VibeRepair+: Token-Saving LLM-Driven Automatic Program Repair for Java and Python

Yihang Jiao
University of Illinois
Urbana-Champaign
yihangj3@illinois.edu

Yingjia Gu
University of Illinois
Urbana-Champaign
gu42@illinois.edu

Shengyue Guo
University of Illinois
Urbana-Champaign
guo79@illinois.edu

Abstract

Automated Program Repair (APR) has advanced rapidly with LLMs, yet code-centric approaches remain prone to hallucination and high token costs. We present **VibeRepair+**, extending the specification-centric VibeRepair paradigm to Java and Python with three improvements: (i) a hybrid retrieval algorithm based on Reciprocal Rank Fusion (RRF) over parallel dense and lexical retrievers; (ii) LLM-based distillation that compresses fix guidance; and (iii) removal of external tool dependency, cutting token consumption from 1,564 to ∼150 tokens per query (∼90% savings). Our pluggable retriever framework enables systematic comparison over a 97,402-example Java bug-fix corpus. Beyond self-LOO sanity (which we show is uninformative), external-query evaluation on 521 Defects4J bugs reveals BM25 outperforms the ada baseline (binary recall@10 = 0.177 vs. 0.138, +27.8% relative) and hybrid RRF leads on five of six quality metrics (recall@10 = 0.190), with 15 unique top-10 hits not captured by either sub-retriever. On Defects4J, VibeRepair+ outperforms the original VibeRepair across all four tested project categories with Qwen Coder 2.5; the Python port using Claude Sonnet 4.6 repairs 26 of 30 BugsInPy single-function bugs (87%).

1 Introduction

Automated Program Repair (APR) aims to reduce developer effort by automatically generating correct patches. Early symbolic approaches such as GenProg (Le Goues et al., 2012) explored genetic search over the patch space, and learning-based methods like SequenceR (Chen et al., 2021) applied sequence-to-sequence models. The emergence of LLMs has accelerated this field: ChatRepair (Xia and Zhang, 2024), AlphaRepair (Xia et al., 2023), and ReinFix (Zhang et al., 2025) achieve strong results on the Defects4J Java benchmark (Just et al., 2014) by prompting LLMs to rewrite buggy code directly.

Despite their success, code-centric LLM repair suffers from a fundamental limitation: it asks the model to simultaneously understand program semantics, infer developer intent, and synthesize correct code in one step. This conflation yields hallucinated fixes—patches that appear syntactically valid but are behaviorally inconsistent with the original specification (Zhu et al., 2026). VibeRepair (Zhu et al., 2026) addresses this by decomposing repair into Transformation (buggy code → behavior spec), Repair (LLM spec correction), and Generation (spec → code). An optional reasoning component enriches difficult cases with RAG over historical bug-fix evidence and static-analysis tools.

Our project, VibeRepair+, extends this framework in two directions. First, we redesign the reasoning component to employ a hybrid retrieval algorithm (parallel RRF over dense and lexical retrievers) and LLM-based distillation, reducing token overhead while improving retrieval relevance. Second, we optimize the pipeline by removing the Java-specific diagnose tool dependency so that the pipeline extends to other languages. We port the full pipeline to Python by integrating BugsInPy (Widyasari et al., 2020) with PyResBugs (Cotroneo et al., 2025) as the RAG database.

2 Related Work

LLM-Based APR. LLM-based APR splits into prompting-based and agentic paradigms. ChatRepair feeds failing tests iteratively to ChatGPT; RepairAgent (Bouzenia et al., 2025) uses an autonomous LLM agent with tools; AlphaRepair uses cloze-style prompting; ThinkRepair (Yin et al., 2024) adopts chain-of-thought reasoning. ReinFix, our direct predecessor, uses OpenAI `text-embedding-3-large` to retrieve similar historical fixes as few-shot context. However, ReinFix and VibeRepair ship a single monolithic retriever

and never ablate retrieval strategy—a gap we address.

Specification-Centric Repair. VibeRepair grounds repair in explicit behavior specifications. SpecRover (Ruan et al., 2025) similarly extracts code intent via LLMs; TENURE (Wen et al., 2018) employs test-guided edit sequences. VibeRepair uniquely combines a structured spec-repair loop with an on-demand reasoning component.

RAG for APR. BM25-Okapi (Robertson and Zaragoza, 2009) remains a strong lexical baseline. Dense retrievers excel at semantic similarity but have higher latency. Hybrid retrieval via RRF (Cormack et al., 2009) has shown consistent gains across IR benchmarks. Our work provides the first systematic retriever comparison specifically for APR.

