# DYSCOR: STAGE-CONSTRAINED DYNAMIC SPARSE CONTEXT ROUTING FOR REPOSITORY-LEVEL CODE REPAIR

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Agentic code repair systems increasingly combine large language models with retrieval, tools, and structured context packs, yet most existing approaches expose similar repository evidence throughout task understanding, localization, patch generation, and feedback revision. This stage-agnostic context construction wastes token budget and can distract repair agents with evidence that is useful only in specific phases. We propose DYSCOR, a stage-constrained dynamic sparse context routing framework for repository-level code repair. DYSCOR decomposes repository information into fine-grained evidence units, assigns stage-aware visibility states, routes compact evidence under token budgets, and updates evidence confidence and visibility after validation using test outcomes, traceback changes, and patch diffs. Experiments on SWE-Bench Lite, SWE-Bench Verified, and RepoRepair-Hard show that DYSCOR consistently improves repair effectiveness, localization quality, and context efficiency. It achieves an average resolve rate of 33.8%, outperforming ToolAgent+StaticPack, StaticPack, and Hybrid-RAG by 4.5, 7.7, and 10.0 absolute points, respectively. DYSCOR also reaches 54.8% file-level Hit@1, 72.4% file-level Hit@5, 34.2% function-level Hit@1, and 55.8% function-level Hit@5, while reducing average input tokens to 58.4K, a 43.5% reduction over ToolAgent+StaticPack, and lowering redundant context from 36.4% to 18.5%.

## 1 INTRODUCTION

Large language models (LLMs), from the Transformer (Vaswani et al., 2017), BERT (Devlin et al., 2019), and T5 (Raffel et al., 2020) to GPT-style few-shot models (Brown et al., 2020) and code-specialized models such as Codex (Chen et al., 2021a) and Code Llama (Rozière et al., 2023), have made repository-level code repair increasingly agentic. Modern repair agents read issue reports, inspect repositories, invoke tools, edit files, and validate patches, often combining chain-of-thought reasoning (Wei et al., 2022) with systems such as SWE-agent (Yang et al., 2024b;a), Agentless (Xia et al., 2024), and AutoCodeRover (Zhang et al., 2024). This setting requires reasoning over issue intent, failing tests, traces, cross-file dependencies, interfaces, and project conventions rather than a single bounded function.

Context construction remains a key bottleneck. Since an entire repository rarely fits in the prompt, existing systems retrieve snippets using BM25 (Robertson & Zaragoza, 2009), dense passage retrieval (Karpukhin et al., 2020), ColBERT-style late interaction (Khattab & Zaharia, 2020), or retrieval-augmented generation (Lewis et al., 2020), and then build context packs containing issue text, candidate files, related symbols, traces, tests, and code regions. These packs are usually static: once collected, similar evidence is reused during task understanding, localization, patch generation, and revision.

This static treatment is misaligned with staged repair. Localization benefits from candidate files, symbol references, call relations, and traceback-aligned regions, much like fault-localization signals such as Tarantula (Jones & Harrold, 2005) and similarity-coefficient localization (Abreu et al., 2006). Patch generation instead needs precise local code, interfaces, and tests, while feedback revision needs patch diffs and new validation signals. Exposing all evidence at every stage wastes budget and can distract the agent. A better context manager should preserve shared task information while making



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

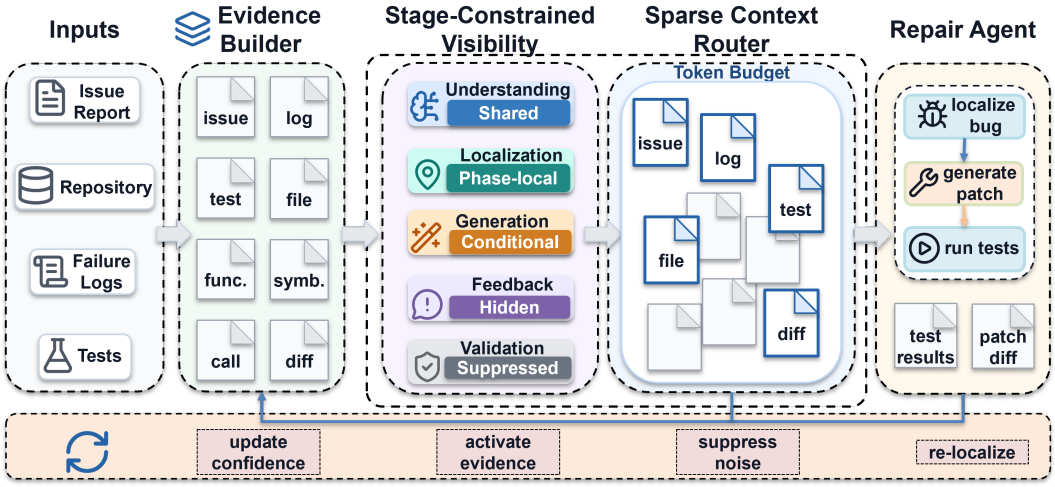



Figure 1: Overview of the DySCoR workflow.

phase-local and feedback-activated evidence visible only when useful, following the broader spirit of ReAct (Yao et al., 2023; 2022), Toolformer (Schick et al., 2023), and Reflexion (Shinn et al., 2023).

We propose **DySCoR**, a **Dy**namic **S**parse **Co**ntext **R**outing framework for agentic repository-level code repair. DySCoR decomposes repository information into fine-grained evidence units, assigns each unit a stage-aware visibility state, and routes a compact subset of visible evidence under token budgets. After validation, DySCoR updates evidence confidence and visibility using test outcomes, traceback changes, and patch diffs, enabling context expansion, suppression, and re-localization. Figure 1 gives an overview of this process: DySCoR starts from issue reports, repository files, failure logs, and tests; converts them into typed evidence units; controls their stage-constrained visibility; routes compact context under token budgets; and updates confidence, activation, suppression, and re-localization decisions after validation feedback.

We evaluate DySCoR on repository-level repair benchmarks, including SWE-bench (Jimenez et al., 2023), against retrieval-based repair, static context packing, and agent-driven search baselines. Results show that DySCoR improves repair effectiveness, localization quality, and context efficiency. Our contributions are: (1) formulating stage-agnostic context exposure as a limitation of repository-level repair agents; (2) introducing stage-constrained evidence visibility over shared, phase-local, conditional, hidden, and suppressed states; (3) designing sparse context routing with feedback-driven updates; and (4) demonstrating consistent gains across repair, localization, and token-efficiency metrics.

## 2 Related Work

### 2.1 Repository-Level Code Repair

**Program Repair.** Automated program repair spans genetic programming (Weimer et al., 2009; Le Goues et al., 2012), semantic and constraint-based repair such as SemFix (Nguyen et al., 2013), DirectFix (Mechtaev et al., 2015), and Angelix (Mechtaev et al., 2016), learned patch priors such as Prophet (Long & Rinard, 2016), template repair such as TBar (Liu et al., 2019), bytecode mutation such as PraPR (Ghanbari & Zhang, 2019), and broader surveys (Gazzola et al., 2019). Neural repair recasts patching as sequence prediction or code editing, including Tufano et al. (Tufano et al., 2019), SequenceR (Chen et al., 2021b), CoCoNuT (Lutellier et al., 2020), CURE (Jiang et al., 2021), DeepFix (Gupta et al., 2017), CODIT (Chakraborty et al., 2022), and Getafix (Bader et al., 2019). Many of these settings bound context to a function or file, whereas repository-level benchmarks such as Defects4J (Just et al., 2014), Bugs.jar (Saha et al., 2018), ManySStuBs4J (Karampatsis & Sutton, 2020), and SWE-bench (Jimenez et al., 2023) require finding sparse repair evidence across large projects.

**Retrieval-Based and Agentic Repair.** Retrieval-augmented repair narrows the repository search space with lexical, dense, or hybrid retrieval: BM25 (Robertson & Zaragoza, 2009), REALM (Guu et al., 2020), RAG (Lewis et al., 2020), DPR (Karpukhin et al., 2020), ColBERT (Khattab & Zaharia, 2020), Fusion-in-Decoder (Izacard & Grave, 2021), FAISS-backed search (Johnson et al., 2017), and code search resources such as CodeSearchNet (Husain et al., 2019). Such methods usually rank evidence once, although localization, generation, and revision need different context. Agentic systems such as SWE-agent (Yang et al., 2024b;a), Agentless (Xia et al., 2024), and AutoCodeRover (Zhang et al., 2024) can search, read, edit, test, and revise, but still risk over-exploration or accumulating noisy observations. DySCoR complements these agents by deciding which evidence should be shared, hidden, activated, or suppressed at each repair stage.

