09	70.60%
Gemini 2.5 Pro	Google	1M	\$1.25/\$10.00	2025-03	53.60%
Gemini 2.5 Flash	Google	1M	\$0.30/\$2.50	2025-03	28.73%

* SWE-bench Verified score with mini-swe-agent
Costs shown are per million tokens (input/output) with no cache in USD
Context window shown as input token limit for Gemini Model(K = thousand, M = million)

Analysis. We also conducted a statistical analysis of the base generation models adopted in each paper, focusing on the models used in the recommended configurations. This distinction allows us to better understand the models authors truly relied on in their proposed systems. We categorized the models into three groups: (1) large open-source models (>10B parameters), (2) small open-source models (<10B parameters), and (3) proprietary models.

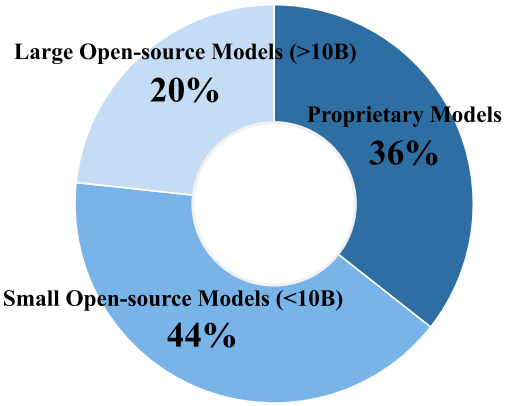

Figure 10: Distribution of base generation models used in the surveyed papers.

Figure 10 reveals that small open-source models and proprietary models appear with comparable frequency and are significantly more popular than large open-source models. The prevalence of small models may be due to limited computational resources in academic or exploratory research settings, or because they can already achieve satisfactory performance on many tasks. Meanwhile, proprietary models are widely adopted, potentially due to two key reasons: (i) their superior performance on complex reasoning and generation tasks, and (ii) their ease of integration via API, reducing engineering and infrastructure overhead. In contrast, large open-source models (>10B) are used far less frequently. This may stem from their high resource demands, including substantial GPU memory and inference time, which can pose barriers to adoption in both academic and industrial settings without dedicated infrastructure.

Takeaways

Backbone selection in RACG involves key trade-offs:

- **Retrieval vs. context window:** Dense retrieval scales to large codebases, while long-context models simplify pipelines but incur higher cost and latency.
- **Sparse vs. dense retrieval:** Sparse methods (BM25) are efficient for keyword matching, whereas dense retrievers better capture semantics at higher cost.
- **Model size vs. deployment:** Smaller open models are accessible, while proprietary models lead performance; practical use depends on task complexity and system constraints.

8 Discussion and Future Directions

This section synthesizes the critical perspectives of RACG. As the field transitions from static augmentation to autonomous engineering, we analyze the critical boundaries of security, the necessity of retrieval in the era of expanding context windows, and the remaining bottlenecks that define future research opportunities.

8.1 Security and Reliability

The integration of RAG into repository-level development introduces a unique attack surface spanning both the generation and retrieval modules. On the generation side, LLMs risk inheriting insecure coding patterns from their training data, which often contains vulnerable open-source projects. On the retrieval side, knowledge base poisoning is a major concern. For instance, ImportSnare (Ye et al., 2025) attack demonstrates that crafted malicious documentation can outrank safe resources, leading to the retrieval of harmful contexts (Ye et al., 2025). DrainCode (Wang et al., 2026e) introduces LLM-Denial-of-Service, where the model is manipulated into generating excessively verbose code to exhaust computational resources. Furthermore, VenomRACG (Li et al., 2025g) demonstrates that a poisoned retriever can be trained to prioritize pre-defined vulnerable code when triggered by specific malicious queries.

To mitigate these risks, several safety-driven architectures have emerged. CodeGuarder (Lin et al., 2025) proposes that retrievers should also inject security knowledge. Tony et al. (2025) combine security knowledge injection with a self-feedback loop. RESCUE (Shi & Zhang, 2026) introduces a systematic defense through a Hierarchical Security Knowledge Base and Hierarchical Multi-faceted Retrieval.

8.2 The Necessity Boundary: RAG vs. Long-Context LLMs

The effectiveness of RAG is context-dependent, and retrieval does not always guarantee better performance. PENG et al. (2025) show that long-context models can match or outperform RAG on small, well-structured repositories, while RAG becomes advantageous as repository size and complexity increase.

Several studies demonstrate the benefits of retrieval. CodeGen4Libs (Liu et al., 2023) validates the usefulness of retrieval for library-oriented generation through user studies. Chen et al. (2025a) show that RAG improves performance on uncommon APIs and remains robust to mild noise, with BM25 performing strongly. Yang et al. (2025c), Galimzyanov et al. (2025) and Hostnik & Robnik-Šikonja (2025) further report consistent gains from retrieval, especially when combined with effective fusion and selection strategies.

Other work explores replacing RAG with stronger long-context modeling. SelectSolve (Jiang et al., 2025b) shows that providing the full repository to a long-context model can rival or exceed agent-based systems. Additional approaches use hierarchical summarization, reinforcement learning, or tool-aware training (e.g., SoRFT (Ma et al., 2025b), CoLT (Li et al., 2025f), ToolGen (Wang et al., 2025b)) to improve context utilization without explicit retrieval.

Industry results also highlight trade-offs. Retrieval improves large-scale systems such as Tencent's WeChat codebase (Yang et al., 2025d), and combining RAG with fine-tuning yields further gains (Wang et al., 2025a). However, fine-tuned models can better capture domain-specific conventions ("dialects") (Finkler et al., 2026), suggesting limits to retrieval alone.

Takeaways

RAG is not universally superior. It excels in large or complex repositories, while long-context models can suffice in smaller settings. Effective systems often combine retrieval, model capacity, and fine-tuning, highlighting a key trade-off between context length, retrieval cost, and adaptability.

8.2.1 Limitations of Existing Approaches

Despite rapid progress, current RACG systems still face several important limitations in repository-level software engineering.

Retrieval and context construction remain difficult. Non-graph-based approaches often retrieve fragmented repository chunks that fail to preserve long-range semantic dependencies, while graph-based systems introduce substantial overhead in graph construction and traversal (Zhang et al., 2025e; Du et al., 2025; Wang et al., 2025c). Their effectiveness also heavily depends on language-specific parsers and static analysis quality, limiting portability across repositories and programming languages (Liu et al., 2024a; Cheng et al., 2024). In addition, retrieval and generation are still frequently optimized separately, creating a disconnect between retrieved context quality and downstream generation utility (Fehr et al., 2025; Dong et al., 2025a).

Agentic RACG systems often trade adaptivity for efficiency. Recent agent frameworks enable iterative search, planning, and tool usage, but they also introduce substantial context and computation overhead (Yang et al., 2024; Sutawika et al., 2026). Multi-step workflows may accumulate noisy context or redundant exploration steps as repositories scale (Chen et al., 2025f; Shah et al., 2025). At the same time, simpler pipelines such as Agentless can achieve competitive performance with significantly lower cost, suggesting that complex agentic workflows are not always necessary (Xia et al., 2025).

Current evaluation settings still lack realism and robustness. Most benchmarks focus on relatively small repositories, synthetic tasks, or Python-centric ecosystems, while repository evolution, collaborative workflows, and multi-language dependencies are rarely modeled explicitly (Jimenez et al., 2024). Recent studies further show that static benchmarks may suffer from contamination and benchmark saturation (OpenAI, 2026). Moreover, most RACG systems are still evaluated in isolated environments rather than integrated software engineering ecosystems such as IDEs, CI/CD pipelines, and collaborative development workflows (Yang et al., 2024; Wang et al., 2025d).

8.2.2 Future Directions

RACG systems should move beyond text-only repositories toward richer software artifacts. Future systems can leverage issue discussions, documentation, execution logs, commit histories, and architectural diagrams to improve semantic grounding in underspecified or evolving tasks (Yang et al., 2025a; Guan et al., 2024). At the same time, multilingual repository support remains increasingly important, as modern software projects often combine multiple programming languages and frameworks within a single codebase (Jiang et al., 2024; Yoo et al., 2025).

