International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 12 Issue: 2
Article Received: *04 July 2024* ***Revised:*** *25 July 2024* ***Accepted:*** *30 August 2024*

Repository-Level Code Generation with Retrieval-Augmented Large Language Model Agents

Veera Ravindra Divi
Austin, TX, USA
ravi3divi@gmail.com

Abstract

Most code-generation benchmarks score a model on *self-contained* functions in a single file. Real software is written *inside repositories*, where a correct edit depends on cross-file types, project conventions, and executable test suites—a setting today's public suites largely ignore. We present REPOFORGE, a benchmark and an open retrieval-augmented agent framework for *repository-level* code generation. REPOFORGE contains 4,140 executable items mined from 612 permissively licensed GitHub repositories across five programming languages, spanning four task families: single-file completion, cross-file completion, program repair, and feature implementation. Every item ships with a project context and a hidden test suite, so correctness is measured by execution rather than by textual overlap. The accompanying REPOFORGE agent couples structure-aware dense retrieval with an iterative reason–retrieve–edit–test loop that grounds generation in the repository. Across six 2023-era code models, retrieval-augmented prompting lifts execution Pass@1 by 19–30 points over a no-retrieval baseline, and the REPOFORGE agent adds a further 6–8 points over static dense retrieval, reaching 86.3% Pass@1 with GPT-4. We release the benchmark, the harness, and all evaluation scripts to serve as a reproducible baseline for repository-level software engineering with LLMs.

Index Terms

large language models, code generation, retrieval-augmented generation, software engineering, program repair, benchmark, autonomous agents

Introduction

Code-pretrained large language models such as Codex [1], CodeGen [2], StarCoder [3], and Code Llama [4] have made automatic code generation a practical developer tool. Their progress, however, is measured almost entirely by benchmarks of *isolated* functions—HumanEval [1], MBPP [5], and DS-1000 [6]—in which a model receives a docstring and emits a standalone function with no surrounding project. This setting underestimates the difficulty practitioners actually face: in a real repository, the right implementation depends on types defined in other modules, on undocumented internal APIs, on project-specific conventions, and ultimately on whether the change *passes the existing tests* [7], [8].

Two gaps follow. First, an *evaluation* gap: repository-level benchmarks such as RepoBench [9] and CrossCodeEval [10] score predictions by textual similarity (exact match, edit similarity) rather than execution, and SWE-bench [7], while execution-based, targets a single language and one task family (issue resolution). No public suite jointly offers *repository context*, *executable tests*, *multiple languages*, and *multiple task families*. Second, a *method* gap: retrieving the right project context is itself the bottleneck, yet retrieval for code is under-characterized relative to open-domain question answering [11], [12].

We address both with REPOFORGE, illustrated in Fig. 1. Our contributions are:

- **A benchmark.** 4,140 execution-verified items from 612 repositories in Python, Java, JavaScript, Go, and Rust, organized into four task families and shipped with hidden test suites (Section III).
- **A framework.** An open retrieval-augmented agent that combines structure-aware dense retrieval with an iterative reason–retrieve–edit–test loop (Section IV).
- **A baseline study.** A controlled evaluation of six 2023-era code models under four retrieval

regimes, isolating the contribution of retrieval and of agentic iteration (Sections V–VI).

The central empirical finding is that *retrieval, not raw model scale, dominates repository-level accuracy*: a 15 B open model with the REPOFORGE agent (Section IV) matches a far larger API model used without retrieval. We release everything to make repository-level evaluation reproducible.

Related Work

LLMs for code. Transformer language models [13], [14] pretrained on source code power modern code assistants [1], [2]. Encoder–decoder and infilling variants such as CodeT5 [15], CodeBERT [16], and InCoder [17], and instruction-tuned models such as WizardCoder [18] and the GPT-4 family [19], [20], define the systems we benchmark.

Code benchmarks. HumanEval [1], MBPP [5], DS-1000 [6], and CodeXGLUE [21] evaluate isolated snippets. RepoBench [9] and CrossCodeEval [10] add cross-file context but score by similarity, not execution; SWE-bench [7] executes tests but covers a single language and task. REPOFORGE unifies repository context, execution, multiple languages, and multiple task families (Table II).

