RepoNav: From Snippet Retrieval to File-Centered Repository Navigation for Code Agents

Anonymous ACL submission

Abstract

Solving repository-level code tasks requires LLM-based agents to use code search tools to navigate large codebases and identify a small set of relevant files and functions. However, current retrieval tools typically return flat lists of isolated code snippets: such lists can surface relevant files, but provide insufficient structure for agents to distinguish the target function from semantically similar alternatives in the same file. We introduce RepoNav, a lightweight post-retrieval interface that reorganizes retrieved snippets into a file-centered navigation scaffold. By presenting compact structural cues and candidate targets, this scaffold guides on-demand file-structure browsing, helping agents compare sibling symbols before selecting a target function. Across diverse models on LocBench, RepoNav improves function-level localization and narrows the file-to-function gap. Controlled ablations demonstrate that these gains come from structured evidence organization rather than simply exposing additional file structure, and the approach also improves performance on a repository-level question-answering benchmark. Our code is available at https://anonymous.4open.science/r/reponav-release-EC30.

1 Introduction

Large language models (LLMs) have driven rapid progress in autonomous software engineering. In repository-scale tasks such as issue resolution and bug localization, agents must operate across two levels of granularity: identifying the relevant files and pinpointing the specific functions within them. Modern code agents equipped with retrieval or agentic search tools (Zhang et al., 2023; Wang et al., 2025b; Yang et al., 2024) can often surface relevant files, yet function-level localization still lags significantly behind. We analyze this behavior using a common mini-SWE-agent scaffold (Yang et al., 2024), keeping the underlying agent loop fixed across exploration interfaces.

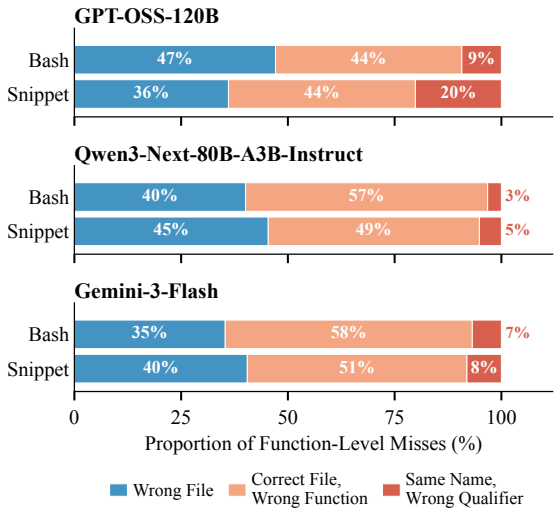


Figure 1: Failure taxonomy for function-level misses. Percentages are computed over misses within each model and setting. A substantial fraction of misses occurs after the agent has reached the correct file, indicating within-file navigation failures. Percentages may not sum to 100 due to rounding.

Why do agents that reach the correct file still miss the target function? Our analysis reveals *premature anchoring*: agents often commit to a salient nearby symbol before inspecting sibling definitions or file-structure views. Flat snippet retrieval reinforces this behavior by presenting relevant chunks and distractors as isolated evidence, with limited cues about what else the file contains. As Figure 1 shows, *Correct File, Wrong Function* accounts for 44–58% of function-level misses across models, confirming within-file navigation as a central bottleneck in function-level localization.

To mitigate this bottleneck under a fixed retrieval substrate, we introduce **RepoNav**, a lightweight framework for repository navigation. RepoNav uses the same dense index and chunk-level retrieval results as the snippet-search baseline, but reorganizes retrieved evidence into a file-centered navigation scaffold. The scaffold exposes compact

structural cues, candidate targets, and continuation hints, and supports on-demand file-structure browsing through `list_symbols`. This design separates file discovery from within-file verification: retrieval proposes candidate files, while RepoNav helps the agent decide which internal symbols to inspect next. Rather than asking agents to read more code upfront, RepoNav makes candidate files structurally expandable and encourages verification of sibling symbols before commitment. Unlike heavyweight graph-based systems (Ouyang et al., 2025; Liu et al., 2025; Chen et al., 2025) that rely on repository-wide graph construction or static-analysis infrastructure, RepoNav is an agent-facing interface layer built on top of existing snippet retrieval outputs.

Across seven models on LocBench (Chen et al., 2025), RepoNav improves function-level localization and reduces the gap between file discovery and function discovery. A tool-matched Snippet+ListSym baseline shows that file-structure browsing is useful, while RepoNav further improves over this baseline by making such browsing more actionable through file-centered organization. Results on SWE-QA-Bench (Peng et al., 2025) show that RepoNav also improves repository-level question answering beyond explicit localization. Our contributions are:

1. **An empirical diagnosis of the file-to-function gap.** We show that a large fraction of function-level failures occur after the agent has already reached the correct file, identifying within-file navigation as a major bottleneck in repository-scale code localization.
2. **A structurally expandable, file-centered navigation interface.** RepoNav reorganizes flat snippet-style outputs into file-centered scaffolds while keeping the underlying dense index and raw retrieval scores fixed. It integrates `list_symbols` as an on-demand file-structure browsing tool, enabling agents to inspect lightweight file skeletons and compare sibling symbols without reading full file bodies.
3. **Behavioral and tool-matched evidence for scaffolded navigation.** A Snippet+ListSym baseline controls for access to file-structure browsing, and controlled ablations show that RepoNav provides additional gains by making structural evidence more actionable and efficient for agent exploration.

2 Background and Related Work

2.1 Code Agents and Exploration Interfaces

SWE-bench (Jimenez et al., 2024) has become a standard benchmark for autonomous code agents (Zhang et al., 2024; Yang et al., 2024; Wang et al., 2025a; Antoniades et al., 2025; Xia et al., 2025), which navigate repositories through interleaved reasoning and tool use (Yao et al., 2023). These agents typically interact with repositories through bash-style exploration or snippet-style search interfaces. Retrieval-augmented generation (Lewis et al., 2020) has been widely adopted for injecting external knowledge into language models; in the code domain, retrieval-based methods such as RepoCoder (Zhang et al., 2023) and CodeRAG-Bench (Wang et al., 2025b) extend this paradigm to repository-level tasks. However, snippet-search interfaces usually present retrieved chunks as flat, chunk-centered lists, while bash-style interfaces expose the raw repository but require the agent to infer structure from command outputs. In both cases, the interface provides limited support for turning retrieved evidence into navigable action choices. This limitation is especially problematic for function-level localization, where the agent must compare multiple candidate symbols after reaching a relevant file rather than merely identify the file itself.

2.2 The File-to-Function Gap

A natural assumption is that function-level localization failures mainly arise because the correct file was never surfaced. Our failure analysis shows otherwise. We categorize each function-level miss into three types: *Wrong File* (the predicted target lies outside the gold file), *Correct File, Wrong Function* (the agent reaches the correct file but selects the wrong function), and *Same Name, Wrong Qualifier* (the prediction matches the local symbol name but not the fully qualified target). As Figure 1 shows, a substantial share of misses occurs after the agent has already reached the correct file: *Correct File, Wrong Function* accounts for 44–58% of all misses. This reveals a file-to-function localization gap: reaching the correct file does not reliably translate into identifying the correct function.