Scalable repository-level reasoning and realistic evaluation remain essential research directions. Recent work increasingly explores hierarchical retrieval, memory compression, and long-context reasoning to support repository-scale understanding with manageable computational cost (Bogomolov et al., 2024; Huo et al., 2026; Ma et al., 2026). Meanwhile, future benchmarks should better reflect practical software engineering workflows such as repository evolution, CI/CD integration, package migration, and collaborative development (Li et al., 2026a). Beyond pass@k or patch correctness, evaluation should also measure factors such as execution efficiency, integration consistency, maintainability, and developer utility (Jimenez et al., 2024).

Agentic software engineering workflows are becoming a central paradigm for RACG. Recent systems such as SWE-agent, OpenHands, Claude Code, and Codex demonstrate a shift from static retrieval-and-generation pipelines toward interactive software engineering agents capable of repository navigation, iterative refinement, and environment interaction (Yang et al., 2024; Wang et al., 2025d). Retrieval is increasingly treated as an adaptive test-time process, where agents dynamically decide what files or resources to inspect during problem solving. Future RACG systems may further support repository-wide coordinated editing tasks requiring continuous reasoning over inter-file dependencies and evolving repository states (Bairi et al., 2024; Wang et al., 2026b).

Bridging retrieval, reasoning, and generation remains the core challenge of RACG. Earlier code intelligence systems primarily focused on retrieval and matching, whereas modern LLM-based systems emphasize generative capability. Future RACG systems will likely move toward tighter integration between retrieval, memory, planning, reasoning, and generation, where retrieved repository artifacts dynamically guide and constrain generation decisions (Dong et al., 2025a; Zhang et al., 2025c; Chen et al., 2025f). Achieving

this integration is essential for building software engineering agents that are both repository-grounded and capable of adaptive, context-aware reasoning.

9 Conclusion

As software engineering increasingly shifts from isolated code completion toward repository-scale reasoning and autonomous development workflows, Retrieval-Augmented Code Generation (RACG) has emerged as a critical paradigm for enabling large language models to operate effectively in real-world environments. Unlike traditional code generation settings, repository-level scenarios require models to continuously retrieve, organize, and reason over evolving contextual information distributed across files, dependencies, documentation, and execution feedback.

In this survey, we presented a comprehensive review of RACG research with a particular focus on Repository-Level Code Generation (RLCG). We argued that RACG should not be viewed as a static "retrieve-then-generate" pipeline, but rather as a coupled and dynamic process involving context construction, retrieval optimization, generation, and environment interaction. To systematically analyze this rapidly evolving landscape, we introduced a unified analytical framework spanning retrieval substrate, control regime, and evaluation setting. Based on this framework, we organized existing methods across non-graph-based and graph-based retrieval paradigms, training-driven optimizations, and autonomous agent architectures, while also reviewing downstream tasks, benchmarks, and practical system configurations.

Our analysis reveals several emerging trends in the field. First, RACG systems are evolving from similarity-based retrieval toward structure-aware and graph-driven reasoning. Second, retrieval is increasingly becoming an adaptive and agentic process rather than a fixed preprocessing step. Third, autonomous software engineering agents are transforming RACG from passive generation pipelines into interactive systems capable of planning, tool usage, iterative refinement, and environment-aware decision making. At the same time, important challenges remain unresolved, including scalability, retrieval reliability, long-context efficiency, benchmark realism, cross-language portability, and robustness in dynamic repositories.

Finally, we highlight the growing importance of understanding the necessity boundary between retrieval and long-context modeling. While long-context LLMs continue to improve rapidly, repository-level software engineering still fundamentally depends on efficient context selection, structured reasoning, and adaptive interaction with external environments. We believe future RACG systems will increasingly integrate retrieval, reasoning, planning, memory, and execution into unified agentic software engineering frameworks.

We hope this survey provides a structured foundation for future research and helps bridge the gap between academic advances in code generation and practical AI-powered software engineering systems.

References

[1] Samuel Abedu, SayedHassan Khatoonabadi, and Emad Shihab. Synergizing llms and knowledge graphs: A novel approach to software repository-related question answering, 2024. URL `https://arxiv.org/abs/2412.03815`.

[2] Lakshya A Agrawal, Aditya Kanade, Navin Goyal, Shuvendu K. Lahiri, and Sriram K. Rajamani. Monitor-guided decoding of code lms with static analysis of repository context. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

[3] Loubna Ben Allal, Raymond Li, Denis Kocetkov, et al. Santacoder: don't reach for the stars!, 2023. URL `https://arxiv.org/abs/2301.03988`.

[4] Nadia Alshahwan, Jubin Chheda, Anastasia Finogenova, Beliz Gokkaya, Mark Harman, Inna Harper, Alexandru Marginean, Shubho Sengupta, and Eddy Wang. Automated unit test improvement using large language models at meta. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, FSE 2024, pp. 185–196, New York, NY, USA, 2024.

Association for Computing Machinery. ISBN 9798400706585. doi: 10.1145/3663529.3663839. URL https://doi.org/10.1145/3663529.3663839.

[5] Anthropic. Claude code: Deep coding at terminal velocity. https://www.anthropic.com/claude-code, August 2025a. Anthropic official website.

[6] Anthropic. Write beautiful code, ship powerful products. https://www.anthropic.com/solutions/coding, August 2025b. Anthropic official website, "Solutions – Coding" page.

[7] Yannick Assogba and Donghao Ren. Evaluating long range dependency handling in code generation LLMs. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=yzACI2vFaX.

[8] Mihir Athale and Vishal Vaddina. Knowledge graph based repository-level code generation. In *2025 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code)*, pp. 169–176. IEEE, May 2025. doi: 10.1109/llm4code66737.2025.00026. URL http://dx.doi.org/10.1109/LLM4Code66737.2025.00026.

[9] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

[10] Jinze Bai, Shuai Bai, Yunfei Chu, et al. Qwen technical report, 2023. URL https://arxiv.org/abs/2309.16609.

[11] Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, Vageesh D. C., Arun Iyer, Suresh Parthasarathy, Sriram Rajamani, B. Ashok, and Shashank Shet. Codeplan: Repository-level coding using llms and planning. *Proc. ACM Softw. Eng.*, 1(FSE), July 2024. doi: 10.1145/3643757. URL https://doi.org/10.1145/3643757.

[12] Berkay Berabi, Jingxuan He, Veselin Raychev, and Martin T. Vechev. Tfix: Learning to fix coding errors with a text-to-text transformer. In *ICML*, 2021.

[13] Manish Bhattarai, Miguel Cordova, Javier Santos, and Dan O'Malley. Arcs: Agentic retrieval-augmented code synthesis with iterative refinement, 2025. URL https://arxiv.org/abs/2504.20434.

[14] Zhangqian Bi, Yao Wan, Zheng Wang, Hongyu Zhang, Batu Guan, Fangxin Lu, Zili Zhang, Yulei Sui, Hai Jin, and Xuanhua Shi. Iterative refinement of project-level code context for precise code generation with compiler feedback. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 2336–2353, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.138. URL https://aclanthology.org/2024.findings-acl.138/.

[15] Andrew Blinn, Xiang Li, June Hyung Kim, and Cyrus Omar. Statically contextualizing large language models with typed holes. *Proc. ACM Program. Lang.*, 8(OOPSLA2), October 2024. doi: 10.1145/3689728. URL https://doi.org/10.1145/3689728.

[16] Egor Bogomolov, Aleksandra Eliseeva, Timur Galimzyanov, et al. Long code arena: a set of benchmarks for long-context code models, 2024. URL https://arxiv.org/abs/2406.11612.

[17] Max Brunsfeld et al. Tree-sitter: An incremental parsing system for programming tools, 2013. URL https://github.com/tree-sitter/tree-sitter. Accessed: 2026-02-16.

[18] Tuan-Dung Bui, Duc-Thieu Luu-Van, Thanh-Phat Nguyen, Thu-Trang Nguyen, Son Nguyen, and Hieu Dinh Vo. Rambo: Enhancing rag-based repository-level method body completion, 2024. URL https://arxiv.org/abs/2409.15204.

[19] Paul E. Calzada, Zahin Ibnat, Tanvir Rahman, Kamal Kandula, Danyu Lu, Sujan Kumar Saha, Farimah Farahmandi, and Mark Tehranipoor. Verilogdb: The largest, highest-quality dataset with a preprocessing framework for llm-based rtl generation, 2025. URL https://arxiv.org/abs/2507.13369.

[20] Jingyi Chen, Songqiang Chen, Jialun Cao, Jiasi Shen, and Shing-Chi Cheung. When llms meet api documentation: Can retrieval augmentation aid code generation just as it helps developers?, 2025a. URL https://arxiv.org/abs/2503.15231.