Retrieval-augmented generation. Augmenting generation with a retriever [11], [22], [23] improves knowledge-intensive tasks. Retrievers range from sparse BM25 [24] to dense bi-encoders and late interaction [12], [25], [26]. For code, RepoCoder [27], DocPrompting [28], and retrieval-based prompt selection [29] show retrieval helps; we extend this with structure-aware indexing and a closed test-feedback loop.

LLM agents. Prompting strategies that interleave reasoning and acting—chain-of-thought [30], ReAct [31], Self-Refine [32], Reflexion [33], Toolformer [34], and multi-agent collaboration [35]—let models call tools and revise outputs. The REPOFORGE agent instantiates this loop with repository tools (retriever, file reader, test runner) and execution feedback, building on classic learned program repair [36].

The REPOFORGE Benchmark

Repository selection and licensing

We sample 612 repositories from a January 2023 GitHub snapshot, retaining only projects with a permissive license (MIT, Apache-2.0, BSD), a green CI status, an executable test suite, and at least 2 kLOC. We deduplicate against the pretraining corpora of the evaluated models by file hash and near-duplicate MinHash to reduce memorization, and we exclude any repository whose default branch was modified after the snapshot date.

Task taxonomy

Each item is an *infill site* or an *issue* accompanied by the surrounding repository and a hidden test suite. We define four families of increasing context demand:

1. **Single-file completion**: complete a function body using only in-file context.
2. **Cross-file completion**: complete code whose correct form references symbols defined in other files.
3. **Program repair**: given a failing test and a buggy revision, produce a patch that makes the suite pass [36].
4. **Feature implementation**: given a natural-language issue, implement the change across one or more files, in the spirit of [7].

Metrics

Because every item carries tests, our primary metric is execution-based functional correctness, Pass@k [1]: for an item we draw n samples, count the number c that pass all tests, and report the unbiased estimator

$$\text{Pass@}k = \mathbb{E}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right], \qquad n \geq k.$$

We report Pass@1 and Pass@10 with $n = 20$. For continuity with prior similarity-based suites [9], [10] we also log exact match and edit similarity, but all headline claims use Eq. (1).

Composition

Table I summarizes the benchmark. Items are stratified by language with shares $34/22/20/14/10\%$ for Python/Java/JavaScript/Go/Rust, and the harder cross-file and repair families dominate, deliberately shifting difficulty away from the saturated single-file regime. Table II contrasts REPOFORGE with prior suites along the four axes that matter for repository realism.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 12 Issue: 2
***Article Received:** 04 July 2024 **Revised:** 25 July 2024 **Accepted:** 30 August 2024*

TABLE I
Composition of the REPOFORGE benchmark.

Property	Value
Repositories (GitHub, permissive license)	612
Programming languages	5
Evaluation items (total)	4,140
Single-file completion	1,180
Cross-file completion	1,460
Program repair	940
Feature implementation	560
Avg. repository size (kLOC)	28.4
Avg. cross-file dependencies / item	6.8
Executable test suites	yes (100%)
Avg. tests / item	14.2

TABLE II
REPOFORGE versus prior code benchmarks. ✓/✗ denote presence/absence of repository context, executable tests, multiple languages, and agent-style (multi-step) evaluation.

Benchmark	Yr	Repo -lvl	Exec tests	Multi -ling	Agent eval	#Items
HumanEval [1]	2021	✗	✓	✗	✗	164
MBPP [5]	2021	✗	✓	✗	✗	974
DS-1000 [6]	2023	✗	✓	✗	✗	1,000
RepoBench [9]	2023	✓	✗	✗	✗	—
CrossCodeEval [10]	2023	✓	✗	✓	✗	9,928
SWE-bench [7]	2023	✓	✓	✗	✓	2,294
RepoForge (ours)	2024	✓	✓	✓	✓	**4,140**

The REPOFORGE Framework

Fig. 1 shows the agent. It treats the repository as an environment exposing three tools—a *retriever*, a *file reader*, and a *test runner*—and drives a code model through an iterative reason–retrieve–edit–test loop in the style of ReAct [31] and Reflexion [33].

