INTERACTIVE EXTERNAL KNOWLEDGE ACQUISITION FOR SOFTWARE ENGINEERING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Repository-level software engineering agents can inspect code and execute tests, yet useful evidence may lie outside their execution environment. Search and browsing tools expand access, but agents must decide when to retrieve, communicate local constraints, and assess external information's applicability. We present BRIDGE, a framework that separates repository-side problem solving from external evidence acquisition. A dedicated retriever provides exploratory and focused interfaces; during focused retrieval, it can request missing repository context and resume its search. An outcome-conditioned prompt-memory procedure helps a parameter-frozen agent use these interfaces, with memory constructed on a disjoint bootstrap set and fixed before evaluation. We evaluate this organization through local-only, direct-tools, and retriever-capacity comparisons. On 450 held-out SWE-bench Verified instances, BRIDGE with GPT-5-mini in both roles achieves $59.6 \pm 1.37\%$, compared with $54.2 \pm 1.21\%$ for direct access to the same web tools with the same adaptation (three-run mean $\pm$ standard deviation). Protocol and adaptation ablations, temporal-policy sensitivity experiments, and cost measurements characterize the design's behavior and computational trade-offs. These experiments show how coordinated evidence acquisition can complement repository-local reasoning.

1 INTRODUCTION

Repository-level software engineering agents solve issues by inspecting code, editing files, and checking candidate changes with execution feedback. SWE-bench has established real-world GitHub issue resolution as a standard setting for studying these capabilities (Jimenez et al., 2024). Recent systems improve the agent's interaction with the repository through specialized interfaces, inference-time search, repository graphs, general agent platforms, and structured repair pipelines (Yang et al., 2024; Antoniades et al., 2025; Ouyang et al., 2025; Wang et al., 2025; Xia et al., 2025). These advances organize the information available within a software engineering environment. Some repair decisions also depend on evidence beyond that environment: an API's behavior in a particular version, a library convention, or a mathematical identity needed by an implementation. Documentation, version histories, and API references can supply information relevant to such decisions (Xavier et al., 2017; Treude & Robillard, 2016; Ajam et al., 2020).

We define the information boundary operationally. *Internal knowledge* comprises the model's parametric knowledge and information directly accessible in the execution environment, including the issue description, checked-out repository, local documentation, dependencies, and execution feedback. *External knowledge* is information acquired from outside that boundary during problem solving. The distinction is about access rather than topic: the same documentation is internal when present locally and external when retrieved from a remote source. The repair objective is unchanged. External evidence is an input to repository-side reasoning, and the agent remains responsible for implementing and checking a candidate patch.

Access alone does not determine how that evidence should enter the repair process. A direct approach equips the repository agent with search and browsing tools (Soni et al., 2026). The agent must then decide when to retrieve, formulate a query, select and navigate sources, extract an answer, and relate it to the local task (He et al., 2024). Delegation offers another organization of these operations, as in broader multi-agent systems (Fourney et al., 2024), but introduces a complementary difficulty: the retriever can explore external sources without observing the repository agent's full debugging state.

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.

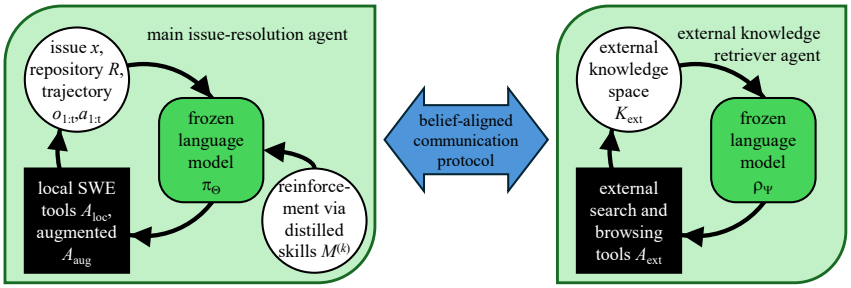


Figure 1: Overview of BRIDGE framework, including communication protocol and reinforcement.