[21] Junkai Chen, Xing Hu, Zhenhao Li, Cuiyun Gao, Xin Xia, and David Lo. Code search is all you need? improving code suggestions with code search. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400702174. doi: 10.1145/3597503.3639085. URL https://doi.org/10.1145/3597503.3639085.

[22] Kang Chen, Xiuze Zhou, Yuanguo Lin, Shibo Feng, Li Shen, and Pengcheng Wu. A survey on privacy risks and protection in large language models, 2025b. URL https://arxiv.org/abs/2505.01976.

[23] Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

[24] Silin Chen, Shaoxin Lin, Xiaodong Gu, Yuling Shi, Heng Lian, Longfei Yun, Dong Chen, Weiguo Sun, Lin Cao, and Qianxiang Wang. Swe-exp: Experience-driven software issue resolution, 2025c. URL https://arxiv.org/abs/2507.23361.

[25] Xiaohan Chen, Zhongying Pan, Quan Feng, Yu Tian, Shuqun Yang, Mengru Wang, Lina Gong, Yuxia Geng, Piji Li, and Xiang Chen. Saracoder: Orchestrating semantic and structural cues for profit-oriented repository-level code completion, 2025d. URL https://arxiv.org/abs/2508.10068.

[26] Zhaoling Chen, Robert Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor Prasanna, Arman Cohan, and Xingyao Wang. LocAgent: Graph-guided LLM agents for code localization. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8697–8727, Vienna, Austria, July 2025e. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.426. URL https://aclanthology.org/2025.acl-long.426/.

[27] Zimin Chen, Yue Pan, Siyu Lu, Jiayi Xu, Claire Le Goues, Martin Monperrus, and He Ye. Prometheus: Unified knowledge graphs for issue resolution in multilingual codebases, 2025f. URL https://arxiv.org/abs/2507.19942.

[28] Wei Cheng, Yuhan Wu, and Wei Hu. Dataflow-guided retrieval augmentation for repository-level code completion. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7957–7977, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.431. URL https://aclanthology.org/2024.acl-long.431/.

[29] Daria Cherniuk, Nikita Sukhorukov, Danil Gusak, Nikita Sushko, Danil Sivtsov, Elena Tutubalina, and Evgeny Frolov. Corova: Compressed representations for vector-augmented code completion, 2026. URL https://arxiv.org/abs/2510.19644.

[30] Andy Clark and David Chalmers. The extended mind. *Analysis*, 58(1):7–19, 1998. ISSN 00032638, 14678284. URL http://www.jstor.org/stable/3328150.

[31] DeepSeek-AI, Qihao Zhu, Daya Guo, et al. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence, 2024. URL https://arxiv.org/abs/2406.11931.

[32] Ken Deng, Jiaheng Liu, He Zhu, et al. R2c2-coder: Enhancing and benchmarking real-world repository-level code completion abilities of code large language models, 2024. URL https://arxiv.org/abs/2406.01359.

[33] Le Deng, Xiaoxue Ren, Chao Ni, Ming Liang, David Lo, and Zhongxin Liu. Enhancing project-specific code completion by inferring internal api information. *IEEE Transactions on Software Engineering*, pp. 1–17, 2025. doi: 10.1109/TSE.2025.3592823.

[34] Ajinkya Deshpande, Anmol Agarwal, Shashank Shet, Arun Iyer, Aditya Kanade, Ramakrishna Bairi, and Suresh Parthasarathy. Class-level code generation from natural language using iterative, tool-enhanced reasoning over repository, 2024. URL https://arxiv.org/abs/2405.01573.

[35] Yangruibo Ding, Zijian Wang, Wasi Uddin Ahmad, et al. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

[36] Yangruibo Ding, Zijian Wang, Wasi Ahmad, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. CoCoMIC: Code completion by jointly modeling in-file and cross-file context. In Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue (eds.), *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 3433–3445, Torino, Italia, May 2024. ELRA and ICCL. URL https://aclanthology.org/2024.lrec-main.305/.

[37] Qian Dong, Jia Chen, Qingyao Ai, Hongning Wang, Haitao Li, Yi Wu, Yao Hu, Yiqun Liu, and Shaoping Ma. Selfracg: Enabling llms to self-express and retrieve for code generation, 2025a. URL https://arxiv.org/abs/2507.19033.

[38] Qingao Dong, Mengfei Wang, Hengzhi Zhang, Zhichao Li, Yuan Yuan, Mu Li, Xiang Gao, Hailong Sun, Chunming Hu, and Weifeng Lv. Infcode-c++: Intent-guided semantic retrieval and ast-structured search for c++ issue resolution, 2025b. URL https://arxiv.org/abs/2511.16005.

[39] Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. A survey on code generation with llm-based agents, 2025c. URL https://arxiv.org/abs/2508.00083.

[40] Kounianhua Du, Jizheng Chen, Renting Rui, Huacan Chai, Lingyue Fu, Wei Xia, Yasheng Wang, Ruiming Tang, Yong Yu, and Weinan Zhang. Codegrag: Bridging the gap between natural language and programming language via graphical retrieval augmented generation, 2025. URL https://arxiv.org/abs/2405.02355.

[41] Avik Dutta, Mukul Singh, Gust Verbruggen, Sumit Gulwani, and Vu Le. RAR: Retrieval-augmented retrieval for code generation in low resource languages. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 21506–21515, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1199. URL https://aclanthology.org/2024.emnlp-main.1199/.

[42] Aryaz Eghbali and Michael Pradel. De-hallucinator: Mitigating llm hallucinations in code generation tasks via iterative grounding, 2024. URL https://arxiv.org/abs/2401.01701.

[43] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pp. 6491–6501, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901. doi: 10.1145/3637528.3671470. URL https://doi.org/10.1145/3637528.3671470.

[44] Dmitriy Fedrushkov, Denis Tereshchenko, Sergey Kovalchuk, and Artem Aliev. Improving project-level code generation using combined relevant context. In Michael H. Lees, Wentong Cai, Siew Ann Cheong, Yi Su, David Abramson, Jack J. Dongarra, and Peter M. A. Sloot (eds.), *Computational Science – ICCS 2025*, pp. 438–445, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-97635-3.

[45] Fabio Fehr, Prabhu Teja Sivaprasad, Luca Franceschi, and Giovanni Zappella. Coret: Improved retriever for code editing, 2025. URL https://arxiv.org/abs/2505.24715.

[46] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A pre-trained model for programming and natural languages. In Trevor Cohn, Yulan He, and Yang Liu (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1536–1547, Online, November 2020. Association for

Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.139. URL https://aclanthology.org/2020.findings-emnlp.139/.

[47] Ulrich Finkler, Irene Manotas, Wei Zhang, Geert Janssen, Octavian Popescu, and Shyam Ramji. Automated customization of llms for enterprise code repositories using semantic scopes, 2026. URL https://arxiv.org/abs/2602.05780.

[48] Timur Galimzyanov, Olga Kolomyttseva, and Egor Bogomolov. Practical code rag at scale: Task-aware retrieval design choices under compute budgets, 2025. URL https://arxiv.org/abs/2510.20609.

[49] Jatin Ganhotra, Sami Serhan, Antonio Abu Nassar, Avraham Shinnar, Ziv Nevo, and Martin Hirzel. Resolving java code repository issues with iswe agent, 2026. URL https://arxiv.org/abs/2603.11356.

[50] Xinyu Gao, Yun Xiong, Deze Wang, Zhenhan Guan, Zejian Shi, Haofen Wang, and Shanshan Li. Preference-guided refactored tuning for retrieval augmented code generation. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, pp. 65–77, New York, NY, USA, 2024a. Association for Computing Machinery. ISBN 9798400712487. doi: 10.1145/3691620.3694987. URL https://doi.org/10.1145/3691620.3694987.

[51] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2024b. URL https://arxiv.org/abs/2312.10997.

[52] God of Prompt. Local llm setup for privacy-conscious businesses. *God of Prompt (blog)*, 2025. https://www.godofprompt.ai/blog/local-llm-setup-for-privacy-conscious-businesses.

[53] Google DeepMind. Gemini. https://deepmind.google/models/gemini/, August 2025. Google DeepMind official website.

[54] Wenchao Gu, Juntao Chen, Yanlin Wang, Tianyue Jiang, Xingzhe Li, Mingwei Liu, Xilin Liu, Yuchi Ma, and Zibin Zheng. What to retrieve for effective retrieval-augmented code generation? an empirical study and beyond, 2025. URL https://arxiv.org/abs/2503.20589.