3 VibeRepair+ Architecture

3.1 Overall Pipeline

VibeRepair+ preserves the three-phase pipeline of VibeRepair while enhancing the Reasoning Component in the Repair Phase. The Transformation Phase converts buggy function and failing tests into a behavior specification; the Repair Phase corrects misalignments (optionally with the Reasoning Component); the Generation Phase synthesizes code and validates against the test suite. Core models are Qwen 2.5 Coder 14B for code–spec translation and Qwen 3 Embedding 8B for dense retrieval (64 GB RAM, 24 GB VRAM).

3.2 Enhanced Reasoning Component

The key contribution of VibeRepair+ is a redesigned Reasoning Component with three improvements. First, we adopt a **hybrid retrieval algorithm** based on Reciprocal Rank Fusion (RRF) (Cormack et al., 2009). Two retrievers run in parallel over the full 97k Java bug-fix corpus: ada (OpenAI `text-embedding-3-large`, dense, 32 ms p50) and BM25-Okapi (lexical, ∼4,400 ms p50). Each contributes its top-100 candidates, fused by RRF rank summation with $k = 60$. This parallel architecture preserves both retrievers' visibility into the full corpus and exploits their complementarity: ada and BM25 agree on top-1 only 34.1% of the time but produce top-10 sets that agree 82% of the time, meaning they diverge in ranking rather than candidate identity. A two-stage cascade variant (ada filters to top-100, BM25 reranks) would reduce latency from ∼4.2 s to ∼37 ms (113× speedup), at the cost of losing BM25's full-corpus visibility; this is left as future work.

Retriever	Type	r@1	MRR@10	p50
ada (P1)	Dense	1.000	1.000	32 ms
BM25 (P2)	Lexical	1.000	1.000	4,400 ms
hybrid (P3)	RRF, k=60	1.000	1.000	4,180 ms

Table 1: Self-LOO evaluation. All retrievers achieve perfect scores on in-corpus queries (Section 4.3 shows this is a self-match artifact).

Second, we add **LLM-based distillation with selective feedback**: retrieved bug-fix examples are distilled into a compact fix-guidance snippet by a lightweight LLM pass. Only examples with cosine similarity above a tunable threshold are included, discarding misleading low-relevance matches.

Third, we **reduce dependence on external tools**. The original reasoning agent calls static analysis tools in arbitrary order, leading to tool-chain errors (e.g., test projects opened without being closed) and long-context hallucinations on local models. VibeRepair+ restructures the agent prompt to encourage direct reasoning over the distilled context, yielding ∼150 tokens—a 90% reduction from 1,564 tokens.

3.3 Python Extension (BugsInPy)

We replace four Java-specific components: (i) benchmark (Defects4J → BugsInPy), (ii) LLM backend (GPT-4o → Claude Sonnet 4.6 via `langchain_anthropic`), (iii) RAG database (ReinFix → PyResBugs), and (iv) validator (`defects4j test` → `bugsinpy-compile` + `pytest`). The repair loop runs up to 5 outer attempts × 3 inner spec-refinement turns; multi-function bugs are skipped.

4 Retriever Evaluation

4.1 Pluggable Retriever Framework

We refactored the original monolithic ReinFix Flask server into a pluggable architecture. Adding a new retriever requires writing one Python file; the HTTP `/search` endpoint, evaluation harness, and metrics (recall@k, MRR, nDCG, latency) require zero modification. Reproducibility is enforced via a corpus fingerprint (SHA-256) and locked holdout split.

4.2 Phase 1 (ada) and Phase 2 (BM25)

We replicate the ReinFix ada retriever over 97,402 Java bug-fix examples embedded as 3,072-dim vectors. Under leave-one-out (LOO) self-evaluation on 1,000 queries, all metrics equal 1.000 with 32 ms p50 latency. BM25-Okapi, plugged in as the second retriever, also achieves recall@1 = 1.000 but at 4,400 ms p50—a 138× slowdown. Both Phase 1 and Phase 2 are thus indistinguishable under self-LOO. External evaluation in Section 4.6 reverses this, with BM25 outperforming ada by +27.8% relative recall@10—inverting the conventional dense-beats-lexical assumption.