Premature anchoring. This asymmetry gives rise to a recurring failure mode: the agent commits to the most salient nearby symbol before inspecting sibling or structurally adjacent candidates. Typical

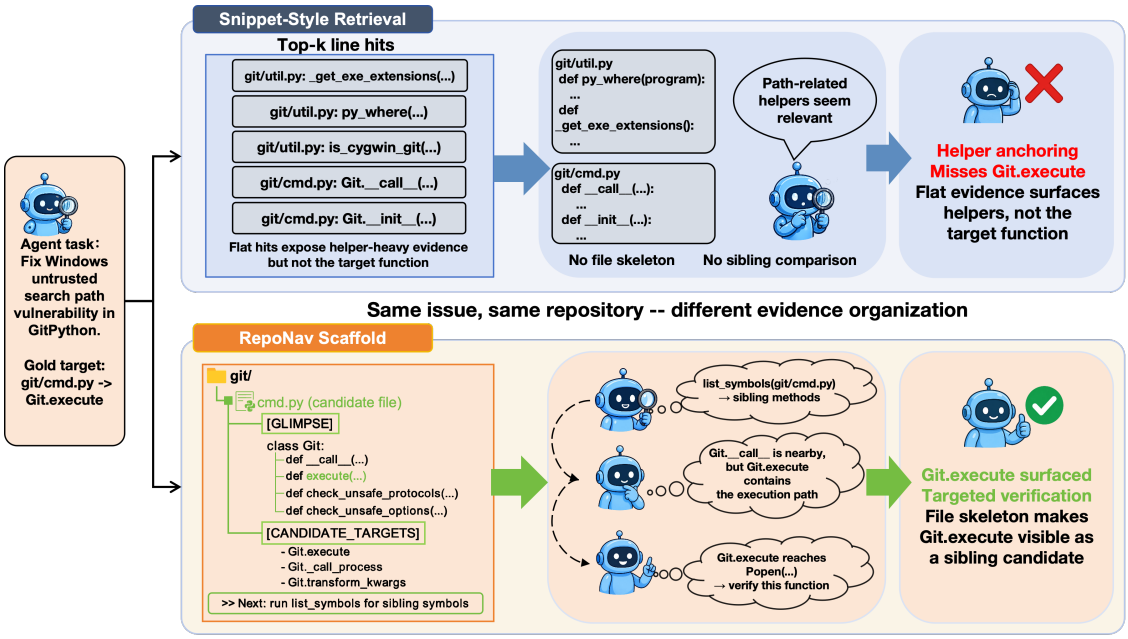


Figure 2: A real GitPython case illustrating function-level divergence under different evidence organization. Top: snippet-style retrieval returns helper-heavy flat evidence and misses `Git.execute`. Bottom: RepoNav organizes the candidate file into a file-centered scaffold and prompts on-demand file-structure browsing with `list_symbols`, making `Git.execute` visible as a sibling candidate for targeted verification.

anchors include public wrappers, request handlers, or entry methods, as illustrated in Figure 2 (see Appendix F for further case studies). This behavior is reminiscent of the "lost in the middle" phenomenon in long-context models (Liu et al., 2024a), where information position affects utilization; here, the agent fixates on the most prominent symbol rather than systematically comparing alternatives. The interface does not encourage continued exploration, motivating a navigation-oriented view: post-retrieval interfaces should organize evidence into actionable choices that make continued exploration easier than premature commitment.

Positioning. Recent efforts have scaled structural representations to the repository level. RepoGraph (Ouyang et al., 2025) and CodexGraph (Liu et al., 2025) map entire codebases into graph databases; LocAgent (Chen et al., 2025) constructs directed code graphs for multi-hop traversal; LingmaAgent (Ma et al., 2024) builds repository-level knowledge representations with Monte Carlo tree search for exploration; and GraphCoder (Liu et al., 2024b) uses code context graphs for retrieval-augmented completion. These methods rely on repository-wide graph construction and specialized traversal or query mechanisms, which introduce additional infrastructure and interaction complexity for agent systems. In the broader NLP setting, StructRAG (Li et al., 2025) converts retrieved documents into task-appropriate structured formats to aid reasoning; RepoNav shares this motivation of post-retrieval restructuring but targets code agents and operates as a lightweight interface layer rather than a general document transformation framework. Rather than precomputing a global graph, RepoNav dynamically extracts local structural cues from only the top-ranked retrieved files and presents them as an immediately consumable text-based scaffold.

Research questions. The preceding analysis motivates three questions: (**RQ1**) Does the file-to-function gap exist consistently across models, and does a navigation-oriented interface reduce it? (**RQ2**) Is the behavioral shift driven by the *volume* of structural information or by how that information is *organized*? (**RQ3**) Does the benefit transfer beyond localization to repository-level question answering?

3 RepoNav: A Navigation Interface for Repository Exploration

RepoNav is a post-retrieval interface layer rather than a new retriever. It does not modify the raw dense chunk retrieval results or similarity scores used by the snippet-search baseline. Instead, it deterministically aggregates retrieved chunks into file-level candidates and changes how the retrieved evidence is presented to the agent. RepoNav turns

raw retrieval hits into cues for comparing plausible symbols before commitment.

3.1 Interface Design

RepoNav follows three design principles. First, it aggregates chunk-level evidence into candidate files, since agents ultimately inspect, reason over, or modify files rather than isolated chunks. Second, it exposes compact file-internal structure, giving the agent enough context to compare nearby symbols without dumping the full file. Third, it adds explicit continuation cues, so that retrieved evidence is treated as a starting point for targeted inspection rather than as a terminal answer.

Concretely, RepoNav serializes the top-ranked files as a plain-text, indentation-based tree. Each file entry contains three blocks.

`[ANCHORS]`: Entry points. After dense retrieval results are aggregated into candidate files, RepoNav identifies *anchor symbols* within those files using a lightweight lexical match between symbol names and the retrieval query issued by the agent. These anchors provide query-relevant entry points into each file. When no symbol name matches the query, RepoNav falls back on the structural sketch described below. Each anchor is annotated with its line span and compact same-file call context, giving the agent a grounded starting point for inspection. In our implementation, anchors are capped at two per file.

`[GLIMPSE]`: Structural sketch. To prevent fixation on anchors, RepoNav exposes up to three non-anchor symbols from the same file, or up to five when the top-ranked file has no anchor match. These symbols provide a compact sketch of what else the file contains, such as additional classes, functions, or methods that may not directly match the query but are structurally relevant. Rather than dumping full function bodies or long signatures, `[GLIMPSE]` presents schematic entries, allowing the agent to notice alternative candidate symbols with minimal context cost.

`[CANDIDATE_TARGETS]`: Actionable target shortlist. This block turns the structural sketch into a compact set of candidate inspection targets. It summarizes anchors, selected glimpse symbols, and lightweight same-file call context without adding new retrieval evidence or full function bodies. The shortlist is capped at four entries and paired with fixed continuation hints, encouraging the agent to inspect sibling symbols before committing. The call context serves only as a navigation cue rather than sound static analysis.

`list_symbols`: File-structure browsing. RepoNav exposes `list_symbols` as an on-demand file-structure browsing tool. Given a file path, it returns a lightweight file skeleton, including imports, classes, functions, methods, line ranges, and optional signatures, but not full function bodies. It does not perform semantic search or rank target functions. Instead, it lets the agent inspect the editable objects inside a candidate file and compare sibling symbols before selecting a target function.

The scaffold itself remains plain text and requires no graph query language. It can therefore be consumed by standard text-based code agents while still exposing enough structure to support targeted navigation. The full serialization budgets, ordering rules, and a worked output example are provided in Appendix G (Figure 8).

3.2 Implementation

RepoNav is implemented as a two-stage post-retrieval pipeline. Both stages require minimal computation and no offline preprocessing beyond the standard dense chunk index used by the snippet baseline. The specific retrieval model and indexing configuration are detailed in Section 4.1.

Stage 1: Evidence aggregation. Given an agent retrieval query, the dense retrieval backend returns top-ranked code chunks and their associated retrieval scores. RepoNav aggregates these chunk-level scores into a file-level ranking using a hybrid scoring rule. For each candidate file f, let C_f denote the retrieved chunks from f, with their similarity scores sorted as $s_{f,(1)} \geq s_{f,(2)} \geq \dots$. We aggregate the top $m_f = \min(m, |C_f|)$ scores as:

$$\mathrm{Score}(f) = \alpha s_{f,(1)} + (1-\alpha)\frac{1}{m_f}\sum_{i=1}^{m_f} s_{f,(i)}.$$

We set m=3 in all experiments. Files with fewer than m retrieved chunks are averaged over their available chunks. The first term captures peak evidence strength, while the second rewards files supported by multiple high-scoring regions rather than a single spurious match. A parameter sensitivity analysis in Appendix A shows that performance is stable across a wide range of α values, with ranking-oriented metrics peaking near α=0.5 and recall metrics plateauing for $\alpha \geq 0.7$. Since α only

affects the file ranking before agent interaction, we select its value on held-out development splits using offline retrieval metrics and then keep it fixed for all downstream experiments.