[55] Zhanming Guan, Junlin Liu, Jierui Liu, Chao Peng, Dexin Liu, Ningyuan Sun, Bo Jiang, Wenchao Li, Jie Liu, and Hang Zhu. Contextmodule: Improving code completion via repository-level contextual information, 2024. URL https://arxiv.org/abs/2412.08063.

[56] Daya Guo, Shuo Ren, Shuai Lu, et al. Graphcodebert: Pre-training code representations with data flow. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=jLoC4ez43PZ.

[57] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. UniXcoder: Unified cross-modal pre-training for code representation. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7212–7225, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.499. URL https://aclanthology.org/2022.acl-long.499/.

[58] Daya Guo, Qihao Zhu, Dejian Yang, et al. Deepseek-coder: When the large language model meets programming – the rise of code intelligence, 2024a. URL https://arxiv.org/abs/2401.14196.

[59] Qi Guo, Xiaohong Li, Xiaofei Xie, Shangqing Liu, Ze Tang, Ruitao Feng, Junjie Wang, Jidong Ge, and Lei Bu. Ft2ra: A fine-tuning-inspired approach to retrieval-augmented code completion. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, pp. 313–324, New York, NY, USA, 2024b. Association for Computing Machinery. ISBN 9798400706127. doi: 10.1145/3650212.3652130. URL https://doi.org/10.1145/3650212.3652130.

[60] Dhruv Gupta, Gayathri Ganesh Lakshmy, and Yiqing Xie. Sacl: Understanding and combating textual bias in code retrieval with semantic-augmented reranking and localization, 2025. URL https://arxiv.org/abs/2506.20081.

[61] Junda He, Christoph Treude, and David Lo. Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Trans. Softw. Eng. Methodol.*, 34(5), May 2025. ISSN 1049-331X. doi: 10.1145/3712003. URL https://doi.org/10.1145/3712003.

[62] Kang He and Kaushik Roy. Swe-adept: An llm-based agentic framework for deep codebase analysis and structured issue resolution, 2026. URL https://arxiv.org/abs/2603.01327.

[63] Marko Hostnik and Marko Robnik-Šikonja. Retrieval-augmented code completion for local projects using large language models. *Expert Systems with Applications*, 292:128596, 2025. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2025.128596. URL https://www.sciencedirect.com/science/article/pii/S0957417425022158.

[64] Chao Hu, Wenhao Zeng, Yuling Shi, Beijun Shen, and Xiaodong Gu. In line with context: Repository-level code generation via context inlining, 2026. URL https://arxiv.org/abs/2601.00376.

[65] Binyuan Hui, Jian Yang, Zeyu Cui, et al. Qwen2.5-coder technical report, 2024. URL https://arxiv.org/abs/2409.12186.

[66] Yu Huo, Kun Zeng, Siyu Zhang, Yuquan Lu, Cheng Yang, Yifu Guo, and Xiaoying Tang. Reposhapley: Shapley-enhanced context filtering for repository-level code completion, 2026. URL https://arxiv.org/abs/2601.03378.

[67] Zahin Ibnat, Paul E. Calzada, Rasin Mohammed Ihtemam, Sujan Kumar Saha, Jingbo Zhou, Farimah Farahmandi, and Mark Tehranipoor. Deepv: A model-agnostic retrieval-augmented framework for verilog code generation with a high-quality knowledge base, 2025. URL https://arxiv.org/abs/2510.05327.

[68] Paul Jaccard. Comparative study of the floral distribution in a portion of the alps and the jura. *Bulletin of the Vaud Society of Natural Sciences*, 37:547–579, 1901.

[69] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

[70] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. *ACM Trans. Softw. Eng. Methodol.*, July 2025a. ISSN 1049-331X. doi: 10.1145/3747588. URL https://doi.org/10.1145/3747588. Just Accepted.

[71] Mingjian Jiang, Yangjun Ruan, Luis Lastras, Pavan Kapanipathi, and Tatsunori Hashimoto. Putting it all into context: Simplifying agents with lclms, 2025b. URL https://arxiv.org/abs/2505.08120.

[72] Tianyue Jiang, Yanli Wang, Yanlin Wang, Daya Guo, Ensheng Shi, Yuchi Ma, Jiachi Chen, and Zibin Zheng. Aligncoder: Aligning retrieval with target intent for repository-level code completion, 2026. URL https://arxiv.org/abs/2601.19697.

[73] Zhijie Jiang, Zejian Shi, Xinyu Gao, and Yun Xiong. Enhancing code generation through retrieval of cross-lingual semantic graphs. In *2024 31st Asia-Pacific Software Engineering Conference (APSEC)*, pp. 151–160, 2024. doi: 10.1109/APSEC65559.2024.00026.

[74] Zhonghao Jiang, Xiaoxue Ren, Meng Yan, Wei Jiang, Yong Li, and Zhongxin Liu. Cosil: Software issue localization via llm-driven code repository graph searching, 2025c. URL https://arxiv.org/abs/2503.22424.

[75] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

[76] Hangzhan Jin and Mohammad Hamdaqa. Ccci: Code completion with contextual information for complex data transfer tasks using large language models, 2025. URL https://arxiv.org/abs/2503.23231.

[77] Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. Inferfix: End-to-end program repair with llms. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, pp. 1646–1656, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400703270. doi: 10.1145/3611643.3613892. URL https://doi.org/10.1145/3611643.3613892.

[78] Mizuki Kondo, Daisuke Kawahara, and Toshiyuki Kurabayashi. Improving repository-level code search with text conversion. In Yang (Trista) Cao, Isabel Papadimitriou, Anaelia Ovalle, Marcos Zampieri, Francis Ferraro, and Swabha Swayamdipta (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 4: Student Research Workshop)*, pp. 130–137, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-srw.15. URL https://aclanthology.org/2024.naacl-srw.15/.

[79] Chalamalasetti Kranti, Sherzod Hakimov, and David Schlangen. Retrieval-augmented code generation for situated action generation: A case study on Minecraft. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 11159–11170, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.652. URL https://aclanthology.org/2024.findings-emnlp.652/.

[80] Minh Le-Anh, Huyen Nguyen, Khanh An Tran, Nam Le Hai, Linh Ngo Van, Nghi D. Q. Bui, and Bach Le. Do not treat code as natural language: Implications for repository-level code generation and beyond, 2026. URL https://arxiv.org/abs/2602.11671.

[81] Chuanyi Li, Jiwei Shang, Yi Feng, and Bin Luo. Hyracc: A hybrid retrieval-augmented framework for more efficient code completion. In *2025 IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering (Forge)*, pp. 61–66, 2025a. doi: 10.1109/Forge66646.2025.00013.

[82] Han Li, Yuling Shi, Shaoxin Lin, Xiaodong Gu, Heng Lian, Xin Wang, Yantao Jia, Tao Huang, and Qianxiang Wang. Swe-debate: Competitive multi-agent debate for software issue resolution, 2025b. URL https://arxiv.org/abs/2507.23348.

[83] Haochen Li, Xin Zhou, and Zhiqi Shen. Rewriting the code: A simple method for large language model augmented code search. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1371–1389, Bangkok, Thailand, August 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.75. URL https://aclanthology.org/2024.acl-long.75/.

[84] Jia Li, Ge Li, Yunfei Zhao, Yongmin Li, Huanyu Liu, Hao Zhu, Lecheng Wang, Kaibo Liu, Zheng Fang, Lanshen Wang, Jiazheng Ding, Xuanming Zhang, Yuqi Zhu, Yihong Dong, Zhi Jin, Binhua Li, Fei Huang, Yongbin Li, Bin Gu, and Mengfei Yang. DevEval: A manually-annotated code generation benchmark aligned with real-world code repositories. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3603–3614, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.214. URL https://aclanthology.org/2024.findings-acl.214/.

[85] Jia Li, Ge Li, Xuanming Zhang, Yunfei Zhao, Yihong Dong, Zhi Jin, Binhua Li, Fei Huang, and Yongbin Li. Evocodebench: an evolving code generation benchmark with domain-specific evaluations. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2025c. Curran Associates Inc. ISBN 9798331314385.

[86] Jia Li, Xianjie Shi, Kechi Zhang, Ge Li, Zhi Jin, Lei Li, Huangzhao Zhang, Jia Li, Fang Liu, Yuwei Zhang, Zhengwei Tao, Yihong Dong, Yuqi Zhu, and Chongyang Tao. Graphcodeagent: Dual graph-guided llm agent for retrieval-augmented repo-level code generation, 2025d. URL https://arxiv.org/abs/2504.10046.