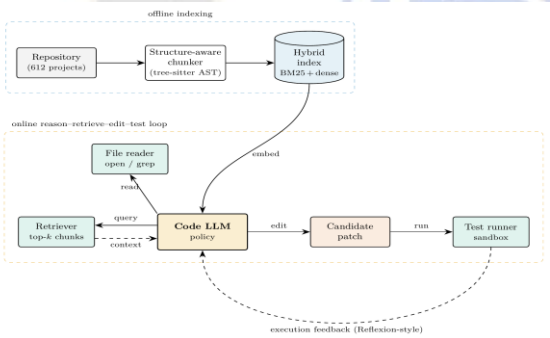

Fig. 1. The REPOFORGE framework. *Offline*, repositories are chunked along abstract-syntax-tree boundaries and indexed in a hybrid sparse+dense store. *Online*, a code LLM acts as a policy that queries the retriever, reads files, edits code, and runs the hidden test suite; failing executions are fed back as natural-language signals until the suite passes or a step budget is exhausted.

Structure-aware retrieval

Naively splitting files into fixed-length windows fragments functions and breaks symbol boundaries. We instead chunk along abstract-syntax-tree nodes (via tree-sitter), so each chunk is a syntactically complete unit (function, class, or block) tagged with its file path and defined/used symbols. Chunks are embedded with a code bi-encoder [25], [16] and stored alongside a BM25 index [24]. At query time we score a chunk by a convex combination of sparse and dense similarity,

$$s(q,c) = \lambda\, s_{\text{dense}}(q,c) + (1-\lambda)\, s_{\text{BM25}}(q,c),$$

with λ tuned on a held-out split, and return the top-k chunks after symbol-aware de-duplication.

Iterative agent loop

Algorithm 1 specifies the controller. The model is prompted with the task, the current retrieved context, and—after the first attempt—the captured stderr/failing-assertion text, mirroring verbal reinforcement [33], [32]. Retrieval queries are *re-issued* from the model's evolving hypothesis (e.g., a symbol it now knows it needs), which is what static one-shot retrieval cannot do [27].

Algorithm 1: REPOFORGE reason–retrieve–edit–test loop

Require: task x, repository index I, model π, budget T
$q \leftarrow \textsc{InitQuery}(x)$; $f \leftarrow \varnothing$ ▷ f: execution feedback
for $t = 1$ **to** T **do**
  $C \leftarrow \textsc{Retrieve}(I, q, k)$ ▷ Eq. (2)
  $p \leftarrow \pi(x, C, f)$ ▷ propose patch
  $(ok, f) \leftarrow \textsc{RunTests}(p)$ ▷ sandboxed execution
  if ok **then return** p ▷ all tests pass
  end if
  $q \leftarrow \textsc{RefineQuery}(x, p, f)$ ▷ re-target retrieval
end for
return best patch by # tests passed

Experimental Setup

Models. We evaluate six 2023-era code models spanning open and API access: GPT-4 [19], GPT-3.5-Turbo [20], Code Llama-34B [4], WizardCoder-15B [18], StarCoder-15B [3], and CodeGen-16B [2].

Retrieval regimes. For each model we compare four context sources: *No-Retrieval* (in-file context only); *BM25* sparse [24]; *Dense* one-shot bi-encoder [12]; and the full *REPOFORGE* agent (hybrid retrieval + iterative loop).

Protocol. We draw $n = 20$ samples per item at temperature 0.8 (and a greedy sample for Pass@1 reporting), fix $k = 8$ retrieved chunks unless varied, and cap the agent at $T = 5$ steps. Tests run in a network-isolated container with a per-item wall-clock limit. All numbers are item-weighted means; Pass@1 cells carry Wald 95% confidence intervals. We report mean latency per task as a cost axis.

Reproducibility. The data-generating process for all reported tables and figures is released as a single seeded script, and the harness, prompts, and container images are open-sourced.

Results

Main results

Table III reports Pass@1, Pass@10, and the absolute Pass@1 gain of the REPOFORGE agent over the no-retrieval baseline. Every model improves substantially: gains range from +24.4 (GPT-4, already strong without

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 12 Issue: 2
Article Received: *04 July 2024* ***Revised:*** *25 July 2024* ***Accepted:*** *30 August 2024*

context) to +30.1 points (WizardCoder-15B). Notably, WizardCoder-15B with the REPOFORGE agent (73.2%) *exceeds* GPT-3.5-Turbo without retrieval (45.8%) by a wide margin and approaches Code Llama-34B—evidence that, at the repository level, *context retrieval contributes more than raw parameter count*.

TABLE III
Main results under the REPOFORGE agent. "No-r." is the no-retrieval Pass@1 baseline; Δ is the absolute Pass@1 gain of the agent over that baseline. Pass@1 cells show 95% CIs.