Stage 2: On-the-fly structural extraction. RepoNav dynamically parses only the top-k files from Stage 1, with k=5 fixed across experiments by default. The initial scaffold auto-expands full three-block entries for the top three candidate files, while the remaining parsed files remain available for on-demand structure browsing. For Python files, the built-in `ast` module extracts function and class definitions, import statements, and a lightweight same-file call neighborhood. The call neighborhood connects functions only when a call expression can be matched to another function defined in the same file; it does not attempt sound static analysis, cross-file dependency recovery, or inter-procedural call-graph construction. When call information is unavailable, RepoNav falls back to file-structure skeletons without caller and callee cues. These extracted cues are assembled into the three-block interface described in Section 3.1.

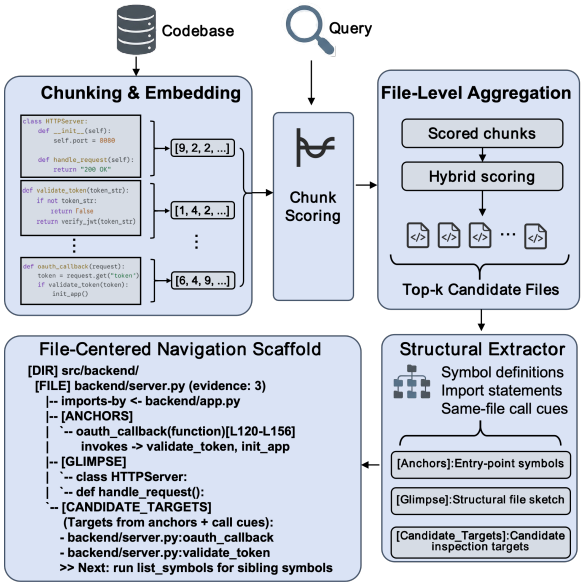


Figure 3: Implementation overview of RepoNav. The retrieval backend is shared with the snippet-search baseline; RepoNav changes the post-retrieval organization of evidence by aggregating chunks into candidate files and extracting lightweight file structure.

Figure 3 summarizes the two-stage post-retrieval pipeline. The process is deterministic and local: RepoNav aggregates retrieved chunks into candidate files and extracts structure only from the top-ranked files, avoiding repository-wide graph construction.

4 Experimental Evaluation

We now evaluate the three research questions introduced in Section 2: whether the file-to-function localization gap holds consistently across models and can be reduced by a navigation-oriented interface (RQ1), whether the observed behavioral shift is driven by the volume of structural information or by how that information is organized (RQ2), and whether the benefit transfers beyond localization to repository-level question answering (RQ3).

4.1 Experimental Setup

Benchmark. We evaluate on LocBench (Chen et al., 2025), a repository-level code localization benchmark with 560 Python instances, each pairing a natural language issue with gold files and functions from the corresponding patch.

Agent settings. All experiments use a common mini-SWE-agent scaffold (Yang et al., 2024) with a fixed agent loop, environment, and interaction budget. We compare four interfaces: **Bash** (shell only), **Snippet Search** (adding a dense search tool returning flat chunk lists), **Snippet+ListSym** (adding the same `list_symbols` tool used by RepoNav to the flat snippet interface, controlling for file-structure browsing access), and **RepoNav** (reorganizing the same dense retrieval substrate into a file-centered scaffold with `list_symbols` access). All three retrieval-based settings share the same chunk index, embedding model, retrieval backend, and raw chunk-level similarity scores. Following Chen et al. (2025), each function is embedded as a single chunk using CodeRankEmbed (Suresh et al., 2025); further indexing details are provided in Appendix B.

Models and metrics. We evaluate seven proprietary and open-weight models.[1] We report Accuracy@k and Recall@k at file, module, and function levels. Accuracy@k requires all gold targets (up to k) to appear in the top-k predictions; Recall@k measures the fraction recovered. Module-level matching maps predictions to their enclosing module. Transfer is additionally evaluated on SWE-QA-Bench (Peng et al., 2025); its setup is described with the RQ3 results.

[1]Gemini 3 Flash (Doshi and Gemini Team, 2025), MiniMax-M2.5 (MiniMax, 2026), gpt-oss-120b (OpenAI, 2025), Qwen2.5-72B-Instruct (Qwen Team, 2024), Qwen3-Coder-30B-A3B-Instruct (Qwen Team, 2025a), Qwen3-Next-80B-A3B-Instruct (Qwen Team, 2025b), and GLM-4.7 (Z.AI, 2025). Figures and tables use shortened names.

Table 1: Main LocBench localization results across seven models. We report File Acc@5, Module Acc@5, Function Acc@5, and Function Rec@10. Methods are abbreviated as **B** = Bash, **S** = Snippet Search, **S+L** = Snippet+ListSym, and **R** = RepoNav; **S+L** is a tool-matched snippet baseline with the same `list_symbols` access as RepoNav. Bold marks the best setting within each model block and metric group. Full Accuracy@k and Recall@k results are provided in Appendix C, Table 6.

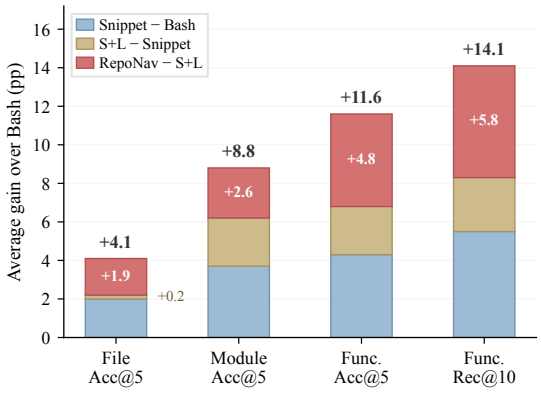
	File Acc@5				Module Acc@5				Function Acc@5				Function Rec@10			
Model	B	S	S+L	R	B	S	S+L	R	B	S	S+L	R	B	S	S+L	R
Qwen2.5-72B	0.589	0.680	0.673	**0.698**	0.396	0.461	0.562	**0.570**	0.254	0.300	0.384	**0.434**	0.307	0.377	0.490	**0.549**
GPT-OSS-120B	0.705	0.759	0.759	**0.770**	0.500	0.613	0.630	**0.650**	0.339	0.464	0.488	**0.527**	0.417	0.571	0.586	**0.618**
Qwen3-Next-80B	0.734	0.732	0.733	**0.739**	0.523	0.568	0.589	**0.632**	0.339	0.411	0.454	**0.498**	0.410	0.517	0.564	**0.622**
Qwen3-Coder-30B	0.700	0.693	0.707	**0.727**	0.564	0.584	0.598	**0.629**	0.420	0.459	0.476	**0.516**	0.498	0.551	0.572	**0.618**
MiniMax-M2.5	0.760	0.764	0.780	**0.803**	0.660	0.671	0.671	**0.691**	0.525	0.532	0.534	**0.601**	0.633	0.641	0.652	**0.708**
GLM-4.7	0.808	0.821	0.812	**0.839**	0.685	0.698	0.709	**0.734**	0.585	0.600	0.577	**0.632**	0.714	0.722	0.695	**0.783**
Gemini-3-Flash	0.836	0.823	0.822	**0.840**	0.709	0.702	0.717	**0.752**	0.625	0.623	0.648	**0.689**	0.737	0.722	0.739	**0.803**
Average	0.733	0.753	0.755	**0.774**	0.577	0.614	0.639	**0.665**	0.441	0.484	0.509	**0.557**	0.531	0.586	0.614	**0.672**

4.2 RQ1: File-to-Function Gap on LocBench

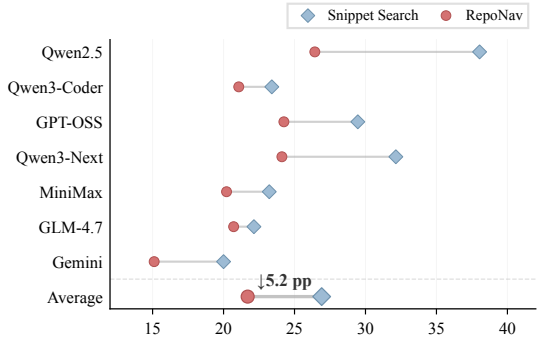
Table 1 reports the main LocBench results. To control for tool access, Snippet+ListSym (S+L) adds `list_symbols` to the flat snippet interface. This lets us separate file-structure browsing gains (S→S+L) from RepoNav's additional file-centered organization gains (S+L→R), as visualized in Figure 4.