[87] Jia Li, Hao Zhu, Huanyu Liu, Xianjie Shi, He Zong, Yihong Dong, Kechi Zhang, Siyuan Jiang, Zhi Jin, and Ge Li. Aligning llms to fully utilize the cross-file context in repository-level code completion. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1477–1489. IEEE Press, 2025e. doi: 10.1109/ASE63991.2025.00125. URL https://doi.org/10.1109/ASE63991.2025.00125.

[88] Jia Li, Hao Zhu, Huanyu Liu, Xianjie Shi, He Zong, Yihong Dong, Kechi Zhang, Siyuan Jiang, Zhi Jin, and Ge Li. aixcoder-7b-v2: Training llms to fully utilize the long context in repository-level code completion, 2025f. URL https://arxiv.org/abs/2503.15301.

[89] Kenan Li, Rongzhi Li, Linghao Zhang, Qirui Jin, Liao Zhu, Xiaosong Huang, Geng Zhang, Yikai Zhang, Shilin He, Chengxing Xie, Xin Zhang, Zijian Jin, Bowen Li, Chaoyun Zhang, Yu Kang, Yufan Huang, Elsie Nallipogu, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. Repolaunch: Automating build&test pipeline of code repositories on any language and any platform, 2026a. URL https://arxiv.org/abs/2603.05026.

[90] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pp. 661–670, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605587998. doi: 10.1145/1772690.1772758. URL https://doi.org/10.1145/1772690.1772758.

[91] Raymond Li, Loubna Ben Allal, Yangtian Zi, et al. Starcoder: may the source be with you! *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=KoFOg41haE. Reproducibility Certification.

[92] Tian Li, Bo Lin, Shangwen Wang, and Yusong Tan. Exploring the security threats of retriever backdoors in retrieval-augmented code generation, 2025g. URL https://arxiv.org/abs/2512.21681.

[93] Xinze Li, Hanbin Wang, Zhenghao Liu, Shi Yu, Shuo Wang, Yukun Yan, Yukai Fu, Yu Gu, and Ge Yu. Building a coding assistant via the retrieval-augmented language model. *ACM Trans. Inf. Syst.*, 43(2), January 2025h. ISSN 1046-8188. doi: 10.1145/3695868. URL https://doi.org/10.1145/3695868.

[94] Yanzhou Li, Shangqing Liu, Kangjie Chen, Tianwei Zhang, and Yang Liu. Impact-driven context filtering for cross-file code completion. In *Second Conference on Language Modeling*, 2025i. URL https://openreview.net/forum?id=0Y2zXLFBji.

[95] Yanzhou Li, Tianlin Li, Yiran Zhang, Shangqing Liu, Aishan Liu, Xianglong Liu, and Yang Liu. From context to intent: Reasoning-guided function-level code completion, 2026b. URL https://arxiv.org/abs/2508.09537.

[96] Yichen Li, Yun Peng, Yintong Huo, and Michael R. Lyu. Enhancing llm-based coding tools through native integration of ide-derived static context. In *Proceedings of the 1st International Workshop on Large Language Models for Code*, LLM4Code '24, pp. 70–74, New York, NY, USA, 2024c. Association for Computing Machinery. ISBN 9798400705793. doi: 10.1145/3643795.3648392. URL https://doi.org/10.1145/3643795.3648392.

[97] Yifan Li, Ensheng Shi, Dewu Zheng, Kefeng Duan, Jiachi Chen, and Yanlin Wang. Repomincoder: Improving repository-level code generation based on information loss screening. In *Proceedings of the 15th Asia-Pacific Symposium on Internetware*, Internetware '24, pp. 229–238, New York, NY, USA, 2024d. Association for Computing Machinery. ISBN 9798400707056. doi: 10.1145/3671016.3674819. URL https://doi.org/10.1145/3671016.3674819.

[98] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. Vuldeepecker: A deep learning-based system for vulnerability detection. In *Proceedings 2018 Network and Distributed System Security Symposium*, NDSS 2018. Internet Society, 2018. doi: 10.14722/ndss.2018.23158. URL http://dx.doi.org/10.14722/ndss.2018.23158.

[99] Keyu Liang, Zhongxin Liu, Chao Liu, Zhiyuan Wan, David Lo, and Xiaohu Yang. Zero-shot cross-domain code search without fine-tuning. *Proc. ACM Softw. Eng.*, 2(FSE), June 2025. doi: 10.1145/3729357. URL https://doi.org/10.1145/3729357.

[100] Ming Liang, Xiaoheng Xie, Gehao Zhang, Xunjin Zheng, Peng Di, Wei Jiang, Hongwei Chen, Chengpeng Wang, and Gang Fan. Repogenix: Dual context-aided repository-level code completion with language models. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, pp. 2466–2467, New York, NY, USA, 2024a. Association for Computing Machinery. ISBN 9798400712487. doi: 10.1145/3691620.3695331. URL https://doi.org/10.1145/3691620.3695331.

[101] Ming Liang, Xiaoheng Xie, Gehao Zhang, Xunjin Zheng, Peng Di, wei jiang, Hongwei Chen, Chengpeng Wang, and Gang Fan. Repofuse: Repository-level code completion with fused dual context, 2024b. URL https://arxiv.org/abs/2402.14323.

[102] Dianshu Liao, Shidong Pan, Xiaoyu Sun, Xiaoxue Ren, Qing Huang, Zhenchang Xing, Huan Jin, and Qinying Li. $\mathbf{A^3}$A3-CodGen: A Repository-Level Code Generation Framework for Code Reuse With Local-Aware, Global-Aware, and Third-Party-Library-Aware . *IEEE Transactions on Software Engineering*, 50(12):3369–3384, December 2024. ISSN 1939-3520. doi: 10.1109/TSE.2024.3486195. URL https://doi.ieeecomputersociety.org/10.1109/TSE.2024.3486195.

[103] Bo Lin, Shangwen Wang, Yihao Qin, Liqian Chen, and Xiaoguang Mao. Give llms a security course: Securing retrieval-augmented code generation via knowledge injection, 2025. URL https://arxiv.org/abs/2504.16429.

[104] Junwei Liu, Yixuan Chen, Mingwei Liu, Xin Peng, and Yiling Lou. Stall+: Boosting llm-based repository-level code completion with static analysis, 2024a. URL https://arxiv.org/abs/2406.10018.

[105] Mingwei Liu, Tianyong Yang, Yiling Lou, Xueying Du, Ying Wang, and Xin Peng. Codegen4libs: A two-stage approach for library-oriented code generation. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 434–445, 2023. doi: 10.1109/ASE56229.2023.00159.

[106] Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=pPjZIOuQuF.

[107] Wei Liu, Ailun Yu, Daoguang Zan, Bo Shen, Wei Zhang, Haiyan Zhao, Zhi Jin, and Qianxiang Wang. Graphcoder: Enhancing repository-level code completion via coarse-to-fine retrieval based on code context graph. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, pp. 570–581, New York, NY, USA, 2024c. Association for Computing Machinery. ISBN 9798400712487. doi: 10.1145/3691620.3695054. URL https://doi.org/10.1145/3691620.3695054.

[108] Wei Liu, Chao Peng, Pengfei Gao, Aofan Liu, Wei Zhang, Haiyan Zhao, and Zhi Jin. Graphlocator: Graph-guided causal reasoning for issue localization, 2025a. URL https://arxiv.org/abs/2512.22469.

[109] Xiangyan Liu, Bo Lan, Zhiyuan Hu, Yang Liu, Zhicheng Zhang, Fei Wang, Michael Qizhe Shieh, and Wenmeng Zhou. CodexGraph: Bridging large language models and code repositories via code graph databases. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 142–160, Albuquerque, New Mexico, April 2025b. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.7. URL https://aclanthology.org/2025.naacl-long.7/.

[110] Yang Liu, Li Zhang, Fang Liu, Zhuohang Wang, Donglin Wei, Zhishuo Yang, Kechi Zhang, Jia Li, and Lin Shi. Enhancing repository-level code generation with call chain-aware multi-view context, 2025c. URL https://arxiv.org/abs/2507.14791.