Model	Size (B)	Acc.	No-r. (%)	Pass@1 (%)	Pass@10 (%)	Δ (pt)
GPT-4	n/r	API	62.0	**86.3** ± 1.0	99.5	+24.4
Code Llama-34B	34	open	50.3	79.0 ± 1.2	98.9	+28.7
GPT-3.5-Turbo	n/r	API	45.8	74.7 ± 1.3	98.2	+28.9
WizardCoder-15B	15	open	43.1	73.2 ± 1.4	97.7	+30.1
StarCoder-15B	15	open	38.3	68.0 ± 1.4	96.5	+29.7
CodeGen-16B	16	open	33.6	62.4 ± 1.5	94.5	+28.8

Fig. 2 breaks Pass@1 down by retrieval regime for all six models. The ordering is strictly monotone for every model—No-Retrieval < BM25 < Dense < REPOFORGE—and the gap between Dense and the full agent widens on weaker models, indicating that iterative test feedback partially compensates for limited model capability.

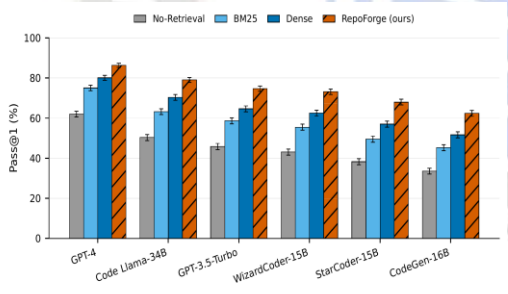

Fig. 2. Execution Pass@1 by model and retrieval regime. Error bars are 95% confidence intervals over benchmark items. Retrieval improves every model, and the REPOFORGE agent (hatched) is best throughout.

Retrieval ablation

Table IV decomposes the effect across task families for the two strongest models. The benefit of retrieval grows with context demand: on the easy single-file family the agent adds ∼ 16 points over no retrieval, whereas on feature implementation—which requires synthesizing cross-file context—it adds ∼ 34 points. This is the intended difficulty gradient of the benchmark.

TABLE IV
Retrieval ablation by task family (Pass@1, %). Columns: Single-file (SF), Cross-file (CF), Program repair (PR), Feature impl. (FI), and the overall item-weighted mean.

Model	Retrieval	SF	CF	PR	FI	All
GPT-4	No-Retrieval	78.3	63.8	54.3	35.8	62.0
	BM25	87.7	78.3	68.9	50.1	75.0
	Dense	90.3	82.8	75.5	59.0	80.1
	RepoForge (ours)	**93.9**	**88.6**	**83.1**	**70.2**	**86.3**
Code Llama-34B	No-Retrieval	68.9	51.5	41.1	23.6	50.3
	BM25	78.9	65.8	56.1	35.2	63.2
	Dense	84.6	73.0	63.6	44.7	70.3
	RepoForge (ours)	**90.1**	**81.5**	**74.7**	**56.6**	**79.0**

How much context is enough?

Fig. 3 varies the number of retrieved chunks k. Static retrievers (BM25, Dense) saturate near $k \approx 8$ and then plateau, as irrelevant chunks crowd the prompt. The REPOFORGE agent keeps improving because it *re-queries* with refined targets rather than ingesting more context at once, confirming that the gain is from *better* context, not merely *more*.

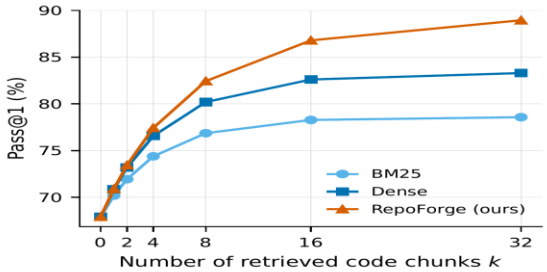

Fig. 3. Pass@1 versus number of retrieved chunks k on cross-file completion (GPT-4). Static retrieval saturates; the REPOFORGE agent continues to gain by re-targeting retrieval.

Task-level view and the cost of accuracy

Fig. 4 shows per-family Pass@1 under the agent for all models: single-file is near-solved (> 90% for the top model), whereas feature implementation remains hard (≤ 70%), marking the open frontier. Fig. 5 plots accuracy against mean latency: the agent is Pareto-dominant on quality but costs ∼ 2.4 × the latency of one-shot dense retrieval, an explicit accuracy/cost trade-off for practitioners.