A topically relevant answer can be inapplicable if the request omits a dependency version, stack trace, or local configuration. We therefore study external knowledge acquisition as a coupled problem of *tool-use decisions* and *communication under asymmetric observations*.

We introduce BRIDGE, **B**elief-aligned **R**etrieval via **I**nteractive **D**ialogue with **G**rounded **E**vidence, a framework that pairs a repository-level agent with a dedicated Internet-facing retriever (Figure 1). The repository agent retains its local inspection, editing, and validation loop, while the retriever manages external search and browsing and returns compact evidence. An exploratory interface supplies candidate answers and sources; a focused interface addresses a more specific knowledge need. During focused retrieval, the retriever can issue a protocol-level `WAIT` request for missing repository context. The main agent supplies that context, and the retriever resumes its search. This handoff supports both local clarification and continuation across retrieval calls. Here, belief alignment means sharing the current hypotheses and constraints needed to make the retrieved evidence applicable to the repair decision.

A new interface also changes the decisions expected of the repository agent. It must learn when external acquisition is useful, what local information to communicate, and how to incorporate the answer into debugging. BRIDGE uses an *outcome-conditioned in-context reinforcement* procedure to construct reusable tool-use lessons for these decisions without updating model parameters. On a bootstrap partition, the procedure contrasts successful forced-retrieval trajectories with failed free-run trajectories, distills their tool-use differences, and refines a compact prompt memory. The memory is fixed before evaluation, and bootstrap tasks are excluded from the held-out evaluation set. The result is a coordinated evidence-acquisition interface together with prompt-level adaptation for using it.

Our evaluation makes direct web access a central comparison. The direct-tools agent operates the same web actions itself; BRIDGE delegates them to the retriever and its interaction protocol. The compared settings share the GPT-5-mini main model, held-out task split, outer interaction budget, and outcome-conditioned adaptation protocol. On the 450 held-out SWE-bench Verified instances, BRIDGE with GPT-5-mini in both roles achieves $59.6 \pm 1.37\%$, versus $54.2 \pm 1.21\%$ for direct tools (three-run mean $\pm$ standard deviation). This 5.4-percentage-point comparison evaluates the complete retrieval organization under matched model capability and tool availability. The original GPT-5-retriever configuration achieves $59.4 \pm 1.32\%$, placing helper capacity alongside interface organization in the empirical study.

The experiments further examine GPT-5-mini and Gemini-3-Flash main-agent configurations on SWE-bench Verified and SWE-rebench, adaptation and no-`WAIT` variants, and a stricter temporal retrieval policy. The no-`WAIT` comparison retains retrieval and adaptation while disabling the clarification/resumption handoff. The temporal comparison evaluates sensitivity to LLM-mediated source-date filtering. We also report the quality–cost trade-off: in the original GPT-5-mini main/GPT-5 retriever configuration, total reported tokens are approximately $1.9\times$ the local-only total, while the average number of repository interaction steps is 16.3% lower. These complementary comparisons connect the design of external-information interfaces to repair outcomes, interaction behavior, and computational cost.

Our contributions are:

- **Coordinated evidence acquisition.** We formulate the information-access boundary and organize external acquisition around a dedicated retriever with exploratory and focused interfaces, while retaining the repository agent's local repair loop.

- **Interactive context exchange.** We introduce a multi-round protocol through which the retriever requests repository-side constraints and resumes external acquisition, connecting the agents' asymmetric observations.
- **Parameter-frozen tool-use adaptation.** We construct and refine outcome-conditioned prompt memory on a disjoint bootstrap partition to guide the agent's retrieval decisions and evidence use.
- **Controlled empirical evaluation.** We compare the complete framework with direct web-tool access, test matched-capability and stronger-retriever configurations, and examine protocol, adaptation, retrieval-policy, and computational trade-offs in repository-level issue resolution.