Finding 1: File-structure browsing helps, and RepoNav adds further function-level gains. Adding `list_symbols` to the flat snippet interface (S+L) already improves function-level localization over Snippet Search: average Function Acc@5 rises from 0.484 to 0.509 (+2.5 points). However, RepoNav yields a further gain to 0.557 (+4.8 points over S+L), indicating that the file-centered scaffold provides substantial additional benefit beyond file-structure browsing access alone. As Figure 4a shows, the S+L→R increment is largest at the module and function levels, precisely where within-file navigation matters most. This two-stage decomposition indicates that RepoNav's gains cannot be fully explained by the availability of `list_symbols`; the organization of retrieved evidence into actionable, file-centered cues is a key contributing factor.

Finding 2: The file-to-function gap persists under snippet retrieval; RepoNav narrows it substantially. We define the *file-to-function gap* as the difference between File Acc@5 and Function Acc@5. Under Bash, this gap averages 29.2 points. Snippet Search slightly narrows it to 26.9 points by improving both file- and function-level accuracy, but the gap remains large because function-level gains do not keep pace with file-level gains, even after relevant files are retrieved. RepoNav reduces the gap to 21.7 points, a 5.2-point reduction over Snippet Search. This indicates that structured navigation closes a qualitatively different bottleneck than flat retrieval.



(a) Three-stage gain decomposition.



(b) File-to-function gap (pp; lower is better)

Figure 4: LocBench gain decomposition and file-to-function gap. (a) Average gains over Bash are decomposed into retrieval gains (S−B), file-structure browsing gains ((S+L)−S), and RepoNav gains under matched `list_symbols` access (R−(S+L)). (b) RepoNav reduces the average file-to-function gap from 26.9 to 21.7 percentage points compared with Snippet Search.

Finding 3: RepoNav remains effective even when file-level performance is already high. For stronger models such as GLM-4.7 and Gemini-3-Flash, File Acc@5 already exceeds 0.80 under Bash or Snippet Search, yet RepoNav still improves Function Acc@5 by 3.2 and 6.6 points (5.3% and 10.6% relative), respectively. Notably, Snippet Search yields negligible or even slightly negative function-level changes relative to Bash for these models (e.g., −0.2 points on Gemini-3-Flash), consistent with the retrieval-only gap reported in Table 5 (Appendix B). RepoNav continues to yield gains, indicating that once relevant files are reachable, the remaining bottleneck increasingly lies in navigating and verifying evidence within those files. Together, these findings provide evidence that file discovery and function discovery are qualitatively different challenges, and that snippet-style retrieval provides insufficient structural context for the latter.

4.3 RQ2: Does Structure Help by Volume or by Actionability?

RQ1 shows that RepoNav improves function-level localization, but the mechanism remains unclear. Does the improvement come from exposing more structural information to the agent, or from organizing retrieved evidence into an action-oriented interface that encourages targeted exploration?

To answer this question, we conduct a controlled ablation on the full 560-instance LocBench benchmark using the same model, dense retrieval backend, search-first workflow, tool access, and verification constraints. Only the presentation of retrieved evidence differs. **File-Only** exposes aggregated candidate files without file-internal structure. **Inline Scaffold** exposes richer file-internal structure directly in the retrieval output. **Tree Scaffold (RepoNav)** organizes structural evidence into a compact, navigable scaffold with selective cues and explicit next-step hints. This design separates file-level aggregation, in-context structural volume, and action-oriented organization, allowing us to test whether structure helps by being larger or by being easier to act on.

Table 2 shows that endpoint localization improves from File-Only to Inline Scaffold and further to Tree Scaffold under matched retrieval, workflow, tool access, and verification constraints. Function Acc@5 increases from 32.61% to 46.96% and 52.13%, while Function Rec@10 increases from 44.29% to 53.84% and 60.98%. Thus, exposing file-internal structure is important, but the compact tree scaffold provides additional gains beyond simply placing more structure in the retrieval output.

Table 2: RQ2 controlled ablation on the full 560-instance LocBench benchmark using GPT-OSS-120B. All settings share the same retrieval backend, tool access, workflow, and verification constraints; only evidence presentation differs. Acc@5/Rec@10 are percentages measuring endpoint localization, while the remaining columns summarize tool use and exploration efficiency.

	Endpoint Localization		Behavior & Efficiency			
Method	Acc@5 (%)	Rec@10 (%)	ListSym%	Files Insp.	Steps	Tokens
File-Only	32.61	44.29	39.11	2.587	10.587	50.2k
Inline Scaffold	46.96	53.84	36.96	2.457	9.630	56.2k
Tree Scaffold	**52.13**	**60.98**	**56.61**	**2.304**	**9.391**	**47.2k**

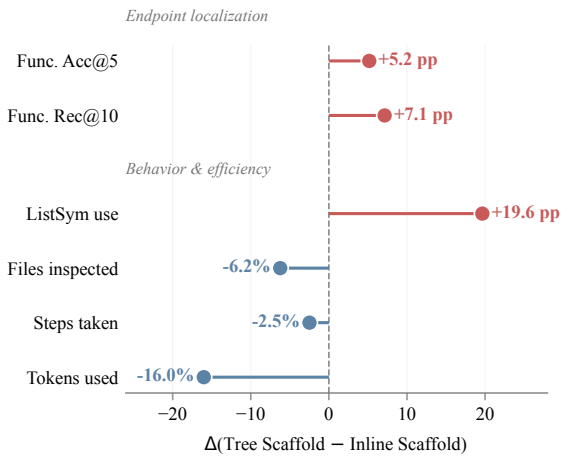
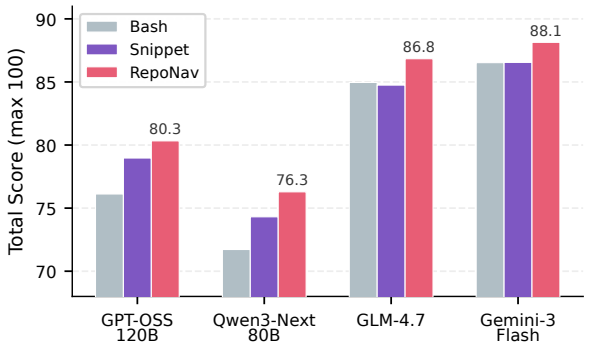


Figure 5: Tree Scaffold vs. Inline Scaffold in the RQ2 controlled ablation. Values report Tree Scaffold minus Inline Scaffold. Endpoint localization and ListSym use are absolute percentage-point changes; efficiency metrics are relative percentage changes, where negative values indicate lower cost.

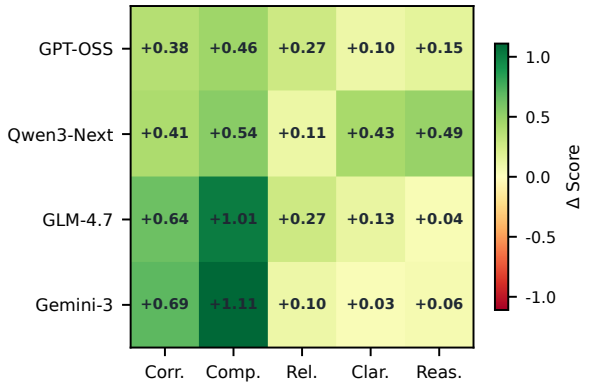
The behavioral metrics show a similar pattern. Compared with Inline Scaffold, Tree Scaffold inspects fewer files, takes fewer steps, and uses fewer tokens, while invoking `list_symbols` more frequently. Figure 5 makes this contrast explicit: Tree Scaffold increases `list_symbols` use by 19.65 percentage points (56.61% vs. 36.96%, a 53% relative increase) while using 16% fewer tokens. Together with the evidence-funnel analysis in Appendix D, this demonstrates that Tree Scaffold improves post-inspection actionability rather than simply broadening inspection.

4.4 RQ3: Does the Benefit Transfer Beyond Localization?

The preceding analyses show that RepoNav improves function-level localization on LocBench. We further ask whether RepoNav also benefits



(a) Overall answer quality.