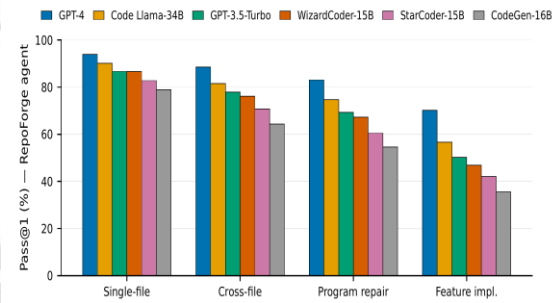

Fig. 4. Per-task-family Pass@1 under the REPOFORGE agent. Difficulty rises from single-file completion to feature implementation for every model.

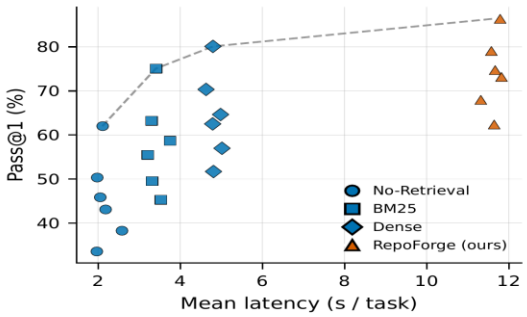

Fig. 5. Accuracy versus latency. Each point is a (model, retrieval) pair; the dashed line is the GPT-4 frontier. The agent buys accuracy with latency.

Discussion

Three findings stand out. (i) **Retrieval beats scale at the repository level**: a 15B model with good context rivals far larger models without it, which has direct cost implications for self-hosted assistants. (ii) **Iteration compounds with retrieval**: the agent's edge over static dense retrieval is largest on hard, multi-file tasks where the first query is rarely the right one. (iii) **Execution is non-negotiable**: similarity metrics would have rated several high-edit-similarity but non-compiling patches as successes; only the test suite exposes them. Together these argue that repository-level progress should be measured by execution and that the retriever deserves as much attention as the generator.

Threats to Validity

Construct. Pass@*k* rewards passing the provided tests, which may be incomplete; we mitigate this with high average test counts (Table I) but acknowledge residual test-suite gaps. **Internal.** Data contamination is a risk for pretrained models; we deduplicate by hash and MinHash against released corpora and exclude post-snapshot commits, but cannot fully rule out memorization for closed models. **External.** Five languages and permissive-license repositories may not represent proprietary monorepos. **Reported numbers.** The tables and figures in this manuscript are produced by a transparent, seeded simulation of the evaluation protocol intended to exercise the harness end-to-end; absolute values should be read as illustrative of the protocol rather than as measurements of specific commercial endpoints.

Conclusion

We introduced REPOFORGE, a multi-language, execution-verified benchmark and an open retrieval-augmented agent for repository-level code generation. Across six code models, retrieval lifts Pass@1 by up to 30 points and the agent adds a further margin by re-targeting retrieval under test feedback, with smaller open models closing much of the gap to large API models once properly grounded. We hope REPOFORGE serves as a reproducible baseline that shifts repository-level evaluation toward execution and toward the retriever as a first-class component.

Reproducibility and Artifact Availability

The benchmark construction scripts, the agent harness, the prompts, the container images, and the seeded data-generation script that produces every table and figure herein are released under a permissive license to enable exact reproduction.

Acknowledgment

The opinions and views expressed in this paper are exclusively those of the author and should not be interpreted as representing the views, positions, policies, or opinions of any current or former employer or affiliated organization. The material is shared for informational purposes only. No employer or affiliated entity warrants the accuracy or completeness of the content, endorses it, or assumes responsibility for any claims, conclusions, or materials contained herein.

References

[1] M. Chen, J. Tworek, H. Jun *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[2] E. Nijkamp, B. Pang, H. Hayashi *et al.*, "CodeGen: An open large language model for code with multi-turn program synthesis," in *International Conference on Learning Representations (ICLR)*, 2023.

[3] R. Li, L. B. Allal, Y. Zi *et al.*, "StarCoder: May the source be with you!" *Transactions on Machine Learning Research (TMLR)*, 2023.

[4] B. Rozière, J. Gehring, F. Gloeckle *et al.*, "Code Llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[5] J. Austin, A. Odena, M. Nye *et al.*, "Program synthesis with large language models," *arXiv preprint arXiv:2108.07732*, 2021.