### 2.2 Context Management for Code Agents

**Context Packing and Long Context.** Context packs organize issue descriptions, candidate files, traces, tests, tool observations, and code snippets, drawing on the naturalness and regularity of code (Hindle et al., 2016; Allamanis et al., 2017). However, most packs are static or monotonically extended. Long-context models and compression methods, including Transformer-XL (Dai et al., 2019), Reformer (Kitaev et al., 2020), Longformer (Beltagy et al., 2020), Big Bird (Zaheer et al., 2020), FlashAttention (Dao et al., 2022), and LLMLingua (Jiang et al., 2023), reduce truncation or token cost but do not decide when each evidence unit should be visible. DySCoR is orthogonal: it can use larger or compressed contexts, but treats visibility as a stage-aware routing decision.

**Agent Memory and State.** LLM-agent work studies memory, tool use, and self-correction through ReAct (Yao et al., 2023; 2022), Reflexion (Shinn et al., 2023), Toolformer (Schick et al., 2023), and Generative Agents (Park et al., 2023). For code repair, memory must be tied to validation feedback: a failed test can indicate an incomplete edit, wrong localization, or regression. DySCoR turns generic memory into typed repair state by tracking evidence confidence and visibility across stages.

## 3 DySCoR Framework

DySCoR is a model-agnostic context manager for agentic repository-level repair. Given an issue $q$, repository $\mathcal{R}$, optional logs $l$, and a repair agent with search, read, edit, and validation tools, DySCoR builds a typed evidence pool, controls stage-level evidence visibility, routes compact context under a token budget, and updates evidence states after validation. It can be paired with code-capable models such as CodeT5 (Wang et al., 2021), UniXcoder (Guo et al., 2022), InCoder (Fried et al., 2022), and Code Llama (Rozière et al., 2023).

### 3.1 Overview

DySCoR assumes the agentic setting used by SWE-agent (Yang et al., 2024a) and AutoCodeRover (Zhang et al., 2024): the agent iteratively understands the task, localizes defects, generates patches, and revises them based on validation feedback. We model the process as:

$$\mathcal{S} = \{s_{\mathrm{u}}, s_{\mathrm{l}}, s_{\mathrm{g}}, s_{\mathrm{f}}, s_{\mathrm{v}}\}, \tag{1}$$

where the stages denote task understanding, localization, patch generation, feedback-driven revision, and final validation. The phase controller may return from $s_{\mathrm{f}}$ to $s_{\mathrm{l}}$ when feedback suggests wrong localization, or to $s_{\mathrm{g}}$ when the edit is incomplete.

The pipeline follows retrieval-and-generation systems such as RAG (Lewis et al., 2020), but makes visibility stage-dependent:

$$\begin{aligned}(q, \mathcal{R}, l) &\to \mathcal{E} \to \mathcal{V}_s \to \mathcal{C}_s \\ &\to \mathrm{Prompt}_s \to a_s \to \mathrm{Update}(\mathcal{E}, \sigma),\end{aligned} \tag{2}$$

where $\mathcal{E}$ is the evidence pool, $\mathcal{V}_s$ is the visible subset for stage $s$, $\mathcal{C}_s$ is the routed context, $a_s$ is the agent output, and $\sigma$ is the repair state. Full algorithmic details are in Appendix A.

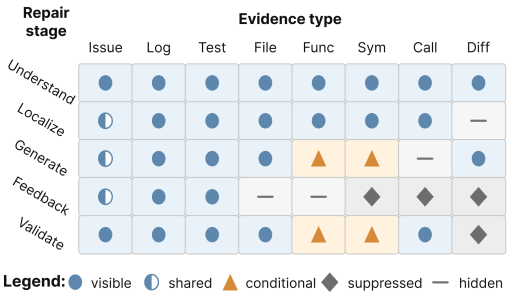



Figure 2: Stage-visibility policy.

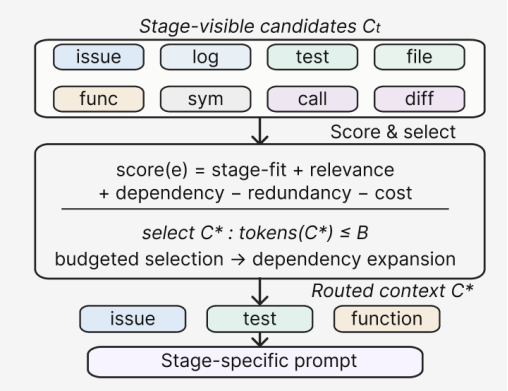



Figure 3: Budget-aware sparse context routing.

### 3.2 Evidence Units

DySCoR represents repository information as typed evidence units, reflecting the importance of local tokens, identifiers, and repository conventions in code understanding (Hindle et al., 2016; Allamanis et al., 2017):

$$e_i = \langle x_i, \tau_i, m_i, d_i, c_i, b_i, v_i \rangle. \tag{3}$$

Here $x_i$ is content, $\tau_i$ is the type, $m_i$ stores metadata such as path, symbol, line span, source, traceback position, and edit history, $d_i$ is a dependency set, $c_i$ is confidence, $b_i$ is token cost, and $v_i$ is visibility. Evidence types include issue, log, test, file, function, class, symbol, call, doc, diff, and tool output. Units are extracted from issue text, repository structure, static dependencies, and dynamic feedback using retrieval and parsing signals such as BM25 (Robertson & Zaragoza, 2009), dense retrieval (Karpukhin et al., 2020), CodeSearchNet-style code search (Husain et al., 2019), and graph-aware code representations (Feng et al., 2020; Guo et al., 2021). File-level, symbol-level, and region-level units give DySCoR both broad localization evidence and precise patch-generation context.

### 3.3 Stage-Constrained Visibility

The core design is that evidence is not equally visible to all stages. Each unit is assigned one of five visibility states:

$$v_i \in \{\text{shared}, \text{phase}, \text{conditional}, \\ \text{hidden}, \text{suppressed}\}. \tag{4}$$

Shared evidence, such as the issue summary and core failure symptom, persists across stages. Phase-local evidence, such as candidate files or local code regions, is visible only where it is useful. Conditional evidence becomes visible when feedback triggers it, hidden evidence is temporarily unavailable, and suppressed evidence is penalized after repeated contradiction. The default policy exposes broad structural context during localization, precise local code and tests during patch generation, and patch diffs plus validation output during feedback revision. Figure 2 illustrates this policy: evidence units enter from the pool on the left, the stage-by-evidence matrix determines which issue, log, test, file, function, symbol, call, and diff units are visible for the current repair stage, and activation rules reveal conditional dependencies or suppress misleading candidates after validation feedback. Appendix A.2 gives the full matrix and trigger rules, including feedback behaviors inspired by Reflexion (Shinn et al., 2023).

### 3.4 Sparse Routing and Feedback Update

For stage $s$, DySCoR selects a context $\mathcal{C}_s$ from visible evidence:

$$\mathcal{C}_s = \arg\max_{\mathcal{C} \subseteq \mathcal{V}_s} \sum_{e_i \in \mathcal{C}} R(e_i, s, \sigma) \quad \text{s.t.} \quad \sum_{e_i \in \mathcal{C}} b_i \leq B_s. \tag{5}$$

Figure 3 shows the routing step in detail. DySCoR starts from visible evidence candidates and scores them by issue relevance, stage fit, dependency utility, validation-feedback alignment, redundancy, and token cost; the router then selects a compact subset within the stage token budget, expands required dependencies, and compiles the resulting units into a stage-specific prompt. The router greedily selects evidence by score-to-cost ratio and then considers required dependencies. The score combines issue relevance, stage compatibility, confidence, dependency utility, validation-feedback alignment, redundancy, and token cost. DySCoR also reserves separate budget portions for shared task information, phase-local evidence, and repair-state feedback; the exact score, default weights, and budget ratios are listed in Appendix A.3.

After validation, DySCoR adds patch diffs and test feedback as new evidence, updates confidence, and changes visibility. If a failure overlaps the edited region and fewer tests fail, DySCoR preserves the localization and expands nearby dependencies. If the failure remains unchanged or points elsewhere, it suppresses the previous candidate and reopens alternatives. If the patch introduces unrelated regressions, DySCoR activates patch diffs and interface constraints to support conservative revision. The full confidence update rule, prompt compilation, tool policy, fallback relaxation, and complexity analysis appear in Appendix A.4.

## 4 Experiment

We evaluate DySCoR under an agentic repository-level code repair setting, following the repository-scale evaluation style introduced by SWE-bench (Jimenez et al., 2023). The experiments are designed to answer four questions. First, does DySCoR improve end-to-end repair performance compared with retrieval-based repair, static context packing, and agent-driven search? Second, does stage-constrained routing improve file-level and function-level localization? Third, does DySCoR reduce redundant context consumption under limited token budgets? Fourth, which components of DySCoR contribute most to the final performance?