(b) Per-dimension Δ (RepoNav − Snippet Search).

Figure 6: SWE-QA-Bench overall results. (a) RepoNav achieves the highest total score across all four models; exact values in Table 3. (b) Gains concentrate in Correctness and Completeness, consistent with the localization findings.

repository-level understanding beyond explicit localization. To this end, we evaluate on SWE-QA-Bench (Peng et al., 2025), a repository-level question-answering benchmark that requires agents to locate, inspect, and synthesize evidence from code repositories. Unlike RQ1, this experiment is intended as a transfer probe of the end-to-end RepoNav setting rather than a causal ablation of tool access. We compare three methods, Bash, Snippet Search, and RepoNav, using the same agent setup, interaction budget, and evaluation protocol. We use the per-model intersection of questions successfully completed and scored by all three methods.

Answer quality. Table 3 shows that RepoNav obtains the highest total score across all four models, with an average gain of +1.76 over Snippet Search and +3.07 over Bash. The gains concentrate in Correctness and Completeness (Figure 6b), showing that structured navigation mainly helps agents find and synthesize the right repository evidence rather than merely improving surface-level answer style. A similar pattern appears for the two strongest models: Snippet Search yields negligible changes relative to Bash, including a slight drop for GLM-4.7 (84.75 vs. 84.96) and a near-tie for Gemini-3-Flash (86.55 vs. 86.53), whereas RepoNav yields clear improvements (86.84 and 88.14). A question-type breakdown (Appendix E.2) shows that gains span all categories, with the largest improvements on *Why* questions requiring cross-file evidence synthesis. This pattern shows that the scaffold supports repository-level question answering by helping agents assemble evidence chains across functions and files rather than only localizing isolated targets.

Table 3: SWE-QA-Bench total scores. Scores are reported on the per-model common subset successfully completed and judged for all three methods. Δ_S denotes RepoNav minus Snippet Search, and Δ_B denotes RepoNav minus Bash. Full dimension-level scores are reported in Appendix E, Table 8.

Model	Bash	Snippet	RepoNav	Δ_S	Δ_B
GPT-OSS-120B	76.12	78.97	**80.33**	+1.36	+4.21
Qwen3-Next-80B	71.72	74.31	**76.29**	+1.98	+4.57
GLM-4.7	84.96	84.75	**86.84**	+2.09	+1.88
Gemini-3-Flash	86.53	86.55	**88.14**	+1.59	+1.61
Average	79.83	81.15	**82.90**	+1.76	+3.07

Summary. The SWE-QA results show that RepoNav also improves repository-level question answering beyond explicit localization. Across four models, RepoNav consistently outperforms both Bash and Snippet Search.

5 Conclusion

This paper investigates how the organization of retrieval outputs shapes the exploration behavior of LLM-based code agents at the repository scale. Across seven models on LocBench, we identify a persistent file-to-function gap: agents can often reach relevant files, yet still fail to localize the correct function. RepoNav narrows this gap by reorganizing retrieved snippets into a lightweight, file-centered navigation scaffold without changing the underlying retriever. Controlled ablations further show that simply exposing more file-structure information is insufficient; RepoNav's gains come from organizing retrieved evidence into a structured, navigable form. Overall, these findings show that retrieval-augmented code agents depend not only on what evidence is retrieved, but also on how that evidence is organized for exploration.

Limitations

Our current evaluation focuses on Python repositories and two benchmarks, LocBench and SWE-QA-Bench. Although RepoNav is designed as a lightweight interface layer rather than a Python-specific method, our implementation currently uses Python's ast module for file-structure extraction. We therefore leave validation on repositories written in other programming languages, such as Java, C, C++, Go, and Rust, to future work. These languages may require language-specific structure extractors, scaffold serialization rules, and evaluation settings. Extending RepoNav to these settings is an important step toward building a more general and practical repository navigation interface for code agents. Our experiments focus on localization and repository-level question answering. Evaluating RepoNav in downstream patch generation, long-horizon maintenance tasks, and interactive developer workflows remains future work.

References

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. 2025. SWE-search: Enhancing software agents with Monte Carlo tree search and iterative refinement. In *The Thirteenth International Conference on Learning Representations*.

Zhaoling Chen, Robert Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor Prasanna, Arman Cohan, and Xingyao Wang. 2025. LocAgent: Graph-guided LLM agents for code localization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8697–8727, Vienna, Austria. Association for Computational Linguistics.

Tulsee Doshi and Gemini Team. 2025. Gemini 3 Flash: Frontier intelligence built for speed. https://blog.google/products-and-platforms/products/gemini/gemini-3-flash/. Accessed: 2026-05-23.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*.

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. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Zhuoqun Li, Xuanang Chen, Haiyang Yu, Hongyu Lin, Yaojie Lu, Qiaoyu Tang, Fei Huang, Xianpei Han, Le Sun, and Yongbin Li. 2025. StructRAG: Boosting knowledge intensive reasoning of LLMs via inference-time hybrid information structurization. In *The Thirteenth International Conference on Learning Representations*.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Wei Liu, Ailun Yu, Daoguang Zan, Bo Shen, Wei Zhang, Haiyan Zhao, Zhi Jin, and Qianxiang Wang. 2024b. GraphCoder: Enhancing repository-level code completion via coarse-to-fine retrieval based on code context graph. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 570–581. Association for Computing Machinery.

Xiangyan Liu, Bo Lan, Zhiyuan Hu, Yang Liu, Zhicheng Zhang, Fei Wang, Michael Qizhe Shieh, and Wenmeng Zhou. 2025. CodexGraph: Bridging large language models and code repositories via code graph databases. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 142–160, Albuquerque, New Mexico. Association for Computational Linguistics.

Yingwei Ma, Qingping Yang, Rongyu Cao, Binhua Li, Fei Huang, and Yongbin Li. 2024. Alibaba LingmaAgent: Improving automated issue resolution via comprehensive repository exploration. *Preprint*, arXiv:2406.01422.

MiniMax. 2026. MiniMax-M2.5. https://huggingface.co/MiniMaxAI/MiniMax-M2.5. Model card. Accessed: 2026-05-25.

OpenAI. 2025. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Siru Ouyang, Wenhao Yu, Kaixin Ma, Zilin Xiao, Zhihan Zhang, Mengzhao Jia, Jiawei Han, Hongming Zhang, and Dong Yu. 2025. RepoGraph: Enhancing AI software engineering with repository-level code graph. In *The Thirteenth International Conference on Learning Representations*.

Weihan Peng, Yuling Shi, Yuhang Wang, Xinyun Zhang, Beijun Shen, and Xiaodong Gu. 2025. SWE-QA: Can language models answer repository-level code questions? *Preprint*, arXiv:2509.14635.

Qwen Team. 2024. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Qwen Team. 2025a. Qwen3-Coder-30B-A3B-Instruct. https://huggingface.co/Qwen/Qwen3-Coder-30B-A3B-Instruct. Model card. Accessed: 2026-05-25.

Qwen Team. 2025b. Qwen3-Next-80B-A3B-Instruct. https://huggingface.co/Qwen/Qwen3-Next-80B-A3B-Instruct. Model card. Accessed: 2026-05-25.

Tarun Suresh, Revanth Gangi Reddy, Yifei Xu, Zach Nussbaum, Andriy Mulyar, Brandon Duderstadt, and Heng Ji. 2025. CoRNStack: High-quality contrastive data for better code retrieval and reranking. In *The Thirteenth International Conference on Learning Representations*.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, and 5 others. 2025a. OpenHands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*.

Zora Zhiruo Wang, Akari Asai, Xinyan Velocity Yu, Frank F. Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. 2025b. CodeRAG-Bench: Can retrieval augment code generation? In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 3199–3214, Albuquerque, New Mexico. Association for Computational Linguistics.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2025. Demystifying LLM-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE):801–824.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Z.AI. 2025. GLM-4.7. https://huggingface.co/zai-org/GLM-4.7. Model card. Accessed: 2026-05-25.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. RepoCoder: Repository-level code completion through iterative retrieval and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2471–2484, Singapore. Association for Computational Linguistics.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. AutoCodeRover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, pages 1592–1604. Association for Computing Machinery.