[111] Ye Liu, Rui Meng, Shafiq Joty, silvio savarese, Caiming Xiong, Yingbo Zhou, and Semih Yavuz. CodeXEmbed: A generalist embedding model family for multilingual and multi-task code retrieval. In *Second Conference on Language Modeling*, 2025d. URL https://openreview.net/forum?id=z3lG70Azbg.

[112] Anton Lozhkov, Raymond Li, Loubna Ben Allal, et al. Starcoder 2 and the stack v2: The next generation, 2024. URL https://arxiv.org/abs/2402.19173.

[113] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, MING GONG, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie LIU. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL https://openreview.net/forum?id=6lE4dQXaUcb.

[114] Shuai Lu, Nan Duan, Hojae Han, Daya Guo, Seung-won Hwang, and Alexey Svyatkovskiy. ReACC: A retrieval-augmented code completion framework. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6227–6240, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.431. URL https://aclanthology.org/2022.acl-long.431/.

[115] Jane Luo, Xin Zhang, Steven Liu, Jie Wu, Jianfeng Liu, Yiming Huang, Yangyu Huang, Chengyu Yin, Ying Xin, Yuefeng Zhan, Hao Sun, Qi Chen, Scarlett Li, and Mao Yang. RPG: A repository planning graph for unified and scalable codebase generation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=VAQq3Y8tIF.

[116] George Ma, Anurag Koul, Qi Chen, Yawen Wu, Sachit Kuhar, Yu Yu, Aritra Sengupta, Varun Kumar, and Murali Krishna Ramanathan. Specagent: A speculative retrieval and forecasting agent for code completion, 2026. URL https://arxiv.org/abs/2510.17925.

[117] Yingwei Ma, Qingping Yang, Rongyu Cao, Binhua Li, Fei Huang, and Yongbin Li. Alibaba lingmaagent: Improving automated issue resolution via comprehensive repository exploration. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, FSE Companion '25, pp. 238–249, New York, NY, USA, 2025a. Association for Computing Machinery. ISBN 9798400712760. doi: 10.1145/3696630.3728549. URL https://doi.org/10.1145/3696630.3728549.

[118] Zexiong Ma, Chao Peng, Pengfei Gao, Xiangxin Meng, Yanzhen Zou, and Bing Xie. SoRFT: Issue resolving with subtask-oriented reinforced fine-tuning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11427–11441, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.559. URL https://aclanthology.org/2025.acl-long.559/.

[119] Zexiong Ma, Chao Peng, Qunhong Zeng, Pengfei Gao, Yanzhen Zou, and Bing Xie. Tool-integrated reinforcement learning for repo deep search, 2025c. URL https://arxiv.org/abs/2508.03012.

[120] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[121] Noor Nashid, Mifta Sintaha, and Ali Mesbah. Retrieval-based prompt selection for code-related few-shot learning. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 2450–2462, 2023. doi: 10.1109/ICSE48619.2023.00205.

[122] Erik Nijkamp, Hiroaki Hayashi, Caiming Xiong, Silvio Savarese, and Yingbo Zhou. Codegen2: Lessons for training llms on programming and natural languages. *ICLR*, 2023a.

[123] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *ICLR*, 2023b.

[124] OpenAI. Introducing gpt-5.5. `https://openai.com/index/introducing-gpt-5-5/`, April 2025a. OpenAI official website.

[125] OpenAI. text-embedding-ada-002. `https://platform.openai.com/docs/models/text-embedding-ada-002`, August 2025b. OpenAI API documentation.

[126] OpenAI. Why swe-bench verified no longer measures frontier coding capabilities. `https://openai.com/index/why-we-no-longer-evaluate-swe-bench-verified/`, 2026.

[127] Otterly.AI Blog. Knowledge cutoff dates of all llms explained. `https://otterly.ai/blog/knowledge-cutoff/`, February 2024. Accessed on September 4, 2025.

[128] Shuyin Ouyang, Jie M. Zhang, Zeyu Sun, and Albert Merono Penuela. Knowledge-enhanced program repair for data science code. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 898–910, 2025a. doi: 10.1109/ICSE55347.2025.00246.

[129] Siru Ouyang, Wenhao Yu, Kaixin Ma, Zilin Xiao, Zhihan Zhang, Mengzhao Jia, Jiawei Han, Hongming Zhang, and Dong Yu. Repograph: Enhancing AI software engineering with repository-level code graph. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL `https://openreview.net/forum?id=dw9VUsSHGB`.

[130] Zhiyuan Pan, Xing Hu, Xin Xia, and Xiaohu Yang. Enhancing repository-level code generation with integrated contextual information, 2024. URL `https://arxiv.org/abs/2406.03283`.

[131] YIBO PENG, Zora Zhiruo Wang, and Daniel Fried. Can long-context language models solve repository-level code generation? In *LTI Student Research Symposium 2025*, 2025. URL `https://openreview.net/forum?id=pmcWo9DtDw`.

[132] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE'08, pp. 68–77, Swindon, GBR, 2008. BCS Learning & Development Ltd.

[133] Huy N. Phan, Hoang N. Phan, Tien N. Nguyen, and Nghi D. Q. Bui. Repohyper: Search-expand-refine on semantic graphs for repository-level code completion. In *2025 IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering (Forge)*, pp. 14–25, 2025. doi: 10.1109/Forge66646.2025.00009.

[134] Revanth Gangi Reddy, Tarun Suresh, JaeHyeok Doo, Ye Liu, Xuan Phi Nguyen, Yingbo Zhou, Semih Yavuz, Caiming Xiong, Heng Ji, and Shafiq Joty. Swerank: Software issue localization with code ranking, 2025. URL `https://arxiv.org/abs/2505.07849`.

[135] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In Bruce W. Croft and C. J. van Rijsbergen (eds.), *SIGIR '94*, pp. 232–241, London, 1994. Springer London. ISBN 978-1-4471-2099-5.

[136] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline M. Hancock-Beaulieu, and Mike Gatford. Okapi at trec-3. *Proceedings of the Third Text REtrieval Conference (TREC-3)*, pp. 109–126, 1995.

[137] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, et al. Code llama: Open foundation models for code, 2024. URL `https://arxiv.org/abs/2308.12950`.

[138] Iman Saberi and Fatemeh Fard. Context-augmented code generation using programming knowledge graphs, 2025. URL `https://arxiv.org/abs/2410.18251`.

[139] Hitesh Sagtani, Rishabh Mehrotra, and Beyang Liu. Improving fim code completions via context & curriculum based learning. In *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining*, WSDM '25, pp. 801–810, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400713293. doi: 10.1145/3701551.3703563. URL https://doi.org/10.1145/3701551.3703563.

[140] Pratik Shah, Rajat Ghosh, Aryan Singhal, and Debojyoti Dutta. Ranger – repository-level agent for graph-enhanced retrieval, 2025. URL https://arxiv.org/abs/2509.25257.

[141] Jiahao Shi and Tianyi Zhang. RESCUE: Retrieval augmented secure code generation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=gbxhesw4UH.

[142] Disha Shrivastava, Denis Kocetkov, Harm de Vries, Dzmitry Bahdanau, and Torsten Scholak. Repofusion: Training code models to understand your repository, 2023a. URL https://arxiv.org/abs/2306.10998.

[143] Disha Shrivastava, Hugo Larochelle, and Daniel Tarlow. Repository-level prompt generation for large language models of code. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023b.

[144] Aryan Singhal, Rajat Ghosh, Ria Mundra, Harshil Dadlani, and Debojyoti Dutta. Code2JSON: Can a zero-shot LLM agent extract code features for code RAG? In *ICLR 2025 Third Workshop on Deep Learning for Code*, 2025. URL https://openreview.net/forum?id=glBWrVLvKi.

[145] Atefeh Sohrabizadeh, Jialin Song, Mingjie Liu, Rajarshi Roy, Chankyu Lee, Jonathan Raiman, and Bryan Catanzaro. Nemotron-CORTEXA: Enhancing LLM agents for software engineering tasks via improved localization and solution diversity. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=k6p8UKRdH7.

[146] Nikita Sorokin, Ivan Sedykh, and Valentin Malykh. Hierarchical embedding fusion for retrieval-augmented code generation, 2026. URL https://arxiv.org/abs/2603.06593.

[147] David Sounthiraraj, Jared Hancock, Yassin Kortam, Ashok Javvaji, Prabhat Singh, and Shaila Shankar. Code-craft: Hierarchical graph-based code summarization for enhanced context retrieval, 2025. URL https://arxiv.org/abs/2504.08975.