### 4.1 Experimental Setup

**Benchmarks.** We evaluate on three repository-level repair benchmarks. The first benchmark is SWE-bench Lite (Jimenez et al., 2023), which contains 300 real-world GitHub issues and is widely used for efficient evaluation of repository-level repair agents. The second benchmark is SWE-bench Verified (Jimenez et al., 2023), which contains 500 human-validated instances with more reliable issue-to-test correspondence. The third benchmark is RepoRepair-Hard, a difficult subset constructed from repository-level repair tasks where the gold patch modifies multiple files, the initial retrieval result does not contain the gold file in the top-5 candidates, or the failure trace does not directly point to the edited function. Table 1 summarizes the benchmark statistics. SWE-bench Lite contains 300 instances from 11 repositories, with 1.42 gold modified files per instance on average. SWE-bench Verified contains 500 instances and has a slightly lower average number of gold modified files, 1.36, while RepoRepair-Hard contains 180 instances and is more context-demanding, with 1.88 gold modified files and 2.76 patch hunks per instance on average.

**Baselines.** We compare DySCoR with retrieval, context-packing, agentic, and recent issue-resolution baselines. **BM25-RAG** retrieves files and functions using BM25 (Robertson & Zaragoza, 2009) sparse lexical retrieval over issue descriptions, failure logs, file paths, identifiers, and docstrings. **Dense-RAG** retrieves code snippets using dense embedding similarity inspired by DPR (Karpukhin et al., 2020). **Hybrid-RAG** combines BM25 (Robertson & Zaragoza, 2009), dense retrieval (Karpukhin et al., 2020), path matching, and traceback matching, and then reranks the top candidates. **StaticPack** uses the same evidence construction module as DySCoR but packs

Table 1: Benchmark statistics.

| Benchmark | Inst. | Repos. | Avg. Files | Avg. Gold Files | Avg. Patch Hunks |
|---|---|---|---|---|---|
| SWE-bench Lite | 300 | 11 | 1,481 | 1.42 | 2.13 |
| SWE-bench Verified | 500 | 12 | 1,536 | 1.36 | 2.01 |
| RepoRepair-Hard | 180 | 17 | 2,913 | 1.88 | 2.76 |

the selected evidence into a fixed context that is reused across all repair stages. **ToolAgent** is a ReAct-style agent (Yao et al., 2023) that can search, read, edit, and test the repository without a pre-built context pack. **ToolAgent+StaticPack** initializes the agent with a static context pack and then allows free tool use. We further include representative recent repair agents and issue-resolution systems: **SWE-Fixer** (Xie et al., 2025), **SWE-agent** (Yang et al., 2024a), **AutoCodeRover** (Zhang et al., 2024), **CodeR** (Chen et al., 2024), **SWE-Search** (Antoniades et al., 2024), **MASAI** (Arora et al., 2024), and **Agentless** (Xia et al., 2024). DySCoR uses the same tool interface and the same initial evidence sources as StaticPack and ToolAgent+StaticPack, but controls evidence visibility and routing according to repair stage and validation feedback.

**Evaluation Metrics.** We use three groups of metrics. For repair effectiveness, we report **Resolve Rate**, the percentage of instances whose generated patch passes the required validation tests, and **Reproduction Pass Rate**, the percentage of instances where the patch passes the failure-reproducing tests. For localization quality, we report file-level Hit@$k$ and function-level Hit@$k$, where a prediction is correct if at least one gold modified file or function is included in the top-$k$ localized candidates. We also report **Gold Context Hit**, the percentage of instances where the final routed context contains at least one gold modified region or its direct dependency. For context efficiency, we report average input tokens over all repair rounds, peak prompt tokens, useful context ratio, redundant context ratio, average repair rounds, and average tool calls. Useful context is defined as evidence overlapping gold files, gold functions, failing tests, traceback frames, or direct structural dependencies. Redundant context is defined as evidence whose line span, semantic embedding, or token-level content has high overlap with already routed evidence.

**Implementation Details.** All methods use Qwen2.5-Coder-32B-Instruct as the base LLM, with the same model checkpoint, decoding configuration, repair timeout, tool interface, and validation harness to ensure a controlled comparison. Unless otherwise specified, the per-round prompt budget is 32K tokens, the maximum number of repair rounds is 5, and the decoding temperature is set to 0. The initial retriever retains at most 50 file candidates and 200 symbol candidates per instance. DySCoR uses the rule-based visibility policy described in Section 3.3, the routing score in Equation 9, and the feedback update rule in Equation 15. The default local context window includes 20 lines before and after the selected symbol. We run each method three times under the same configuration and report the average results. For statistical significance testing, we use paired bootstrap resampling with 10,000 samples over benchmark instances.

### 4.2 Repair Results

Table 2 reports repair results; Appendix B provides the full localization and context-efficiency results in Tables 6 and 7. DySCoR achieves the best average resolve rate of 33.8%, outperforming ToolAgent+StaticPack by 4.5 absolute points, StaticPack by 7.7 points, and Hybrid-RAG by 10.0 points. On SWE-bench Lite (Jimenez et al., 2023), DySCoR reaches 35.3% resolve rate, compared with 31.0% for ToolAgent+StaticPack and 27.7% for StaticPack. On SWE-bench Verified (Jimenez et al., 2023), DySCoR obtains 37.8% resolve rate, which is 4.8 points higher than ToolAgent+StaticPack. On RepoRepair-Hard, the gap becomes larger: DySCoR reaches 28.3%, while ToolAgent+StaticPack reaches 23.9% and StaticPack reaches 20.6%. This suggests that stage-constrained dynamic routing is especially useful when the repair requires multi-file reasoning or when the initial retrieval result is incomplete.

Figure 4 visualizes the average resolve rate across the three benchmarks. The average resolve rate increases from 20.6% for BM25-RAG to 23.8% for Hybrid-RAG, showing that stronger retrieval

Table 2: Repair performance. Superscripts on DySCoR report absolute differences from ToolAgent+StaticPack; green indicates a favorable change.

| Method | Lite | | Verified | | Hard | | Avg. |
|---|---|---|---|---|---|---|---|
| | Res.↑ | Rep.↑ | Res.↑ | Rep.↑ | Res.↑ | Rep.↑ | Res.↑ |
| BM25-RAG | 21.7 | 40.0 | 24.0 | 42.2 | 16.1 | 31.7 | 20.6 |
| Dense-RAG | 22.3 | 41.7 | 24.6 | 43.0 | 16.7 | 33.3 | 21.2 |
| Hybrid-RAG | 25.0 | 44.3 | 27.4 | 46.0 | 18.9 | 36.1 | 23.8 |
| StaticPack | 27.7 | 47.0 | 30.0 | 49.8 | 20.6 | 39.4 | 26.1 |
| ToolAgent | 29.0 | 49.3 | 31.2 | 51.6 | 22.2 | 42.2 | 27.5 |
| ToolAgent+Sta. | 31.0 | 52.0 | 33.0 | 54.0 | 23.9 | 44.4 | 29.3 |
| SWE-Fixer | 24.8 | 44.8 | 26.5 | 46.7 | 18.7 | 35.9 | 23.3 |
| SWE-agent | 25.7 | 46.0 | 27.9 | 48.1 | 19.6 | 37.4 | 24.4 |
| AutoCodeRover | 27.0 | 47.8 | 29.1 | 50.2 | 21.2 | 39.0 | 25.8 |
| CodeR | 28.2 | 49.0 | 30.6 | 51.8 | 22.0 | 40.6 | 26.9 |
| SWE-Search | 29.3 | 50.3 | 31.7 | 53.1 | 23.1 | 42.0 | 28.0 |
| MASAI | 30.1 | 51.0 | 32.3 | 53.7 | 23.6 | 43.2 | 28.7 |
| Agentless | 30.6 | 51.5 | 32.8 | 54.2 | 23.8 | 43.8 | 29.1 |
| **DySCoR** | **35.3**$^{+4.3}$ | **57.7**$^{+5.7}$ | **37.8**$^{+4.8}$ | **59.0**$^{+5.0}$ | **28.3**$^{+4.4}$ | **50.0**$^{+5.6}$ | **33.8**$^{+4.5}$ |

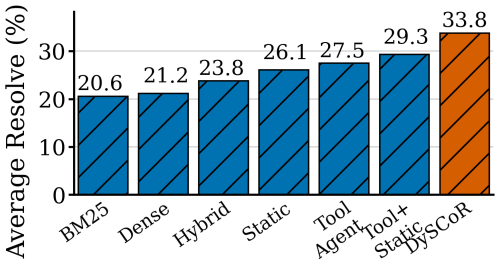



Figure 4: Average repository repair performance.