A Parameter Sensitivity Analysis

The hybrid scoring rule uses a weight α to balance peak evidence from the highest-scoring chunk ($s_{f,(1)}$) against consistency across multiple retrieved chunks. To verify that our results are not sensitive to this choice, we perform a lightweight offline sensitivity analysis: for each of 3 repository-level splits, we reuse the cached dense index and chunk-level scores, recompute file-level rankings for $\alpha \in \{0.0, 0.1, \ldots, 1.0\}$, and re-evaluate file retrieval metrics without rerunning the downstream agent.

Table 4: File-level retrieval metrics as a function of α, aggregated over 3 repository-level development splits. Performance is stable across the full range: Acc@1 and MRR peak near α=0.5, while Hit@10 and Recall@10 plateau for $\alpha \geq 0.7$. We use α=0.5 in all reported experiments.

α	Acc@1	Hit@10	Recall@10	MRR
0.0 (Avg)	0.4965	0.8056	0.7648	0.6077
0.1	0.5035	0.8056	0.7648	0.6125
0.2	0.5069	0.8056	0.7648	0.6136
0.3	0.5208	0.8056	0.7648	0.6231
0.4	0.5174	0.8056	0.7648	0.6222
0.5	**0.5243**	0.8056	0.7648	**0.6251**
0.6	0.5174	0.8090	0.7671	0.6229
0.7	0.5139	**0.8125**	**0.7679**	0.6212
0.8	0.5069	**0.8125**	**0.7679**	0.6155
0.9	0.5104	**0.8125**	**0.7679**	0.6168
1.0 (Max)	0.5139	**0.8125**	**0.7679**	0.6168

As shown in Table 4, all metrics vary within a narrow band across the full α range. Ranking-oriented metrics (Acc@1, MRR) peak near α=0.5, while recall metrics reach a broad plateau for $\alpha \geq 0.7$. The two extremes, pure averaging (α=0) and pure max pooling (α=1), are both competitive, demonstrating that the hybrid formulation is robust rather than requiring careful tuning. We fix α=0.5 throughout all experiments reported in this paper.

B Retrieval Configuration and Baseline Performance

Our dense index follows the function-level chunking protocol of Chen et al. (2025): each top-level function or method definition is treated as a single chunk and embedded using CodeRankEmbed (Suresh et al., 2025). During the agent loop, the agent issues free-form natural language search queries; the retrieval backend returns the top-ranked chunks by cosine similarity. RepoNav's file-level aggregation rule and the snippet baseline operate on the same retrieved chunks; only the post-retrieval presentation differs.

Table 5 reports the retrieval-only accuracy of the shared dense index, evaluated without any downstream agent. These numbers provide a retrieval-only reference point for the fixed dense index before downstream agent interaction. The file-to-function localization gap is already visible at the retrieval-only stage: File Acc@5 reaches 0.721, whereas Function Acc@5 is only 0.348, which is less than half. Since Snippet Search and RepoNav use the same fixed dense index and raw chunk-level scores, this result supports our interpretation that RepoNav's downstream gains arise from post-retrieval evidence organization and navigation rather than from changes to the underlying retriever.

Table 5: Retrieval-only performance of the dense index used by all agent experiments. No downstream agent is involved; scores reflect pure embedding-based retrieval.

	File			Module			Function	
	@1	@3	@5	@5	@10	@15	@5	@10
Acc	0.505	0.661	0.721	0.521	0.604	0.666	0.348	0.430

C Full LocBench Results

Table 6 provides the complete LocBench localization results across all seven models, four exploration settings, and the reported Accuracy@k and Recall@k variants. The main text (Table 1) reports File Acc@5, Module Acc@5, Function Acc@5, and Function Rec@10; this table additionally includes lower- and higher-rank Accuracy@k and Recall@k variants where applicable, enabling fine-grained comparison across the reported k values.

Several patterns emerge from the complete results that are not visible in the summary table. First, RepoNav's gains are consistent across k values: improvements at Acc@1 tend to be slightly smaller than those at larger k, suggesting that RepoNav helps agents recover more of the gold target set within the candidate budget rather than always ranking all required targets first. Second, the recall-level gains are generally larger than the accuracy-level gains at comparable k, reflecting that RepoNav helps agents localize a greater fraction of the total gold targets per instance. Third, the file-level recall columns show that RepoNav largely preserves file-level coverage while improving module- and function-level localization, although small drops appear for some models.

Table 6: Complete LocBench localization results with all @k variants. This table supplements Table 1 with additional Acc@k and Recall@k values not shown in the main text. Bold indicates the best setting within each model block.