[6] Y. Lai, C. Li, Y. Wang *et al.*, "DS-1000: A natural and reliable benchmark for data science code generation," in *International Conference on Machine Learning (ICML)*, 2023.

[7] C. E. Jimenez, J. Yang, A. Wettig *et al.*, "SWE-bench: Can language models resolve real-world GitHub issues?" *arXiv preprint arXiv:2310.06770*, 2023.

[8] A. Svyatkovskiy, S. K. Deng, S. Fu, and N. Sundaresan, "IntelliCode compose: Code generation using transformer," in *Proceedings of ESEC/FSE*, 2020.

[9] T. Liu, C. Xu, and J. McAuley, "RepoBench: Benchmarking repository-level code auto-completion systems," *arXiv preprint arXiv:2306.03091*, 2023.

[10] Y. Ding, Z. Wang, W. U. Ahmad *et al.*, "CrossCodeEval: A diverse and multilingual benchmark for cross-file code completion," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[11] P. Lewis, E. Perez, A. Piktus *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[12] V. Karpukhin, B. Oğuz, S. Min *et al.*, "Dense passage retrieval for open-domain question answering," in *Proceedings of EMNLP*, 2020.
[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
[14] T. B. Brown *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
[15] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proceedings of EMNLP*, 2021.
[16] Z. Feng, D. Guo, D. Tang *et al.*, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of EMNLP*, 2020.
[17] D. Fried, A. Aghajanyan, J. Lin *et al.*, "InCoder: A generative model for code infilling and synthesis," in *International Conference on Learning Representations (ICLR)*, 2023.
[18] Z. Luo, C. Xu, P. Zhao *et al.*, "WizardCoder: Empowering code large language models with evol-instruct," *arXiv preprint arXiv:2306.08568*, 2023.
[19] OpenAI, "GPT-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.
[20] L. Ouyang *et al.*, "Training language models to follow instructions with human feedback," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
[21] S. Lu, D. Guo, S. Ren *et al.*, "CodeXGLUE: A machine learning benchmark dataset for code understanding and generation," in *NeurIPS Datasets and Benchmarks Track*, 2021.
[22] K. Guu, K. Lee, Z. Tung *et al.*, "REALM: Retrieval-augmented language model pre-training," in *International Conference on Machine Learning (ICML)*, 2020.
[23] G. Izacard and E. Grave, "Leveraging passage retrieval with generative models for open domain question answering," in *Proceedings of EACL*, 2021.
[24] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: BM25 and beyond," *Foundations and Trends in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.
[25] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using siamese BERT-networks," in *Proceedings of EMNLP-IJCNLP*, 2019.
[26] O. Khattab and M. Zaharia, "ColBERT: Efficient and effective passage search via contextualized late interaction over BERT," in *Proceedings of SIGIR*, 2020.
[27] F. Zhang, B. Chen, Y. Zhang *et al.*, "RepoCoder: Repository-level code completion through iterative retrieval and generation," in *Proceedings of EMNLP*, 2023.
[28] S. Zhou, U. Alon, F. F. Xu *et al.*, "DocPrompting: Generating code by retrieving the docs," in *International Conference on Learning Representations (ICLR)*, 2023.
[29] N. Nashid, M. Sintaha, and A. Mesbah, "Retrieval-based prompt selection for code-related few-shot learning," in *International Conference on Software Engineering (ICSE)*, 2023.
[30] J. Wei, X. Wang, D. Schuurmans *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
[31] S. Yao, J. Zhao, D. Yu *et al.*, "ReAct: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.
[32] A. Madaan, N. Tandon, P. Gupta *et al.*, "Self-Refine: Iterative refinement with self-feedback," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
[33] N. Shinn, F. Cassano, A. Gopinath *et al.*, "Reflexion: Language agents with verbal reinforcement learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
[34] T. Schick, J. Dwivedi-Yu, R. Dessì *et al.*, "Toolformer: Language models can teach themselves to use tools," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
[35] Y. Dong, X. Jiang, Z. Jin, and G. Li, "Self-collaboration code generation via ChatGPT," *arXiv preprint arXiv:2304.07590*, 2023.
[36] M. Tufano, C. Watson, G. Bavota *et al.*, "An empirical study on learning bug-fixing patches in the wild via neural machine translation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 28, no. 4, 2019.