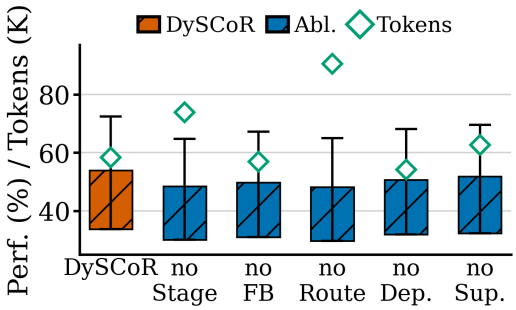



Figure 5: Ablation performance and token cost.

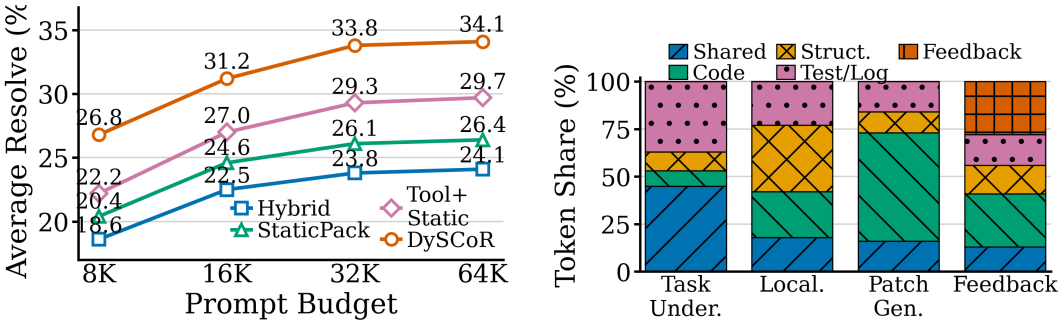
improves repair. StaticPack further improves the score to 26.1%, indicating that structured context helps beyond raw retrieval. ToolAgent+StaticPack reaches 29.3%, but DySCoR achieves 33.8%, showing that the main gain comes from controlling when and how evidence is exposed rather than simply giving the agent more context or more tools.

Table 6 reports localization quality, following the long-standing use of localization metrics in debugging and repair studies such as Tarantula (Jones & Harrold, 2005) and similarity-coefficient fault localization (Abreu et al., 2006). DySCoR achieves 54.8% file-level Hit@1 and 72.4% file-level Hit@5, outperforming ToolAgent+StaticPack by 6.9 and 6.7 points, respectively. At the function level, DySCoR achieves 34.2% Hit@1 and 55.8% Hit@5, which are 7.1 and 7.8 points higher than ToolAgent+StaticPack. The Gold Context Hit rate also increases from 59.3% to 66.1%. These results show that DySCoR does not merely improve prompt efficiency; it also routes more repair-relevant evidence to the agent.

For context efficiency, Table 7 shows that DySCoR uses 58.4K input tokens per instance on average, while StaticPack uses 84.6K and ToolAgent+StaticPack uses 103.4K. This corresponds to a 31.0% token reduction compared with StaticPack and a 43.5% reduction compared with ToolAgent+StaticPack. DySCoR also reduces the redundant context ratio from 36.4% to 18.5% compared with ToolAgent+StaticPack, while increasing the useful context ratio from 32.5% to 42.6%. The average number of repair rounds decreases from 3.9 to 3.1, suggesting that better context routing helps the agent converge faster.

Table 3: Ablation results.

| Variant | Resolve | File H@5 | Func. H@5 | Input Tok. | Redun. Ctx. |
|---|---|---|---|---|---|
| **DySCoR** | **33.8** | **72.4** | **55.8** | **58.4K** | **18.5** |
| w/o Stage Constraints | 30.1 | 64.7 | 50.6 | 73.9K | 29.8 |
| w/o Dynamic Feedback | 31.0 | 67.2 | 51.3 | 56.9K | 20.1 |
| w/o Sparse Routing | 29.8 | 65.0 | 49.9 | 90.5K | 38.7 |
| w/o Dependency Expansion | 32.0 | 68.1 | 51.9 | 54.2K | 19.4 |
| w/o Suppression | 32.4 | 69.6 | 53.7 | 62.7K | 24.6 |




(a) Budget robustness.

(b) context distribution.

Figure 6: Budget robustness and context distribution.

### 4.3 Ablation and Robustness

Table 3 reports ablation results. Removing sparse routing causes the largest performance drop, reducing average resolve rate from 33.8% to 29.8%, while increasing input tokens from 58.4K to 90.5K. Removing stage constraints reduces resolve rate to 30.1% and increases redundant context from 18.5% to 29.8%, showing that stage-level visibility is important for preventing irrelevant evidence from entering patch generation and feedback revision. Removing dynamic feedback reduces resolve rate to 31.0%, indicating that validation-driven context update contributes 2.8 points. Removing dependency expansion mainly hurts localization and function-level Hit, reducing function Hit@5 from 55.8% to 51.9%. Removing suppression has a smaller but consistent effect, decreasing resolve rate from 33.8% to 32.4%.

Figure 5 summarizes the ablation variants as performance-efficiency candles. Each wick spans Resolve, file-level Hit@5, and function-level Hit@5; each hatched body runs from Resolve to the mean of these three performance metrics; and the hollow diamond marks the input-token cost on the same axis. DySCoR has the strongest profile, with 33.8% Resolve, 72.4% file-level Hit@5, 55.8% function-level Hit@5, and 58.4K input tokens. Removing sparse routing lowers Resolve to 29.8% and raises input tokens to 90.5K, while removing stage constraints lowers Resolve to 30.1% and raises redundant context to 29.8%.

Figure 6a shows robustness under different context budgets. With an 8K prompt budget, DySCoR achieves 26.8% average resolve rate, outperforming ToolAgent+StaticPack by 4.6 points and StaticPack by 6.4 points. At 16K, DySCoR improves to 31.2%, while ToolAgent+StaticPack reaches 27.0%. At 32K, DySCoR reaches 33.8%, and increasing the budget to 64K only slightly improves it to 34.1%. This saturation is consistent with recent evidence that larger context or more data is not a substitute for targeted evidence selection, even for data-constrained language models (Muennighoff et al., 2023).

Table 4 reports performance under different maximum repair rounds. When only one round is allowed, DySCoR achieves 24.5% resolve rate, nearly matching StaticPack with three rounds. With three rounds, DySCoR reaches 31.9%, and with five rounds it reaches 33.8%. Increasing the limit to seven

Table 4: Repair rounds.

| Method | 1 Round | 3 Rounds | 5 Rounds | 7 Rounds |
|---|---|---|---|---|
| StaticPack | 18.7 | 24.9 | 26.1 | 26.5 |
| ToolAgent+StaticPack | 20.3 | 27.8 | 29.3 | 29.8 |
| **DySCoR** | **24.5** | **31.9** | **33.8** | **34.3** |

rounds brings only a small improvement to 34.3%. The diminishing gain after five rounds indicates that most successful repairs occur within a small number of stage transitions.

Additional analyses of stage-wise context, feedback actions, failure categories, and representative cases are provided in Appendix B.1.

**Statistical Significance.** Across all benchmarks, DySCoR improves average resolve rate over ToolAgent+StaticPack by 4.5 absolute points. Paired bootstrap testing shows that this improvement is statistically significant with $p < 0.01$. The 95% confidence interval of DySCoR's average resolve rate is $[32.1, 35.4]$, while that of ToolAgent+StaticPack is $[27.8, 30.9]$. The intervals do not overlap substantially, supporting the conclusion that stage-constrained dynamic routing provides a robust improvement rather than a variance artifact.

The experimental results support three conclusions. First, DySCoR improves end-to-end repair performance across standard, verified, and hard repository-level repair tasks. Second, DySCoR improves localization by exposing structural evidence during localization and suppressing low-value evidence during patch generation. Third, DySCoR substantially reduces redundant context consumption, indicating that effective code agents benefit not only from more context, but from stage-aware control over context visibility.

## 5 Conclusion

We presented **DySCoR**, a stage-constrained dynamic sparse context routing framework for agentic repository-level code repair. DySCoR treats repository evidence as typed, stateful units rather than a fixed context pack, allowing task understanding, localization, patch generation, and feedback revision to receive different evidence according to their roles. It also updates routing states after validation, so test outcomes, traceback changes, and patch diffs can guide later context selection. Across repository-level repair benchmarks, DySCoR achieves a 33.8% average resolve rate, improving over ToolAgent+StaticPack, StaticPack, and Hybrid-RAG by 4.5, 7.7, and 10.0 absolute points. It also reaches 72.4% file-level Hit@5 and 55.8% function-level Hit@5, while lowering average input tokens to 58.4K and redundant context to 18.5%. These results show that stage-aware evidence control improves repair accuracy, localization, and efficiency. They also indicate that context management is a central design dimension for reliable repair agents, not only an auxiliary retrieval step. Future work will explore learned visibility policies, richer validation signals, and broader applications of stage-constrained routing.