Model	Setting	File Acc			Module Acc				Function Acc				File Recall			Module Recall			Function Recall		
		@1	@3	@5	@1	@3	@5	@10	@1	@3	@5	@10	@3	@5	@10	@3	@5	@10	@3	@5	@10
Qwen2.5-72B	Bash	0.4714	0.5679	0.5893	0.3429	0.3661	0.3964	0.4214	0.2571	0.2339	0.2536	0.2643	0.5929	0.6206	0.6295	0.4048	0.4345	0.4622	0.2756	0.2946	0.3067
	Snippet Search	0.6304	0.6607	0.6804	0.4357	0.4446	0.4607	0.4696	0.3196	0.2821	0.3000	0.3089	0.6985	0.7179	0.7223	0.4943	0.5176	0.5272	0.3482	0.3663	0.3771
	Snippet+ListSym	0.6464	0.6696	0.6732	0.5229	0.5139	0.5621	0.5386	0.3886	0.3629	0.3843	0.4064	0.7063	0.7106	0.7112	0.5405	0.5785	0.5886	0.4193	0.4436	0.4901
	RepoNav	**0.6661**	**0.6911**	**0.6982**	**0.5411**	**0.5375**	**0.5696**	**0.5929**	**0.4071**	**0.3946**	**0.4339**	**0.4786**	**0.7301**	**0.7382**	**0.7382**	**0.5914**	**0.6297**	**0.6488**	**0.4634**	**0.5113**	**0.5492**
GPT-OSS-120B	Bash	0.6179	0.6875	0.7054	0.4679	0.4643	0.5000	0.5196	0.3643	0.3214	0.3393	0.3536	0.7223	0.7397	0.7457	0.5167	0.5549	0.5742	0.3798	0.4007	0.4171
	Snippet Search	0.7125	0.7518	0.7589	0.5964	0.5786	0.6125	0.6357	0.4929	0.4393	0.4643	0.4893	0.7887	0.7967	0.7976	0.6372	0.6748	0.6936	0.5226	0.5462	0.5707
	Snippet+ListSym	**0.7268**	0.7500	0.7589	0.5929	0.6007	0.6304	0.6429	0.4911	0.4521	0.4875	0.5089	0.7863	0.7967	0.7976	0.6567	0.6815	0.7020	0.5409	0.5663	0.5858
	RepoNav	0.7250	**0.7589**	**0.7696**	**0.5982**	**0.6071**	**0.6500**	**0.6625**	**0.5064**	**0.4868**	**0.5271**	**0.5486**	**0.7964**	**0.8056**	**0.8074**	**0.6810**	**0.7163**	**0.7269**	**0.5636**	**0.5990**	**0.6184**
Qwen3-Next-80B	Bash	0.6482	0.7107	0.7339	0.4804	0.4929	0.5232	0.5464	0.3696	0.3321	0.3393	0.3500	0.7430	0.7668	0.7760	0.5352	0.5701	0.5936	0.3751	0.3944	0.4098
	Snippet Search	0.7179	0.7286	0.7321	0.5536	0.5589	0.5679	0.5804	0.4482	0.4054	0.4107	0.4268	0.7698	0.7764	0.7764	0.6062	0.6346	0.6520	0.4634	0.4957	0.5170
	Snippet+ListSym	0.7103	0.7304	0.7329	0.5807	0.5818	0.5886	0.6464	0.4986	0.4557	0.4539	0.4718	0.7380	0.7535	0.7557	0.6200	0.6542	0.6768	0.5010	0.5309	0.5636
	RepoNav	**0.7339**	**0.7357**	**0.7393**	**0.6018**	**0.6018**	**0.6321**	**0.6482**	**0.5146**	**0.4893**	**0.4982**	**0.5268**	**0.7770**	**0.7814**	**0.7819**	**0.6562**	**0.7000**	**0.7212**	**0.5355**	**0.5774**	**0.6215**
Qwen3-Coder-30B	Bash	0.6429	0.6857	0.7000	0.5393	0.5464	0.5643	0.5732	0.4571	0.4089	0.4196	0.4232	0.7215	0.7375	0.7396	0.5834	0.6186	0.6322	0.4523	0.4858	0.4982
	Snippet Search	0.6786	0.6893	0.6929	0.5679	0.5571	0.5839	0.5875	0.4804	0.4464	0.4589	0.4679	0.7225	0.7282	0.7309	0.6035	0.6429	0.6537	0.4972	0.5321	0.5507
	Snippet+ListSym	0.6804	0.6964	0.7071	0.5857	0.5768	0.5982	0.6036	0.4964	0.4643	0.4757	0.4982	0.7326	0.7436	0.7454	0.6235	0.6577	0.6663	0.5231	0.5607	0.5717
	RepoNav	**0.6946**	**0.7196**	**0.7268**	**0.6036**	**0.6036**	**0.6286**	**0.6357**	**0.5054**	**0.4893**	**0.5161**	**0.5393**	**0.7555**	**0.7623**	**0.7623**	**0.6424**	**0.6887**	**0.6997**	**0.5303**	**0.5887**	**0.6180**
MiniMax-M2.5	Bash	0.7354	0.7425	0.7604	0.6193	0.6264	0.6604	0.6729	0.5443	0.5086	0.5246	0.5407	0.7771	0.7948	0.8098	0.6669	0.7141	0.7347	0.5525	0.6038	0.6328
	Snippet Search	0.7393	0.7518	0.7643	0.6321	0.6429	0.6714	0.6857	0.5500	0.5268	0.5321	0.5482	0.7822	0.7963	0.7996	0.6847	0.7298	0.7485	0.5722	0.6157	0.6409
	Snippet+ListSym	0.7589	0.7732	0.7804	0.6482	0.6411	0.6714	0.6893	0.5696	0.5339	0.5339	0.5589	0.8093	0.8199	0.8235	0.6858	0.7345	0.7566	0.5712	0.6229	0.6519
	RepoNav	**0.7736**	**0.7843**	**0.8032**	**0.6532**	**0.6586**	**0.6907**	**0.6996**	**0.5882**	**0.5921**	**0.6011**	**0.6136**	**0.8135**	**0.8347**	**0.8471**	**0.7073**	**0.7449**	**0.7696**	**0.6279**	**0.6869**	**0.7084**
GLM-4.7	Bash	0.7568	0.7800	0.8079	0.6496	0.6425	0.6854	0.7264	0.5782	0.5639	0.5854	0.6300	0.8031	0.8207	0.8260	0.6840	0.7324	0.7763	0.5987	0.6556	0.7137
	Snippet Search	0.7875	0.8036	0.8214	0.6768	0.6625	0.6982	0.7268	0.6054	0.5875	0.6000	0.6357	0.8405	0.8554	0.8634	0.7044	0.7595	0.7907	0.6224	0.6765	0.7223
	Snippet+ListSym	0.7857	0.8036	0.8125	0.6607	0.6786	0.7089	0.7304	0.5804	0.5607	0.5768	0.6036	0.8396	0.8470	0.8506	0.7197	0.7665	0.7896	0.6007	0.6582	0.6950
	RepoNav	**0.7886**	**0.8064**	**0.8389**	**0.6950**	**0.6968**	**0.7336**	**0.7586**	**0.6343**	**0.5950**	**0.6318**	**0.6771**	**0.8525**	**0.8765**	**0.8889**	**0.7683**	**0.8144**	**0.8200**	**0.6681**	**0.7509**	**0.7826**
Gemini-3-Flash	Bash	0.8089	**0.8250**	0.8357	0.7036	0.6982	0.7089	0.7196	0.6286	0.6054	0.6250	0.6464	**0.8637**	**0.8743**	**0.8761**	0.7434	0.7731	0.7868	0.6558	0.7041	0.7365
	Snippet Search	0.8161	0.8143	0.8232	0.7089	0.6857	0.7018	0.7018	0.6357	0.6089	0.6232	0.6304	0.8526	0.8612	0.8612	0.7337	0.7654	0.7712	0.6558	0.7028	0.7223
	Snippet+ListSym	0.8078	0.8115	0.8221	0.7154	0.7026	0.7165	0.7242	0.6363	0.6327	0.6481	0.6519	0.8372	0.8465	0.8583	0.7296	0.7702	0.8015	0.6624	0.7151	0.7387
	RepoNav	**0.8196**	0.8207	**0.8404**	**0.7536**	**0.7393**	**0.7518**	**0.7625**	**0.6857**	**0.6750**	**0.6893**	**0.7125**	0.8603	0.8691	0.8700	**0.7860**	**0.8140**	**0.8252**	**0.7105**	**0.7666**	**0.8026**

D RQ2 Evidence Funnel

To complement the behavioral analysis in Section 4.3, we report three diagnostic trajectory statistics: whether the gold file becomes available to the agent either through the retrieved candidate set or through subsequent shell exploration (*Coverage*), whether the agent inspects it given coverage (*Inspection*), and whether inspection leads to correct localization (*Resolution*). These statistics are coarse trajectory diagnostics rather than mutually exhaustive paths through the agent workflow, and should not be interpreted as a multiplicative decomposition of the endpoint localization metrics in Table 2.

Table 7: Evidence funnel across the three RQ2 interface variants. Each stage conditions on the previous one and should be read as a diagnostic decomposition, not as a multiplicative estimate of endpoint localization performance. All three settings share the same dense retrieval backend; Coverage can still vary because agents may discover files through shell exploration after the initial retrieval output.

Setting	Coverage	Inspection	Resolution
File-Only	78.26%	97.22%	92.50%
Inline Scaffold	80.43%	**97.30%**	92.68%
Tree Scaffold	**82.61%**	92.11%	**97.50%**

As Table 7 shows, Inline Scaffold and Tree Scaffold exhibit different diagnostic profiles. Inline Scaffold attains the highest inspection rate (97.30%), consistent with an information-heavy interface that encourages the agent to read broadly. Tree Scaffold leads to more selective inspection (92.11%) but achieves the highest resolution rate once the gold file is inspected (97.50%), consistent with the more frequent `list_symbols` usage and fewer wasted steps reported in Table 2.

The key difference between these two behavioral profiles can be summarized as follows. Inline Scaffold maximizes the probability of *looking at* the gold file: by embedding rich structural detail directly in the retrieval output, it lowers the cost of passive browsing. Tree Scaffold instead maximizes the probability of *correctly acting on* the gold file once inspected: by presenting compact cues with explicit next-step prompts, it encourages the agent to actively verify candidates through tool use rather than relying on in-context information alone. This decomposition reinforces the conclusion that Tree Scaffold's advantage lies in the quality of post-inspection navigation rather than the breadth of initial coverage.

E SWE-QA Full Results

E.1 Dimension-Level Scores

Table 8 presents the full dimension-level SWE-QA-Bench results, complementing the aggregate total scores reported in Table 3. Each answer is independently scored on a 20-point scale across five dimensions: Correctness, Completeness, Relevance, Clarity, and Reasoning.

Across all four models, RepoNav achieves the

highest score on every individual dimension. The gains are most pronounced in Correctness and Completeness, consistent with the hypothesis that structured navigation helps agents find and synthesize the right evidence rather than merely improving surface-level answer quality. Relevance, Clarity, and Reasoning show smaller but consistently positive improvements, indicating that better evidence navigation has a downstream effect on overall answer coherence.

Table 8: SWE-QA-Bench dimension-level results. Results are reported on the per-model common subset successfully scored for all three methods. Each dimension is scored on a 20-point scale; Total is their sum (max 100). Dimensions: Correctness (Corr.), Completeness (Comp.), Relevance (Rel.), Clarity (Clar.), and Reasoning (Reas.).