4.3 Diagnostic: Self-LOO Is Not Quality

Three diagnostics confirm Phase 1/2 scores are artifacts: zero byte-level duplicates exist in the 97,402 corpus; after excluding self, ada top-1 cosine drops to mean 0.75; cross-retriever comparison on 900 shared queries shows top-1 agreement of only 34.1% and top-10 Jaccard of 0.166. This motivates external-query evaluation (Sections 4.5–4.7) and hybrid retrieval via RRF.

4.4 External Tool Removal

The original pipeline uses Joern for runtime analysis as part of the bug–code search pair. We claim the buggy code itself contains the bug signal, so the tool dependency is unnecessary and the original bug+code embedding concatenation is a naïve combination that hurts retrieval. Using buggy code alone as the search index removes the diagnose tools and makes the pipeline language-agnostic.

4.5 External-Query Evaluation Setup

To replace self-LOO, each of the 273 single-function Defects4J bugs serves as a real-world retrieval query whose buggy code is fed to the retriever; relevance is measured against a proxy ground truth derived from the corresponding fix patch.

Proxy ground truth. We extract modified lines via unified diff (`difflib`) and compute the Jaccard of token sets using an operator-aware tokenizer that captures comparison, boolean, and arithmetic operators (recovering 170 of 171 bugs that a word-only tokenizer left with empty deltas due to operator-only fixes such as `==` → `!=`). The top-5 corpus rows by Jaccard form the proxy positive set ($|\text{gold}| = 5$).

Retr.	r@1	r@5	r@10	MRR	nDCG	p50
ada	.083	.109	.138	.097	.044	285 ms
BM25	.085	.138	.177	.111	.055	3,756 ms
hybrid	**.086**	**.154**	**.190**	**.114**	.055	4,247 ms

Table 2: External-query retrieval on 521 Defects4J bugs. All three achieve self-LOO recall@1 = 1.000 (Table 1); the gap is the central finding of Section 4.3.

Leak check. A two-stage check (ada cosine $>$ 0.95 + Levenshtein normalized $>$ 0.95) detects *zero* byte-overlap, confirming a genuinely external evaluation.

Final eval set. 521 of 533 bugs after leak filtering (0 removed), empty-delta filter (1: Lang-53), and skipping bugs lacking annotations (11).

Metrics. Primary: query-level binary hit rate (recall@k = 1 if any of $K = 5$ positives is in top-k). Secondary: gold-coverage $|\text{hits}[:k] \cap \text{gold}|/|\text{gold}|$ (Robertson and Zaragoza, 2009). Random retrieval yields binary recall@10 $\approx 5\times10^{-4}$; ada at 0.138 represents a 276× improvement, so retrieval is meaningfully discriminative.

4.6 BM25 vs. ada on External Queries

Table 2 reports retrieval quality on the 521 external queries. BM25 consistently outperforms ada on all five quality metrics; binary recall@10 is 0.177 vs. 0.138—a +27.8% relative gain ($\Delta = +0.038$, above the ±0.02 standard-error band at $N = 521$).

This inverts the dense-beats-lexical assumption: Java bug-fix patterns are dominated by domain-specific tokens (API names, class paths, method signatures) for which BM25's IDF weighting is particularly effective. BM25's quality comes at a 13× latency cost (p50 3,756 ms vs. 285 ms; p99 38.5 s vs. 3.1 s), with p99 alone exceeding typical ∼30 s LLM-agent per-turn budgets.

4.7 Hybrid RRF Results and Oracle Ceiling

Hybrid RRF achieves binary recall@10 = 0.190, leading on five of six quality metrics (Table 2). The improvement over BM25 ($\Delta = +0.013$) falls within the ±0.02 standard-error band; the stronger evidence of hybrid's value is per-bug behavior (Table 3).

The ada∪BM25 oracle ceiling is $110/521 = 0.211$—the upper bound of any fusion of these two retrievers. 15 bugs (2.9%) are hit by hybrid yet missed by both sub-retrievers: cases where the

Bucket (top-10 hit)	Count	Frac.
All three (A ∩ B ∩ H)	53	0.102
ada + BM25 only (\H)	1	0.002
ada + hybrid only (\B)	7	0.013
BM25 + hybrid only (\A)	24	0.046
ada only	11	0.021
BM25 only	14	0.027
hybrid only (unique RRF gain)	**15**	**0.029**
Missed by all three	396	0.760
ada ∪ BM25 oracle ceiling	110	0.211
hybrid ∩ oracle (76.4% realized)	84	0.161

Table 3: Per-bug top-10 hit decomposition on 521 Defects4J bugs.