## 6 Limitations

DySCoR improves context efficiency and repair performance, but it still depends on the quality of initial localization signals, validation feedback, and the underlying repair agent. It may fail when the root cause is deeply hidden, when tests provide weak or misleading feedback, or when the correct fix requires broad architectural changes across multiple files. Our experiments focus on SWE-bench-style Python repository repair, so future work should evaluate whether stage-constrained context routing generalizes to other programming languages, larger industrial repositories, and software-engineering tasks beyond bug fixing.

## AI USE STATEMENT

Generative AI tools were used for language polishing and for adapting the manuscript's LaTeX formatting to the ICLR 2027 submission template. The authors are responsible for the final content of the paper.

## REFERENCES

Rui Abreu, Peter Zoeteweij, and Arjan J. C. van Gemund. An evaluation of similarity coefficients for software fault localization. In *12th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2006), 18-20 December, 2006, University of California, Riverside, USA*, pp. 39–46. IEEE Computer Society, 2006. doi: 10.1109/PRDC.2006.18. URL https://doi.org/10.1109/PRDC.2006.18.

Miltiadis Allamanis, Earl T. Barr, Premkumar T. Devanbu, and Charles Sutton. A survey of machine learning for big code and naturalness. *CoRR*, abs/1709.06182, 2017. URL http://arxiv.org/abs/1709.06182.

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Yang Wang. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement. *CoRR*, abs/2410.20285, 2024. doi: 10.48550/ARXIV.2410.20285. URL https://doi.org/10.48550/arXiv.2410.20285.

Daman Arora, Atharv Sonwane, Nalin Wadhwa, Abhav Mehrotra, Saiteja Utpala, Ramakrishna Bairi, Aditya Kanade, and Nagarajan Natarajan. MASAI: modular architecture for software-engineering AI agents. *CoRR*, abs/2406.11638, 2024. doi: 10.48550/ARXIV.2406.11638. URL https://doi.org/10.48550/arXiv.2406.11638.

Johannes Bader, Andrew Scott, Michael Pradel, and Satish Chandra. Getafix: learning to fix bugs automatically. *Proc. ACM Program. Lang.*, 3(OOPSLA):159:1–159:27, 2019. doi: 10.1145/3360585. URL https://doi.org/10.1145/3360585.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020. URL https://arxiv.org/abs/2004.05150.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Saikat Chakraborty, Yangruibo Ding, Miltiadis Allamanis, and Baishakhi Ray. CODIT: code editing with tree-based neural models. *IEEE Trans. Software Eng.*, 48(4):1385–1399, 2022. doi: 10.1109/TSE.2020.3020502. URL https://doi.org/10.1109/TSE.2020.3020502.

Dong Chen, Shaoxin Lin, Muhan Zeng, Daoguang Zan, Jian-Gang Wang, Anton Cheshkov, Jun Sun, Hao Yu, Guoliang Dong, Artem Aliev, Jie Wang, Xiao Cheng, Guangtai Liang, Yuchi Ma, Pan Bian, Tao Xie, and Qianxiang Wang. Coder: Issue resolving with multi-agent and task graphs. *CoRR*, abs/2406.01304, 2024. doi: 10.48550/ARXIV.2406.01304. URL https://doi.org/10.48550/arXiv.2406.01304.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian,

Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021a. URL https://arxiv.org/abs/2107.03374.

Zimin Chen, Steve Kommrusch, Michele Tufano, Louis-Noël Pouchet, Denys Poshyvanyk, and Martin Monperrus. Sequencer: Sequence-to-sequence learning for end-to-end program repair. *IEEE Trans. Software Eng.*, 47(9):1943–1959, 2021b. doi: 10.1109/TSE.2019.2940179. URL https://doi.org/10.1109/TSE.2019.2940179.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc Viet Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In Anna Korhonen, David R. Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 2978–2988. Association for Computational Linguistics, 2019. doi: 10.18653/V1/P19-1285. URL https://doi.org/10.18653/v1/p19-1285.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/V1/N19-1423. URL https://doi.org/10.18653/v1/n19-1423.

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, Online Event, 16-20 November 2020*, Findings of ACL, pp. 1536–1547. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.FINDINGS-EMNLP.139. URL https://doi.org/10.18653/v1/2020.findings-emnlp.139.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. Incoder: A generative model for code infilling and synthesis. *CoRR*, abs/2204.05999, 2022. doi: 10.48550/ARXIV.2204.05999. URL https://doi.org/10.48550/arXiv.2204.05999.

Luca Gazzola, Daniela Micucci, and Leonardo Mariani. Automatic software repair: A survey. *IEEE Trans. Software Eng.*, 45(1):34–67, 2019. doi: 10.1109/TSE.2017.2755013. URL https://doi.org/10.1109/TSE.2017.2755013.

Ali Ghanbari and Lingming Zhang. Prapr: Practical program repair via bytecode mutation. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*, pp. 1118–1121. IEEE, 2019. doi: 10.1109/ASE.2019.00116. URL https://doi.org/10.1109/ASE.2019.00116.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin B. Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. Graphcodebert: Pre-training

code representations with data flow. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=jLoC4ez43PZ.

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), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pp. 7212–7225. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.ACL-LONG.499. URL https://doi.org/10.18653/v1/2022.acl-long.499.

Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish K. Shevade. Deepfix: Fixing common C language errors by deep learning. In Satinder Singh and Shaul Markovitch (eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pp. 1345–1351. AAAI Press, 2017. doi: 10.1609/AAAI.V31I1.10742. URL https://doi.org/10.1609/aaai.v31i1.10742.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: retrieval-augmented language model pre-training. *CoRR*, abs/2002.08909, 2020. URL https://arxiv.org/abs/2002.08909.

Abram Hindle, Earl T. Barr, Mark Gabel, Zhendong Su, and Premkumar T. Devanbu. On the naturalness of software. *Commun. ACM*, 59(5):122–131, 2016. doi: 10.1145/2902362. URL https://doi.org/10.1145/2902362.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search. *CoRR*, abs/1909.09436, 2019. URL http://arxiv.org/abs/1909.09436.

Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In Paola Merlo, Jörg Tiedemann, and Reut Tsarfaty (eds.), *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, pp. 874–880. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.EACL-MAIN.74. URL https://doi.org/10.18653/v1/2021.eacl-main.74.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 13358–13376. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.825. URL https://doi.org/10.18653/v1/2023.emnlp-main.825.

Nan Jiang, Thibaud Lutellier, and Lin Tan. CURE: code-aware neural machine translation for automatic program repair. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*, pp. 1161–1173. IEEE, 2021. doi: 10.1109/ICSE43902.2021.00107. URL https://doi.org/10.1109/ICSE43902.2021.00107.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *CoRR*, abs/2310.06770, 2023. doi: 10.48550/ARXIV.2310.06770. URL https://doi.org/10.48550/arXiv.2310.06770.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *CoRR*, abs/1702.08734, 2017. URL http://arxiv.org/abs/1702.08734.

James A. Jones and Mary Jean Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In David F. Redmiles, Thomas Ellman, and Andrea Zisman (eds.), *20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), November 7-11, 2005, Long Beach, CA, USA*, pp. 273–282. ACM, 2005. doi: 10.1145/1101908.1101949. URL https://doi.org/10.1145/1101908.1101949.

René Just, Darioush Jalali, and Michael D. Ernst. Defects4j: a database of existing faults to enable controlled testing studies for java programs. In Corina S. Pasareanu and Darko Marinov (eds.), *International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014*, pp. 437–440. ACM, 2014. doi: 10.1145/2610384.2628055. URL https://doi.org/10.1145/2610384.2628055.

Rafael-Michael Karampatsis and Charles Sutton. How often do single-statement bugs occur?: The manysstubs4j dataset. In Sunghun Kim, Georgios Gousios, Sarah Nadi, and Joseph Hejderup (eds.), *MSR '20: 17th International Conference on Mining Software Repositories, Seoul, Republic of Korea, 29-30 June, 2020*, pp. 573–577. ACM, 2020. doi: 10.1145/3379597.3387491. URL https://doi.org/10.1145/3379597.3387491.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pp. 6769–6781. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.EMNLP-MAIN.550. URL https://doi.org/10.18653/v1/2020.emnlp-main.550.

Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over BERT. In Jimmy X. Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu (eds.), *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pp. 39–48. ACM, 2020. doi: 10.1145/3397271.3401075. URL https://doi.org/10.1145/3397271.3401075.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=rkgNKkHtvB.