[148] Hongjin Su, Shuyang Jiang, Yuhang Lai, Haoyuan Wu, Boao Shi, Che Liu, Qian Liu, and Tao Yu. EvoR: Evolving retrieval for code generation. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 2538–2554, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.143. URL https://aclanthology.org/2024.findings-emnlp.143/.

[149] Lintang Sutawika, Aditya Bharat Soni, Bharath Sriraam R R, Apurva Gandhi, Taha Yassine, Sanidhya Vijayvargiya, Yuchen Li, Xuhui Zhou, Yilin Zhang, Leander Melroy Maben, and Graham Neubig. Codescout: An effective recipe for reinforcement learning of code search agents, 2026. URL https://arxiv.org/abs/2603.17829.

[150] Hanzhuo Tan, Qi Luo, Ling Jiang, Zizheng Zhan, Jing Li, Haotian Zhang, and Yuqun Zhang. Prompt-based code completion via multi-retrieval augmented generation. *ACM Trans. Softw. Eng. Methodol.*, March 2025. ISSN 1049-331X. doi: 10.1145/3725812. URL https://doi.org/10.1145/3725812. Just Accepted.

[151] Xunzhu Tang, Jiechao Gao, Jin Xu, Tiezhu Sun, Yewei Song, Saad Ezzini, Wendkûuni C. Ouédraogo, Jacques Klein, and Tegawendé F. Bissyandé. SynFix: Dependency-aware program repair via RelationGraph analysis. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 4878–4894, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.252. URL https://aclanthology.org/2025.findings-acl.252/.

[152] Ze Tang, Jidong Ge, Shangqing Liu, Tingwei Zhu, Tongtong Xu, Liguo Huang, and Bin Luo. Domain adaptive code completion via language models and decoupled domain databases. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 421–433, 2023. doi: 10.1109/ASE56229.2023.00076.

[153] Hongyuan Tao, Ying Zhang, Zhenhao Tang, Hongen Peng, Xukun Zhu, Bingchang Liu, Yingguang Yang, Ziyin Zhang, Zhaogui Xu, Haipeng Zhang, Linchao Zhu, Rui Wang, Hang Yu, Jianguo Li, and Peng Di. Code graph model (cgm): A graph-integrated large language model for repository-level software engineering tasks, 2025. URL https://arxiv.org/abs/2505.16901.

[154] Aider Team. Aider polyglot leaderboard. https://aider.chat/docs/leaderboards/, 2024. Accessed: 2025-08-21.

[155] Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[156] Catherine Tony, Emanuele Iannone, and Riccardo Scandariato. Retrieve, refine, or both? using task-specific guidelines for secure python code generation, 2025. manuscript available at https://emaiannone.github.io/assets/pdf/c6.pdf.

[157] Yunda Tsai, Mingjie Liu, and Haoxing Ren. Rtlfixer: Automatically fixing rtl syntax errors with large language model. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, DAC '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706011. doi: 10.1145/3649329.3657353. URL https://doi.org/10.1145/3649329.3657353.

[158] Voyage AI. voyage-code-3: more accurate code retrieval with lower dimensional, quantized embeddings. https://blog.voyageai.com/2024/12/04/voyage-code-3/, December 2024. Voyage AI blog post.

[159] Baoyi Wang, Xingliang Wang, Guochang Li, Chen Zhi, Junxiao Han, Xinkui Zhao, Nan Wang, Shuiguang Deng, and Jianwei Yin. Greprag: An empirical study and optimization of grep-like retrieval for code completion, 2026a. URL https://arxiv.org/abs/2601.23254.

[160] Chaozheng Wang, Zezhou Yang, Shuzheng Gao, Cuiyun Gao, Ting Peng, Hailiang Huang, Yuetang Deng, and Michael Lyu. Rag or fine-tuning? a comparative study on lcms-based code completion in industry. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, FSE Companion '25, pp. 93–104, New York, NY, USA, 2025a. Association for Computing Machinery. ISBN 9798400712760. doi: 10.1145/3696630.3728535. URL https://doi.org/10.1145/3696630.3728535.

[161] Chong Wang, Jian Zhang, Yebo Feng, Tianlin Li, Weisong Sun, Yang Liu, and Xin Peng. Teaching code llms to use autocompletion tools in repository-level code generation. *ACM Trans. Softw. Eng. Methodol.*, 34(7), August 2025b. ISSN 1049-331X. doi: 10.1145/3714462. URL https://doi.org/10.1145/3714462.

[162] Huacan Wang, Ziyi Ni, Shuo Zhang, Shuo Lu, Sen Hu, Ziyang He, Chen Hu, Jiaye Lin, Yifu Guo, Yuntao Du, and Pin Lyu. Repomaster: Autonomous exploration and understanding of github repositories for complex task solving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026b. URL https://openreview.net/forum?id=aSfBbhUJAa.

[163] Jicheng Wang, Yifeng He, and Hao Chen. Repogenreflex: Enhancing repository-level code completion with verbal reinforcement and retrieval-augmented generation, 2024. URL https://arxiv.org/abs/2409.13122.

[164] Mengzhen Wang, Shukai Ma, Songwen Gong, Jiexin Wang, Ruolin Chen, Liuwen Cao, and Yi Cai. Sracg: A code generation framework with selective retrieval augmentation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(39):33584–33592, Mar. 2026c. doi: 10.1609/aaai.v40i39.40647. URL https://ojs.aaai.org/index.php/AAAI/article/view/40647.

[165] Peiding Wang, Li Zhang, Fang Liu, Chongyang Tao, and Yinghao Zhu. Codemem: Ast-guided adaptive memory for repository-level iterative code generation, 2026d. URL https://arxiv.org/abs/2601.02868.

[166] Weishi Wang, Yue Wang, Shafiq Joty, and Steven C.H. Hoi. Rap-gen: Retrieval-augmented patch generation with codet5 for automatic program repair. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, pp. 146–158, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400703270. doi: 10.1145/3611643.3616256. URL https://doi.org/10.1145/3611643.3616256.

[167] Xingliang Wang, Baoyi Wang, Chen Zhi, Junxiao Han, Xinkui Zhao, Jianwei Yin, and Shuiguang Deng. Grace: Graph-guided repository-aware code completion through hierarchical code fusion, 2025c. URL https://arxiv.org/abs/2509.05980.

[168] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025d. URL https://openreview.net/forum?id=OJd3ayDDoF.

[169] Yanlin Wang, Yanli Wang, Daya Guo, Jiachi Chen, Ruikai Zhang, Yuchi Ma, and Zibin Zheng. RLCoder: Reinforcement Learning for Repository-Level Code Completion . In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 1140–1152, Los Alamitos, CA, USA, May 2025e. IEEE Computer Society. doi: 10.1109/ICSE55347.2025.00014. URL https://doi.ieeecomputersociety.org/10.1109/ICSE55347.2025.00014.

[170] Yanlin Wang, Jiadong Wu, Tianyue Jiang, Mingwei Liu, Jiachi Chen, Chong Wang, Ensheng Shi, Xilin Liu, Yuchi Ma, and Zibin Zheng. Draincode: Stealthy energy consumption attacks on retrieval-augmented code generation via context poisoning, 2026e. URL https://arxiv.org/abs/2601.20615.

[171] Ying Wang, Wenjun Mao, Chong Wang, Zhenhao Zhou, Yicheng Zhou, Wenyun Zhao, Yiling Lou, and Xin Peng. Extracting conceptual knowledge to locate software issues, 2025f. URL https://arxiv.org/abs/2509.21427.

[172] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 8696–8708, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.685. URL https://aclanthology.org/2021.emnlp-main.685/.

[173] Zora Zhiruo Wang, Akari Asai, Xinyan Velocity Yu, Frank F. Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. CodeRAG-bench: Can retrieval augment code generation? In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 3199–3214, Albuquerque, New Mexico, April 2025g. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.176. URL https://aclanthology.org/2025.findings-naacl.176/.

[174] Di Wu, Wasi Uddin Ahmad, Dejiao Zhang, Murali Krishna Ramanathan, and Xiaofei Ma. Repoformer: selective retrieval for repository-level code completion. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[175] Peiyang Wu, Nan Guo, Junliang Lv, Xiao Xiao, and Xiaochun Ye. Rtlrepocoder: Repository-level rtl code completion through the combination of fine-tuning and retrieval augmentation, 2025. URL https://arxiv.org/abs/2504.08862.