Model	Method	Corr.	Comp.	Rel.	Clar.	Reas.	Total
GPT-OSS-120B	Bash	14.06	11.68	17.55	16.54	16.29	76.12
	Snippet Search	14.46	13.20	18.08	16.72	16.51	78.97
	RepoNav	**14.84**	**13.66**	**18.35**	**16.82**	**16.66**	**80.33**
Qwen3-Next-80B	Bash	12.44	10.52	17.91	15.71	15.14	71.72
	Snippet Search	13.22	11.78	17.96	15.88	15.47	74.31
	RepoNav	**13.63**	**12.32**	**18.07**	**16.31**	**15.96**	**76.29**
GLM-4.7	Bash	16.11	15.96	17.93	17.50	17.46	84.96
	Snippet Search	16.05	15.96	17.81	17.50	17.43	84.75
	RepoNav	**16.69**	**16.97**	**18.08**	**17.63**	**17.47**	**86.84**
Gemini-3-Flash	Bash	16.39	16.16	18.17	17.91	17.86	86.53
	Snippet Search	16.35	16.02	18.11	17.90	17.87	86.55
	RepoNav	**17.04**	**17.13**	**18.21**	**17.93**	**17.93**	**88.14**

E.2 Question-Type Breakdown

Figure 7 breaks down SWE-QA-Bench performance by question type (*How*, *What*, *Where*, *Why*) across all four evaluated models. The analysis complements the aggregate results in Table 3 by showing that RepoNav's gains are not confined to a single question category.

Score improvements are observed across all types, with the largest and most consistent gains on *Why* questions. This is consistent with the localization findings: *Why* questions typically require synthesizing evidence across multiple files and tracing causal chains through the codebase—precisely the scenario where structured navigation helps agents avoid premature anchoring on a single entry point.

Interaction savings (measured in steps) are substantial for three of four models. GPT-OSS-120B shows smaller and mixed savings, suggesting that efficiency gains may depend on model-specific exploration behavior rather than model strength alone. The explicit continuation cues in RepoNav therefore appear to interact with each model's default exploration policy.

F Additional Case Studies

We provide additional case studies illustrating navigation regimes beyond those highlighted in the main text. Figure 2 illustrates the core same-file disambiguation mechanism; below we describe a cross-file pivoting case and a remaining failure boundary.

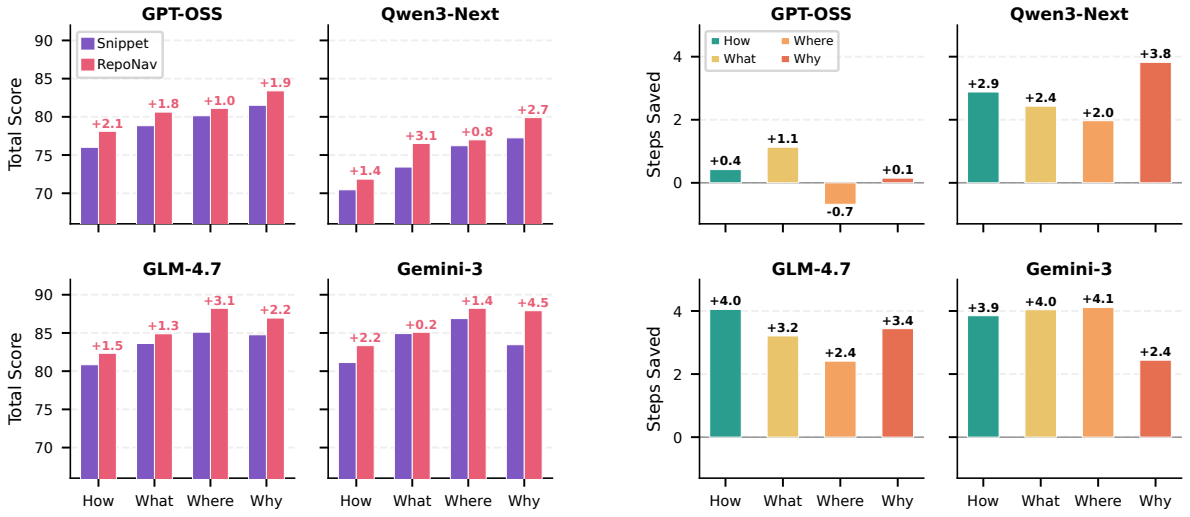
Cross-file pivoting (LocBench).
In `DS4SD__docling-314`, the visible symptom appears near Markdown output code, while the gold target lies in an upstream implementation file, `msword_backend.py`. Snippet-style search keeps the agent near the output-side sink, since the highest-scoring chunks come from Markdown-related code that shares surface keywords with the issue. RepoNav retains the upstream implementation file in its candidate structure because its hybrid file-level scoring rule rewards files supported by multiple moderately scored chunks, rather than only by a single dominant match. The tree scaffold then exposes both symptom-side and source-side files in a navigable list, enabling the agent to pivot from output-related code to the gold implementation target.

Remaining failure boundary (LocBench).
In `yt-dlp__yt-dlp-11615`, RepoNav successfully brings the agent to the correct file, but the agent still stops at higher-level wrapper methods rather than drilling down to the gold targets, which are lower-level helper functions called by the wrapper. The `[GLIMPSE]` block lists these helpers, but the agent does not inspect them further, treating the wrapper as a sufficient answer. This suggests that RepoNav can reduce wrong-file fixation, but deep same-file evidence harvesting remains an open problem, particularly when the gold target is multiple call-hops away from the most salient entry point.

G RepoNav Scaffold Specification and Output Format

G.1 Serialization Budgets and Ordering

RepoNav uses fixed budgets and deterministic ordering rules to keep the scaffold compact and reproducible. For each candidate file, `[ANCHORS]` includes at most two symbols. Anchors are selected by case-insensitive substring matching between query tokens and symbol names, and ranked by token overlap, symbol-kind priority, and span length.



(a) Score by question type.

(b) Interaction savings by question type.

Figure 7: SWE-QA-Bench question-type breakdown across four models. (a) Score improvements span all question types, with the largest gains on *Why* questions. (b) Interaction savings are consistent for three of four models; GPT-OSS-120B shows mixed results. Model names are abbreviated for space; see Section 4.1 for full names.

`[GLIMPSE]` includes up to three non-anchor symbols per file, selected by query-aware ranking while preserving symbol-kind diversity. If the top-ranked file has no anchor match, this budget is increased to five to expose a broader file sketch.

`[CANDIDATE_TARGETS]` is capped at four entries. Candidates are ordered first by anchors, then by same-file call-neighborhood symbols of anchors, and finally by selected glimpse symbols. A fixed continuation cue is appended whenever a candidate block is present.

Call context is name-only: it records same-file caller and callee symbol names without function bodies, arguments, or interprocedural analysis. By default, RepoNav parses the top five candidate files and auto-expands full three-block scaffolds for the top three.

G.2 Serialized Output Example

Figure 8 shows a truncated example of RepoNav's serialized output as presented to the agent. The indentation-based tree provides file-centered organization without requiring any specialized query language or structured API from the agent.

The three blocks—`[ANCHORS]`, `[GLIMPSE]`, and `[CANDIDATE_TARGETS]`—correspond to the design principles described in Section 3.1. The anchor block provides grounded entry points with call-context annotations (`invokes ->` and `invoked-by <-`). The glimpse block exposes non-anchor symbols as a structural sketch, preventing fixation on anchors alone. The candidate targets block aggregates actionable options and ends with an explicit continuation cue (» `Next: run` `list_symbols` `on this file to inspect sibling symbols.`), lowering the cost of continued exploration.

```
[DIR] src/backend/
  [FILE] backend/server.py (evidence: 3)
     |-- imports-by <- backend/app.py
     |-- [ANCHORS]
     |    `-- oauth_callback(function)[L120-L156]
     |          invokes -> validate_token, init_app
     |-- [GLIMPSE]
     |    `-- class HTTPServer:
     |    `-- def handle_request():
     `-- [CANDIDATE_TARGETS]
         - backend/server.py:oauth_callback
         - backend/server.py:validate_token
         >> Next: run `list_symbols` on this file to
            inspect sibling symbols.
```

Figure 8: A truncated example of RepoNav's serialized output. The indentation-based tree provides file-centered organization, a compact structural sketch, and explicit continuation cues. The plain-text format requires no specialized query language from the agent.