Claire Le Goues, Michael Dewey-Vogt, Stephanie Forrest, and Westley Weimer. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In Martin Glinz, Gail C. Murphy, and Mauro Pezzè (eds.), *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, pp. 3–13. IEEE Computer Society, 2012. doi: 10.1109/ICSE.2012.6227211. URL https://doi.org/10.1109/ICSE.2012.6227211.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Kui Liu, Anil Koyuncu, Dongsun Kim, and Tegawendé F. Bissyandé. Tbar: revisiting template-based automated program repair. In Dongmei Zhang and Anders Møller (eds.), *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019*, pp. 31–42. ACM, 2019. doi: 10.1145/3293882.3330577. URL https://doi.org/10.1145/3293882.3330577.

Fan Long and Martin C. Rinard. Automatic patch generation by learning correct code. In Rastislav Bodík and Rupak Majumdar (eds.), *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pp. 298–312. ACM, 2016. doi: 10.1145/2837614.2837617. URL https://doi.org/10.1145/2837614.2837617.

Thibaud Lutellier, Hung Viet Pham, Lawrence Pang, Yitong Li, Moshi Wei, and Lin Tan. Coconut: combining context-aware neural translation models using ensemble for program repair. In Sarfraz Khurshid and Corina S. Pasareanu (eds.), *ISSTA '20: 29th ACM SIGSOFT International Symposium*

*on Software Testing and Analysis, Virtual Event, USA, July 18-22, 2020*, pp. 101–114. ACM, 2020. doi: 10.1145/3395363.3397369. URL https://doi.org/10.1145/3395363.3397369.

Sergey Mechtaev, Jooyong Yi, and Abhik Roychoudhury. Directfix: Looking for simple program repairs. In Antonia Bertolino, Gerardo Canfora, and Sebastian G. Elbaum (eds.), *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, pp. 448–458. IEEE Computer Society, 2015. doi: 10.1109/ICSE.2015.63. URL https://doi.org/10.1109/ICSE.2015.63.

Sergey Mechtaev, Jooyong Yi, and Abhik Roychoudhury. Angelix: scalable multiline program patch synthesis via symbolic analysis. In Laura K. Dillon, Willem Visser, and Laurie A. Williams (eds.), *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, pp. 691–701. ACM, 2016. doi: 10.1145/2884781.2884807. URL https://doi.org/10.1145/2884781.2884807.

Niklas Muennighoff, Alexander M. Rush, Boaz Barak, Teven Le Scao, Aleksandra Piktus, Nouamane Tazi, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling data-constrained language models. *CoRR*, abs/2305.16264, 2023. doi: 10.48550/ARXIV.2305.16264. URL https://doi.org/10.48550/arXiv.2305.16264.

Hoang Duong Thien Nguyen, Dawei Qi, Abhik Roychoudhury, and Satish Chandra. Semfix: program repair via semantic analysis. In David Notkin, Betty H. C. Cheng, and Klaus Pohl (eds.), *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pp. 772–781. IEEE Computer Society, 2013. doi: 10.1109/ICSE.2013.6606623. URL https://doi.org/10.1109/ICSE.2013.6606623.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. *CoRR*, abs/2304.03442, 2023. doi: 10.48550/ARXIV.2304.03442. URL https://doi.org/10.48550/arXiv.2304.03442.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020. URL https://jmlr.org/papers/v21/20-074.html.

Stephen E. Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, 2009. doi: 10.1561/1500000019. URL https://doi.org/10.1561/1500000019.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950, 2023. doi: 10.48550/ARXIV.2308.12950. URL https://doi.org/10.48550/arXiv.2308.12950.

Ripon K. Saha, Yingjun Lyu, Wing Lam, Hiroaki Yoshida, and Mukul R. Prasad. Bugs.jar: a large-scale, diverse dataset of real-world java bugs. In Andy Zaidman, Yasutaka Kamei, and Emily Hill (eds.), *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, pp. 10–13. ACM, 2018. doi: 10.1145/3196398.3196473. URL https://doi.org/10.1145/3196398.3196473.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *CoRR*, abs/2302.04761, 2023. doi: 10.48550/ARXIV.2302.04761. URL https://doi.org/10.48550/arXiv.2302.04761.

Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *CoRR*, abs/2303.11366, 2023. doi: 10.48550/ARXIV.2303.11366. URL https://doi.org/10.48550/arXiv.2303.11366.

Michele Tufano, Jevgenija Pantiuchina, Cody Watson, Gabriele Bavota, and Denys Poshyvanyk. On learning meaningful code changes via neural machine translation. In Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (eds.), *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pp. 25–36. IEEE / ACM, 2019. doi: 10.1109/ICSE.2019.00021. URL https://doi.org/10.1109/ICSE.2019.00021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Yue Wang, Weishi Wang, Shafiq R. 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, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pp. 8696–8708. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.EMNLP-MAIN.685. URL https://doi.org/10.18653/v1/2021.emnlp-main.685.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. Automatically finding patches using genetic programming. In *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*, pp. 364–374. IEEE, 2009. doi: 10.1109/ICSE.2009.5070536. URL https://doi.org/10.1109/ICSE.2009.5070536.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents. *CoRR*, abs/2407.01489, 2024. doi: 10.48550/ARXIV.2407.01489. URL https://doi.org/10.48550/arXiv.2407.01489.

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. *CoRR*, abs/2501.05040, 2025. doi: 10.48550/ARXIV.2501.05040. URL https://doi.org/10.48550/arXiv.2501.05040.

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 Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024a. URL http://papers.nips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

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. *CoRR*, abs/2405.15793, 2024b. doi: 10.48550/ARXIV.2405.15793. URL https://doi.org/10.48550/arXiv.2405.15793.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *CoRR*, abs/2210.03629, 2022. doi: 10.48550/ARXIV.2210.03629. URL https://doi.org/10.48550/arXiv.2210.03629.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/c8512d142a2d849725f31a9a7a361ab9-Abstract.html.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. *CoRR*, abs/2404.05427, 2024. doi: 10.48550/ARXIV.2404.05427. URL https://doi.org/10.48550/arXiv.2404.05427.

# A METHOD DETAILS

## A.1 EVIDENCE CONSTRUCTION AND DEPENDENCIES

DySCoR builds evidence units from four sources. From the issue report, it extracts the task summary, constraints, mentioned files, mentioned symbols, and natural-language failure symptoms. From the repository, it extracts file summaries, imports, classes, functions, signatures, docstrings, and local code windows. From static analysis, it extracts symbol references, caller-callee relations, inheritance relations, and test-to-source links when available. From dynamic feedback, it extracts failing tests, traceback frames, assertion messages, command outputs, patch diffs, and newly introduced errors. File-level and symbol-level units are produced with tree-sitter or language-specific AST tooling, while text candidates can be produced by BM25 (Robertson & Zaragoza, 2009), dense retrieval (Karpukhin et al., 2020), or a hybrid retriever. DySCoR uses three default granularities. File-level units provide repository structure and are mainly used in task understanding and localization. Symbol-level units correspond to functions, methods, classes, and exported variables, and are mainly used in localization and patch generation. Region-level units correspond to line windows around traceback frames, selected symbols, or previous edits, and are mainly used in patch generation and feedback revision. For a selected function or method, DySCoR attaches $w = 20$ lines before and after the definition by default.

Evidence units may depend on other units. If a function body is selected, its signature, enclosing class, imports, and directly referenced definitions may be required for correct interpretation:

$$d_i = \{e_j \mid e_j \text{ is structurally required by } e_i\}. \tag{6}$$

During routing, selected units trigger dependency expansion with full or compressed dependency forms depending on the remaining budget.

## A.2 VISIBILITY DETAILS

DySCoR defines a stage-evidence visibility matrix:

$$M \in \{0, 1, \rho\}^{|\mathcal{S}|\times|\mathcal{E}|}, \tag{7}$$

where $M_{s,i} = 1$ means $e_i$ is visible in stage $s$, $M_{s,i} = 0$ means invisible, and $M_{s,i} = \rho$ means conditionally visible. Conditional visibility is resolved by the repair state $\sigma$:

$$\begin{aligned}\mathcal{V}_s = \{e_i \in \mathcal{E} \mid\ & M_{s,i} = 1 \vee \\ & (M_{s,i} = \rho \wedge g(e_i, s, \sigma) = 1)\}.\end{aligned} \tag{8}$$

Table 5: Default stage-constrained visibility policy. 1 denotes visible, 0 denotes invisible, and $\rho$ denotes conditionally visible.

| **Evidence Type** | $s_{\mathrm{u}}$ | $s_{\mathrm{l}}$ | $s_{\mathrm{g}}$ | $s_{\mathrm{f}}$ | $s_{\mathrm{v}}$ |
|---|---|---|---|---|---|
| Issue Summary | 1 | 1 | 1 | 1 | 1 |
| Failure Symptom | 1 | 1 | 1 | 1 | 1 |
| Candidate File | 0 | 1 | $\rho$ | 1 | 0 |
| Symbol Reference | 0 | 1 | $\rho$ | 1 | 0 |
| Call Relation | 0 | 1 | 0 | $\rho$ | 0 |
| Local Code Region | 0 | $\rho$ | 1 | 1 | 0 |
| Relevant Test | $\rho$ | 1 | 1 | 1 | 1 |
| Patch Diff | 0 | 0 | $\rho$ | 1 | 1 |
| Validation Output | 0 | 0 | 0 | 1 | 1 |
| Low-Confidence Candidate | 0 | $\rho$ | 0 | $\rho$ | 0 |

The condition function $g(e_i, s, \sigma)$ uses rule-based triggers. A low-confidence candidate becomes visible if the current patch fails and the traceback does not overlap with the previous edit. A call relation becomes visible in feedback revision if the failing test changes error type, indicating a missing dependency or incomplete behavioral update. A patch diff becomes visible in patch generation only after at least one candidate patch has been produced.