[176] Yusen Wu and Xiaotie Deng. Hcag: Hierarchical abstraction and retrieval-augmented generation on theoretical repositories with llms, 2026. URL https://arxiv.org/abs/2603.20299.

[177] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying llm-based software engineering agents. *Proc. ACM Softw. Eng.*, 2(FSE), June 2025. doi: 10.1145/3715754. URL https://doi.org/10.1145/3715754.

[178] Chengxing Xie, Bowen Li, Chang Gao, He Du, Wai Lam, Difan Zou, and Kai Chen. SWE-fixer: Training open-source LLMs for effective and efficient GitHub issue resolution. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 1123–1139, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. URL https://aclanthology.org/2025.findings-acl.62/.

[179] Boyang Yang, Haoye Tian, Jiadong Ren, Shunfu Jin, Yang Liu, Feng Liu, and Bach Le. Enhancing repository-level software repair via repository-aware knowledge graphs, 2025a. URL https://arxiv.org/abs/2503.21710.

[180] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: agent-computer interfaces enable automated software engineering. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

[181] John Yang, Carlos E Jimenez, Alex L Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R Narasimhan, Diyi Yang, Sida Wang, and Ofir Press. SWE-bench multimodal: Do AI systems generalize to visual software domains? In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=riTiq3i21b.

[182] Zezhou Yang, Sirong Chen, Cuiyun Gao, Zhenhao Li, Xing Hu, Kui Liu, and Xin Xia. An empirical study of retrieval-augmented code generation: Challenges and opportunities. *ACM Trans. Softw. Eng. Methodol.*, 34(7), August 2025c. ISSN 1049-331X. doi: 10.1145/3717061. URL https://doi.org/10.1145/3717061.

[183] Zezhou Yang, Ting Peng, Cuiyun Gao, Chaozheng Wang, Hailiang Huang, and Yuetang Deng. A deep dive into retrieval-augmented generation for code completion: Experience on wechat, 2025d. URL https://arxiv.org/abs/2507.18515.

[184] Kai Ye, Liangcai Su, and Chenxiong Qian. Importsnare: Directed 'code manual' hijacking in retrieval-augmented code generation. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, CCS '25, pp. 335–349, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400715259. doi: 10.1145/3719027.3765161. URL https://doi.org/10.1145/3719027.3765161.

[185] Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. Learning to mine aligned code and natural language pairs from stack overflow. In *International Conference on Mining Software Repositories*, MSR, pp. 476–486. ACM, 2018. doi: https://doi.org/10.1145/3196398.3196408.

[186] Jaeseok Yoo, Hojae Han, Youngwon Lee, Jaejin Kim, and Seung-won Hwang. PERC: Plan-as-query example retrieval for underrepresented code generation. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (eds.), *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 7982–7997, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.534/.

[187] Hao Yu, Bo Shen, Dezhi Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianxiang Wang, and Tao Xie. Codereval: A benchmark of pragmatic code generation with generative pre-trained models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024a. Association for Computing Machinery. ISBN 9798400702174. doi: 10.1145/3597503.3623316. URL https://doi.org/10.1145/3597503.3623316.

[188] Xinran Yu, Chun Li, Minxue Pan, and Xuandong Li. Droidcoder: Enhanced android code completion with context-enriched retrieval-augmented generation. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, pp. 681–693, New York, NY, USA, 2024b. Association for Computing Machinery. ISBN 9798400712487. doi: 10.1145/3691620.3695063. URL https://doi.org/10.1145/3691620.3695063.

[189] Zhongming Yu, Hejia Zhang, Yujie Zhao, Hanxian Huang, Matrix Yao, Ke Ding, and Jishen Zhao. Orcaloca: An LLM agent framework for software issue localization. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=LyUfPOvM6I.

[190] Daoguang Zan, Bei Chen, Yongshun Gong, Junzhi Cao, Fengji Zhang, Bingchao Wu, Bei Guan, Yilong Yin, and Yongji Wang. Private-library-oriented code generation with large language models. *Knowledge-Based Systems*, 326:113934, 2025. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2025.113934. URL https://www.sciencedirect.com/science/article/pii/S0950705125009797.

[191] Dun Zhang, Jiacheng Li, Ziyang Zeng, and Fulong Wang. Jasper and stella: distillation of sota embedding models, 2025a. URL https://arxiv.org/abs/2412.19048.

[192] Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. RepoCoder: Repository-level code completion through iterative retrieval and generation. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2471–2484, Singapore, December 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.151. URL https://aclanthology.org/2023.emnlp-main.151/.

[193] Jiajun Zhang, Zeyu Cui, Jiaxi Yang, Lei Zhang, Yuheng Jing, Zeyao Ma, Tianyi Bai, Zilei Wang, Qiang Liu, Liang Wang, Binyuan Hui, and Junyang Lin. From completion to editing: Unlocking context-aware code infilling via search-and-replace instruction tuning, 2026a. URL https://arxiv.org/abs/2601.13384.

[194] Lei Zhang, Yunshui Li, Jiaming Li, Xiaobo Xia, Jiaxi Yang, Run Luo, Minzheng Wang, Longze Chen, Junhao Liu, Qiang Qu, and Min Yang. Hierarchical context pruning: Optimizing real-world code completion with repository-level pretrained code llms. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(24):25886–25894, Apr. 2025b. doi: 10.1609/aaai.v39i24.34782. URL https://ojs.aaai.org/index.php/AAAI/article/view/34782.

[195] Sheng Zhang, Yifan Ding, Shuquan Lian, Shun Song, and Hui Li. CodeRAG: Finding relevant and necessary knowledge for retrieval-augmented repository-level code completion. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 23278–23288, Suzhou, China, November 2025c. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1187. URL https://aclanthology.org/2025.emnlp-main.1187/.

[196] Wuyang Zhang, Chenkai Zhang, Zhen Luo, Jianming Ma, Wangming Yuan, Chuqiao Gu, and Chenwei Feng. Semanticforge: Repository-level code generation through semantic knowledge graphs and constraint satisfaction, 2025d. URL https://arxiv.org/abs/2511.07584.

[197] Xiangyu Zhang, Yu Zhou, Guang Yang, and Taolue Chen. Syntax-aware retrieval augmented code generation. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 1291–1302, Singapore, December 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.90. URL https://aclanthology.org/2023.findings-emnlp.90/.

[198] Yilin Zhang, Xinran Zhao, Zora Zhiruo Wang, Chenyang Yang, Jiayi Wei, and Tongshuang Wu. cast: Enhancing code retrieval-augmented generation with structural chunking via abstract syntax tree, 2025e. URL https://arxiv.org/abs/2506.15655.

[199] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, pp. 1592–1604, New York, NY, USA, 2024a. Association for Computing Machinery. ISBN 9798400706127. doi: 10.1145/3650212.3680384. URL https://doi.org/10.1145/3650212.3680384.

[200] Zhaoxi Zhang, Yitong Duan, Yanzhi Zhang, Yiming Xu, Zhixiang Wang, Kun Liang, Yang Li, Jiahui Liang, Deguo Xia, Jizhou Huang, Jiyan He, and Yunfang Wu. One tool is enough: Reinforcement learning for repository-level llm agents, 2026b. URL https://arxiv.org/abs/2512.20957.

[201] Zhe Zhang, Xingyu Liu, Yuanzhang Lin, Xiang Gao, Hailong Sun, and Yuan Yuan. Llm-based unit test generation via property retrieval, 2024b. URL https://arxiv.org/abs/2410.13542.

[202] Qianhui Zhao, Li Zhang, Fang Liu, Xiaoli Lian, Qiaoyuanhe Meng, Ziqian Jiao, Zetong Zhou, Jia Li, and Lin Shi. Fastcoder: Accelerating repository-level code generation via efficient retrieval and verification, 2025a. URL https://arxiv.org/abs/2502.17139.

[203] Yicong Zhao, Shisong Chen, Jiacheng Zhang, and Zhixu Li. Recode: Improving llm-based code repair with fine-grained retrieval-augmented generation. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, CIKM '25, pp. 4368–4378, New York, NY, USA, 2025b. Association for Computing Machinery. ISBN 9798400720406. doi: 10.1145/3746252.3761035. URL https://doi.org/10.1145/3746252.3761035.

[204] Qiming Zhu, Jialun Cao, Xuanang Chen, Yaojie Lu, Hongyu Lin, Xianpei Han, Le Sun, and Shing-Chi Cheung. Across programming language silos: A study on cross-lingual retrieval-augmented code generation, 2025. URL https://arxiv.org/abs/2506.03535.