Project	VR	VR+	Δ	Bugs
Chart	53%	75%	+22 pp	17
Closure	22%	43%	+21 pp	96
Lang	63%	68%	+5 pp	41
Math	49%	55%	+6 pp	76

Table 4: Plausible repair rates on Defects4J subset.

proxy positive lies in one sub-retriever's rank 11–100 and RRF surfaces it by combining tail rank evidence from both retrievers' top-100 candidates. These unique hits cluster on Math (6 bugs) and JacksonDatabind (4 bugs). Hybrid realizes 84 of the 110 oracle hits (76.4%); the 24% gap reflects RRF's fairness property, motivating future ablation of k and per-retriever top-K, and exploration of cascade architectures.

5 Experimental Results

5.1 Defects4J: VibeRepair vs. VibeRepair+

We evaluated a subset of Defects4J v1.2, comparing the original VibeRepair (Qwen, reasoning mode) against VibeRepair+ with the improved distillation. The external-tool-independent ∼150-token prompt replaces the original 1,564-token prompt (∼90% reduction).

VibeRepair+ shows the largest gains on Chart and Closure (+22 pp, +21 pp), where distilled domain-specific guidance helps most; for Closure the baseline suffered tool-chain failures that accumulated into memory overflow. Gains are smaller but positive for Lang and Math, which contain more diverse bug patterns.

5.2 Local Deployment with Qwen

Running locally with Qwen 2.5 Coder 14B is constrained by capacity and long-context hallucination, yielding competitive but lower rates than GPT-4o-backed baselines. Our improvements were not verified on online frontier LLMs; with stronger models, the relative gains from distillation and tool removal may be smaller.

5.3 BugsInPy: Python Repair

On 30 single-function Python bugs from 7 projects (`luigi`, `fastapi`, `scrapy`, `black`, `sanic`, `spacy`, `PySnooper`), VibeRepair+ with Claude Sonnet 4.6 repairs 26 bugs (87% plausible repair rate). `luigi` (12/12) shows perfect repair; `fastapi` (3/5) is the most challenging. This confirms the specification-centric paradigm generalizes to Python and Claude Sonnet 4.6 is a viable drop-in replacement for GPT-4o. However, the Python RAG corpus (∼5k bugs) is much smaller than the Java one (∼97k), making exact RAG match difficult.

6 Limitations and Future Work

Several limitations apply. Qwen 2.5 Coder 14B suffers from long-context hallucination; stronger LLMs may absorb some of our pipeline gains. Multi-function bugs are currently skipped. The Python RAG corpus is much smaller than Java. The D-eval proxy ground truth is token-overlap, not program-equivalence, so absolute recall numbers are lower bounds. Hybrid RRF inherits BM25's latency (p99 = 38.9 s); a cascade variant could cut this to ∼37 ms.

Future work: (1) validate the pipeline with frontier online LLMs to disentangle retrieval and capacity gains; (2) implement the cascade RRF variant and ablate k / per-retriever top-K (hybrid currently realizes only 76.4% of the oracle ceiling); and (3) build a larger Python bug-fix RAG corpus and extend coverage to multi-function bugs.

7 Conclusion

VibeRepair+ extends specification-centric APR to Java and Python with: a hybrid retrieval algorithm (parallel RRF, $k = 60$); LLM-based distillation; external-tool-independent prompts (90% token reduction); a pluggable retriever framework; and a Python port (87% plausible repair on BugsInPy with Claude Sonnet 4.6). The external-query evaluation on 521 Defects4J bugs reveals three findings absent in the original ReinFix evaluation: BM25 outperforms ada by 27.8% despite identical self-LOO sanity; ada and BM25 are complementary (oracle ceiling 21.1% vs. 13.8% / 17.7%); and hybrid RRF leads on five of six metrics with 15 unique top-10 hits. Together, these results confirm that

specification-centric repair generalizes across languages, retrieval strategy materially affects downstream APR quality, and structured information distillation is a practical lever for cost reduction.

Contributions

Yihang Jiao (yihangj3) designed the pluggable retriever framework and the dual binary/coverage metrics; implemented and benchmarked the three retrievers (ada Phase 1, BM25 Phase 2, parallel-RRF hybrid Phase 3); designed the diagnostic that established self-LOO as uninformative (34.1% top-1 agreement, 0.166 Jaccard); designed the external-query evaluation methodology (operator-aware token-overlap proxy GT, two-stage ada+Levenshtein leak check); executed the full D-eval (1,563 retrieval calls on 521 Defects4J bugs); and produced the three D-eval findings: BM25 wins by +27.8% over ada, ada∪BM25 oracle ceiling of 0.211, and 15 unique hybrid-only hits realizing 76.4% of that ceiling (Sections 4.3–4.7, Tables 1–3).