### A.3 Routing Details

The routing score combines semantic, structural, stage, and feedback signals:

$$\begin{aligned} R(e_i, s, \sigma) := \alpha \cdot \mathrm{Rel}(e_i, q, l) + \beta \cdot \mathrm{Stage}(e_i, s) \\ + \gamma \cdot c_i + \delta \cdot \mathrm{Dep}(e_i, \sigma) \\ + \eta \cdot \mathrm{Fb}(e_i, \sigma) - \lambda \cdot \mathrm{Red}(e_i, \mathcal{C}) \\ - \mu \cdot \mathrm{Cost}(e_i). \end{aligned} \tag{9}$$

$\mathrm{Rel}$ combines BM25, embedding similarity, identifier overlap, path overlap, and traceback matching. $\mathrm{Stage}$ measures type-stage suitability according to Table 5. $\mathrm{Dep}$ rewards evidence that connects selected candidates to imports, callers, callees, or enclosing classes. $\mathrm{Fb}$ measures alignment with validation feedback. $\mathrm{Red}$ penalizes overlap with already selected evidence, and $\mathrm{Cost}$ is normalized token cost. Unless tuned on a development set, DySCoR uses:

$$\begin{aligned} \alpha = 0.30, \quad \beta = 0.20, \quad \gamma = 0.15, \\ \delta = 0.15, \quad \eta = 0.15, \quad \lambda = 0.10, \quad \mu = 0.05. \end{aligned} \tag{10}$$

The initial confidence score is:

$$\begin{aligned} c_i^{(0)} = 0.5 \cdot \mathrm{Rel}(e_i, q, l) \\ + 0.3 \cdot \mathrm{Trace}(e_i, l) \\ + 0.2 \cdot \mathrm{Struct}(e_i, \mathcal{R}). \end{aligned} \tag{11}$$

The total prompt budget $B$ is split after accounting for the instruction template and output schema:

$$B_s = B_s^{\mathrm{shared}} + B_s^{\mathrm{phase}} + B_s^{\mathrm{state}}. \tag{12}$$

The default ratio is $2 : 6 : 2$; patch generation shifts to $2 : 7 : 1$, and feedback revision uses $2 : 5 : 3$. This budgeted design is complementary to long-context and compression methods such as Longformer (Beltagy et al., 2020), Big Bird (Zaheer et al., 2020), FlashAttention (Dao et al., 2022), and LLMLingua (Jiang et al., 2023).

Redundancy is:

$$\mathrm{Red}(e_i, \mathcal{C}) = \max_{e_j \in \mathcal{C}} \mathrm{sim}(e_i, e_j), \tag{13}$$

where $\mathrm{sim}$ is the maximum of embedding similarity, token-level Jaccard similarity, and line-span overlap for code units from the same file. Units with $\mathrm{Red}(e_i, \mathcal{C}) > 0.85$ are skipped unless they are required dependencies.

### A.4 Feedback and Execution Details

DySCoR maintains:

$$\sigma = \langle s, H, P, T, L, A \rangle, \tag{14}$$

where $s$ is the current stage, $H$ is interaction history, $P$ is generated patches, $T$ is the latest validation result, $L$ is suspected locations, and $A$ is activated conditional evidence. The phase controller starts with task understanding, moves to localization until a candidate exceeds $\theta_{\mathrm{loc}} = 0.60$, enters patch generation, and then moves to feedback revision or final validation. In feedback revision, it returns to generation if the failure overlaps the edited region and to localization if the failure points elsewhere or creates unrelated regressions.

After validation, confidence is updated as:

$$c_i^{(t+1)} = \mathrm{clip}_{[0,1]}\left(c_i^{(t)} + \Delta_i^{(t)}\right), \tag{15}$$

$$\begin{aligned} \Delta_i^{(t)} = r_{\mathrm{sel}} \cdot \mathbb{I}[e_i \in \mathcal{C}_s] \\ + r_{\mathrm{trace}} \cdot \mathrm{Trace}(e_i, T) \\ + r_{\mathrm{edit}} \cdot \mathrm{Edit}(e_i, p) \\ + r_{\mathrm{improve}} \cdot \mathrm{Improve}(T) \\ - r_{\mathrm{fail}} \cdot \mathrm{Contra}(e_i, T) \\ - r_{\mathrm{stale}} \cdot \mathrm{Stale}(e_i). \end{aligned} \tag{16}$$

**Algorithm 1** DySCoR for Agentic Repository-Level Code Repair

**Require:** Issue $q$, repository $\mathcal{R}$, logs $l$, budget $B$, max rounds $N$, agent $\mathcal{A}$
**Ensure:** Validated patch $p$ or failure report

$\mathcal{E} \leftarrow \mathrm{BuildEvidence}(q, \mathcal{R}, l)$
$\sigma \leftarrow \mathrm{InitializeState}(q, l)$
$P \leftarrow \emptyset$
**for** $t = 1$ to $N$ **do**
  $s \leftarrow \mathrm{PhaseController}(\sigma)$
  $M \leftarrow \mathrm{VisibilityPolicy}(\mathcal{E}, \sigma)$
  $\mathcal{V}_s \leftarrow \{e_i \in \mathcal{E} \mid M_{s,i} = 1 \vee (M_{s,i} = \rho \wedge g(e_i, s, \sigma) = 1)\}$
  $B_s \leftarrow \mathrm{AllocateBudget}(B, s, \sigma)$
  $\mathcal{C}_s \leftarrow \mathrm{SparseRoute}(\mathcal{V}_s, B_s, s, \sigma)$
  $\mathrm{Prompt}_s \leftarrow \mathrm{CompilePrompt}(s, \mathcal{C}_s, \sigma)$
  $a_s \leftarrow \mathcal{A}(\mathrm{Prompt}_s)$
  $\sigma \leftarrow \mathrm{ParseAgentOutput}(a_s, \sigma)$
  **if** $a_s$ contains a patch **then**
    $p \leftarrow \mathrm{ApplyPatch}(a_s, \mathcal{R})$
    $T \leftarrow \mathrm{ValidatePatch}(p, \mathcal{R})$
    $P \leftarrow P \cup \{p\}$
    **if** $T$ passes required validation **then**
      **return** $p$
    **end if**
    $\mathcal{E} \leftarrow \mathrm{AddDiffAndFeedbackEvidence}(\mathcal{E}, p, T)$
    $\mathcal{E} \leftarrow \mathrm{UpdateConfidence}(\mathcal{E}, p, T, \sigma)$
    $\sigma \leftarrow \mathrm{UpdateRepairState}(\sigma, p, T)$
  **end if**
  **if** $\mathrm{NeedFallback}(\sigma)$ **then**
    $\mathcal{E} \leftarrow \mathrm{RelaxVisibility}(\mathcal{E}, \sigma)$
  **end if**
**end for**
**return** $\mathrm{FailureReport}(P, \sigma)$

Here Contra is the contradiction score. Default weights are $r_{\mathrm{sel}} = 0.03$, $r_{\mathrm{trace}} = 0.08$, $r_{\mathrm{edit}} = 0.05$, $r_{\mathrm{improve}} = 0.08$, $r_{\mathrm{fail}} = 0.10$, and $r_{\mathrm{stale}} = 0.02$.

DySCoR compiles each stage prompt as:

$$\begin{aligned}\mathrm{Prompt}_s = I_s \oplus C_s^{\mathrm{shared}} \oplus C_s^{\mathrm{phase}} \\ \oplus C_s^{\mathrm{state}} \oplus O_s,\end{aligned} \tag{17}$$

where $I_s$ is the stage instruction and $O_s$ is the output schema. The minimal tool interface contains file search, file read, symbol search, edit application, and test execution, following tool-use motivation from Toolformer (Schick et al., 2023) and software-agent interfaces such as SWE-agent (Yang et al., 2024a). If the agent fails to localize, produces syntactically invalid patches, or fails validation for $m = 2$ consecutive rounds, DySCoR relaxes constraints in this order: additional local window, sibling symbols, direct callers and callees, files sharing test coverage, and broader retrieval candidates.

Let $n = |\mathcal{E}|$ and $k$ be selected units per stage. Computing visibility is $O(n)$. Greedy routing requires $O(n \log n)$ sorting in the worst case, or $O(n + k \log n)$ with a heap. Dependency expansion is linear in selected dependency edges. In practice, retrieval and visibility rules restrict routing to a few hundred candidates.

# B Additional Experimental Results

## B.1 Detailed Analysis

Figure 6b shows the stage-wise context distribution produced by DySCoR. In task understanding, shared issue and failure information accounts for 45% of routed tokens, while test and log evidence accounts for 37%. In localization, structural evidence such as symbol references and call relations becomes dominant, accounting for 35% of tokens. In patch generation, local code evidence increases to 57%, while structural evidence decreases to 11%. In feedback revision, feedback and diff evidence

Table 6: Localization quality. Superscripts on DySCoR report absolute differences from ToolAgent+StaticPack; green indicates a favorable change.

| Method | File Hit | | | Function Hit | | | Gold Ctx.↑ |
|---|---|---|---|---|---|---|---|
| | @1 | @3 | @5 | @1 | @3 | @5 | |
| BM25-RAG | 34.6 | 48.8 | 55.1 | 18.3 | 29.4 | 35.6 | 47.9 |
| Dense-RAG | 35.2 | 49.6 | 55.8 | 18.7 | 29.9 | 36.3 | 48.6 |
| Hybrid-RAG | 39.4 | 53.9 | 60.2 | 22.1 | 34.7 | 40.9 | 52.8 |
| StaticPack | 42.7 | 57.1 | 62.4 | 24.8 | 37.9 | 44.2 | 55.5 |
| ToolAgent | 45.8 | 59.7 | 63.9 | 25.6 | 39.4 | 45.8 | 57.2 |
| ToolAgent+Sta. | 47.9 | 61.8 | 65.7 | 27.1 | 41.6 | 48.0 | 59.3 |
| SWE-Fixer | 38.7 | 53.0 | 59.8 | 21.4 | 34.0 | 40.7 | 52.0 |
| SWE-agent | 40.6 | 55.2 | 61.0 | 22.5 | 35.8 | 42.3 | 53.6 |
| AutoCodeRover | 43.3 | 57.5 | 63.4 | 24.0 | 37.5 | 44.1 | 55.0 |
| CodeR | 44.8 | 59.1 | 64.5 | 25.5 | 39.2 | 45.7 | 56.4 |
| SWE-Search | 46.0 | 60.5 | 65.8 | 26.4 | 40.4 | 47.0 | 57.8 |
| MASAI | 46.9 | 61.2 | 66.4 | 26.8 | 41.0 | 47.5 | 58.6 |
| Agentless | 47.5 | 61.6 | 66.9 | 27.0 | 41.4 | 47.8 | 58.9 |
| **DySCoR** | **54.8**$^{+6.9}$ | **68.9**$^{+7.1}$ | **72.4**$^{+6.7}$ | **34.2**$^{+7.1}$ | **49.3**$^{+7.7}$ | **55.8**$^{+7.8}$ | **66.1**$^{+6.8}$ |

Table 7: Context efficiency. Superscripts on DySCoR report absolute differences from ToolAgent+StaticPack; green indicates a favorable change.

| Method | Input Tok.↓ | Peak Tok.↓ | Useful Ctx.↑ | Redun. Ctx.↓ | Rounds↓ | Tool Calls↓ |
|---|---|---|---|---|---|---|
| BM25-RAG | 66.8K | 15.4K | 25.1 | 39.6 | 3.5 | 7.8 |
| Dense-RAG | 68.5K | 15.9K | 26.2 | 38.9 | 3.5 | 8.0 |
| Hybrid-RAG | 71.2K | 16.7K | 27.4 | 37.8 | 3.6 | 8.4 |
| StaticPack | 84.6K | 27.3K | 31.2 | 33.7 | 3.7 | 6.1 |
| ToolAgent | 96.2K | 24.0K | 29.9 | 41.2 | 4.1 | 14.8 |
| ToolAgent+Sta. | 103.4K | 30.6K | 32.5 | 36.4 | 3.9 | 12.7 |
| SWE-Fixer | 76.5K | 21.8K | 29.6 | 35.2 | 3.5 | 7.2 |
| SWE-agent | 88.7K | 24.9K | 30.8 | 38.5 | 4.0 | 13.6 |
| AutoCodeRover | 91.3K | 25.6K | 31.4 | 37.2 | 3.8 | 12.1 |
| CodeR | 94.8K | 27.1K | 32.0 | 36.8 | 3.9 | 12.9 |
| SWE-Search | 116.2K | 31.5K | 33.0 | 39.6 | 4.7 | 18.4 |
| MASAI | 99.5K | 29.7K | 33.4 | 35.9 | 4.0 | 14.0 |
| Agentless | 69.4K | 22.5K | 34.2 | 31.0 | 3.4 | 6.8 |
| **DySCoR** | **58.4K**$^{-45.0K}$ | **18.8K**$^{-11.8K}$ | **42.6**$^{+10.1}$ | **18.5**$^{-17.9}$ | **3.1**$^{-0.8}$ | **8.9**$^{-3.8}$ |

accounts for 28%, and local code evidence accounts for 28%. These distributions match the intended design: DySCoR exposes broad structural context during localization, precise code context during generation, and validation-oriented context during revision. Table 8 analyzes feedback-driven routing actions. When validation produces the same traceback but fewer failing tests, DySCoR performs local revision in 34.8% of feedback cases, and 41.2% of these cases are eventually resolved. When validation reveals a new traceback in a dependency, DySCoR expands dependency context in 21.6% of cases, with a 33.5% later success rate. When the failure is unchanged and does not overlap with the edited region, DySCoR triggers re-localization in 28.9% of cases. This action has a lower but still meaningful success rate of 29.7%. New regressions account for 14.7% of feedback cases and are the hardest to recover from, with a success rate of 22.4%.

Table 9 categorizes unresolved DySCoR cases. The largest category is wrong or incomplete localization, accounting for 32.4% of failures. The second largest category is semantically incomplete patches, accounting for 25.8%. Environment or reproduction issues account for 14.6%, while large architecture-level changes account for 9.1%. These results indicate that DySCoR reduces context inefficiency but does not eliminate two fundamental repair challenges: identifying deeply hidden root causes and generating semantically complete edits across complex code paths.

Table 8: Feedback actions.

| Feedback Pattern | Action Share | Success After Action |
|---|---|---|
| Same traceback, fewer failing tests | 34.8 | 41.2 |
| New traceback in dependency | 21.6 | 33.5 |
| Unchanged failure, no edit overlap | 28.9 | 29.7 |
| New unrelated regression | 14.7 | 22.4 |

Table 9: Failure categories.

| Failure Category | Share Among Unresolved Cases |
|---|---|
| Wrong or incomplete localization | 32.4 |
| Semantically incomplete patch | 25.8 |
| Environment or reproduction failure | 14.6 |
| Insufficient validation signal | 10.7 |
| Large multi-file architectural change | 9.1 |
| Parser or tool limitation | 7.4 |

Table 10 presents three representative cases. In Case A, StaticPack exposes 44 snippets and 19.2K tokens during patch generation, while DySCoR routes 21 snippets and 10.8K tokens by suppressing low-confidence localization candidates. The patch succeeds after three rounds. In Case B, DySCoR activates dependency context after a new traceback appears, reducing exposed snippets from 36 to 27 and context tokens from 17.6K to 14.2K, resolving the instance in four rounds. In Case C, DySCoR reduces context tokens from 22.5K to 13.1K compared with StaticPack but fails because the final patch requires a multi-file behavioral change. These cases show both the benefit and limitation of dynamic routing: it can reduce distraction and recover from some localization errors, but it cannot guarantee success when the required semantic edit is beyond the agent's generation ability.

Table 10: Case study.

| Case | Static Snip. | DySCoR Snip. | Static Tok. | DySCoR Tok. | Outcome |
|---|---|---|---|---|---|
| A: Distracting candidates | 44 | 21 | 19.2K | 10.8K | Pass in 3 rounds |
| B: New dependency traceback | 36 | 27 | 17.6K | 14.2K | Pass in 4 rounds |
| C: Multi-file semantic edit | 51 | 29 | 22.5K | 13.1K | Fail after 5 rounds |