Yingjia Gu (gu42) ported the VibeRepair pipeline from Java to Python, replacing the Defects4J validator with a `bugsinpy-compile + pytest` runner inside `pyenv`-managed virtual environments and swapping GPT-4o for Claude Sonnet 4.6 via `langchain_anthropic`; integrated PyResBugs as the Python RAG database; collected and prepared the 30-bug BugsInPy evaluation subset across 7 open-source projects (`luigi`, `fastapi`, `scrapy`, `black`, `sanic`, `spacy`, `PySnooper`); and ran the end-to-end Python repair experiments yielding the 87% plausible repair rate, with `luigi` reaching 12/12 and `fastapi` 3/5 (Sections 3.3, 5.3).

Shengyue Guo (guo79) executed the local deployment of the full pipeline with Qwen 2.5 Coder 14B (64 GB RAM, 24 GB VRAM), including managing the long-context hallucination and memory-overflow issues that arose for the original VibeRepair on Closure; designed and validated the external-tool-removal optimization that replaces Joern-based runtime analysis with buggy-code-only embedding (Section 4.5); implemented the LLM-based distillation component within the Reasoning module, compressing the 1,564-token prompt to ∼150 tokens (∼90% reduction); and ran the Java Defects4J APR evaluation comparing VibeRepair vs. VibeRepair+ across Chart, Closure, Lang, and Math (Sections 5.1–5.2, Table 4).

References

Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. 2025. RepairAgent: An autonomous, LLM-based agent for program repair. In *Proceedings of the 47th International Conference on Software Engineering (ICSE)*, pages 2188–2200.

Zimin Chen, Steve Kommrusch, Michele Tufano, Louis-Noël Pouchet, Denys Poshyvanyk, and Martin Monperrus. 2021. SequenceR: Sequence-to-sequence learning for end-to-end program repair. *IEEE Transactions on Software Engineering*, 47(9):1943–1959.

Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. 2009. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 758–759.

Domenico Cotroneo, Giovanni De Rosa, and Pietro Liguori. 2025. PyResBugs: A dataset of residual python bugs for natural language-driven fault injection. Preprint.

René Just, Darioush Jalali, and Michael D. Ernst. 2014. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA)*, pages 437–440.

Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. 2012. GenProg: A generic method for automatic software repair. *IEEE Transactions on Software Engineering*, 38(1):54–72.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389.

Haifeng Ruan, Yuntong Zhang, and Abhik Roychoudhury. 2025. SpecRover: Code intent extraction via LLMs. In *Proceedings of the 47th International Conference on Software Engineering (ICSE)*, pages 963–974.

Ming Wen, Junjie Chen, Rongxin Wu, Dan Hao, and Shing-Chi Cheung. 2018. Context-aware patch generation for better automated program repair. In *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, pages 1–11.

Ratnadira Widyasari, Sheng Qin Sim, Camellia Lok, Haodi Qi, Jack Phan, Qijin Tay, Constance Tan, Fiona Wee, Jodie Ethelda Tan, Yuheng Yieh, Brian Goh, Ferdian Thung, Hong Jin Kang, Thong Hoang, David Lo, and Eng Lieh Ouh. 2020. BugsInPy: A database of existing bugs in Python programs to enable controlled testing and debugging studies. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*.

Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated program repair in the era of large pre-trained language models. In *Proceedings of the 45th International Conference on Software Engineering (ICSE)*, pages 1482–1494.

Chunqiu Steven Xia and Lingming Zhang. 2024. Keep the conversation going: Fixing 162 out of 337 bugs for $0.42 each using ChatGPT. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pages 819–831.

Xin Yin, Chao Ni, Shaohua Wang, Zhenhao Li, Limin Zeng, and Xiaohu Yang. 2024. ThinkRepair: Self-directed automated program repair. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pages 1274–1286.

Jiahong Zhang, Kaibo Huang, Junjie Zhang, Yepang Liu, and Chao Chen. 2025. Repair ingredients are all you need: Improving large language model-based program repair via repair ingredients search. Preprint.

Tianyu Zhu, Lucas C. Cordeiro, Mustafa A. Mustafa, and Youcheng Sun. 2026. Specification vibing for automated program repair. Preprint.