2 Problem Setup

2.1 Repository-Level SWE Agents with Mixed Information

We study repository-level issue resolution in the SWE-bench-style setting (Jimenez et al., 2024; Yang et al., 2024; Antoniades et al., 2025). A task is $\tau = (x, \mathcal{R}_0, \mathcal{V})$, comprising an issue description, an initial repository, and a benchmark validation procedure. The agent produces a patch, and $\mathcal{V}$ determines its evaluation outcome. Hidden evaluation tests are not available during inference; these are distinct from the local tests and execution feedback that the agent can access while developing the patch. At step t, its local trajectory s_t contains the issue, current repository, and previous actions and observations. Local actions $\mathcal{A}_{\text{loc}}$ include inspection, code search, editing, command execution, and testing. Restricting actions to this interface does not imply that local or parametric knowledge is sufficient for every task.

Information boundary. We define *internal knowledge* by access: the model's parametric knowledge and information directly accessible in the benchmark environment, including repository files, local documentation, dependencies, and available test feedback. A document already present in that environment is therefore internal. *External knowledge*, $\mathcal{K}_{\text{ext}}$, lies beyond this boundary; our implementation accesses it through public web sources such as API references, release notes, and technical documentation. The repair objective and evaluation remain unchanged, while information-acquisition actions expand. This access boundary is separate from evidence eligibility: relevant technical information may aid repair, whereas target solutions or post-problem information can compromise benchmark validity. We evaluate retrieval restrictions and their limitations separately.

2.2 External Knowledge Retrieval under Asymmetric Observations

BRIDGE delegates web search and browsing to an *External Knowledge Retriever* ρ_ψ (Figure 1). The main agent sends a message m_t containing its information need and selected local context; the retriever accesses $\mathcal{K}_{\text{ext}}$ and returns evidence e_t. The augmented interface is

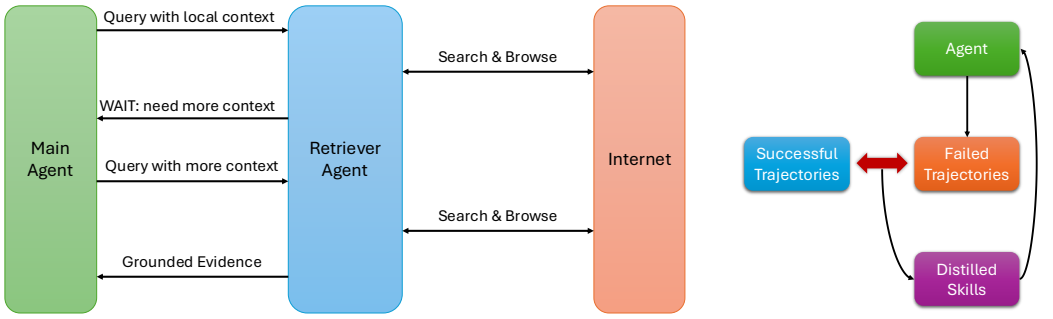
$$\mathcal{A}_{\text{aug}} = \mathcal{A}_{\text{loc}} \cup \{\textsc{RequestExternal}(m_t)\}. \tag{1}$$

The policy $a_t \sim \pi_\theta(\cdot \mid s_t, e_{1:t})$ selects from this interface, with $e_{1:t}$ denoting evidence returned so far. The main agent observes the repository trajectory and received evidence; the retriever observes web sources and communicated context, not the full repository state. Missing dependency versions, stack traces, or configuration can therefore produce topically relevant but inapplicable answers.

This interface couples two decisions: *when to retrieve* and *what local context to communicate*. Merely exposing a new tool does not determine either decision: an agent may omit useful retrieval or search unnecessarily. The framework below addresses tool-use adaptation and communication across this observation boundary.

3 BRIDGE Framework

BRIDGE combines a dedicated external knowledge retriever, multi-round communication of local constraints, and outcome-conditioned in-context tool-use memory. The main agent retains responsibility for repository inspection, implementation, and local validation; the retriever manages web-facing evidence acquisition.



(a) Multi-round belief-aligned communication.

(b) Outcome-conditioned in-context reinforcement.

Figure 2: Two mechanisms in BRIDGE. **(a)** A focused retriever can WAIT for local context or continuation, then resume from the returned URL. **(b)** Contrasting bootstrap trajectories yields reusable tool-use lessons for future prompts. Model parameters remain fixed.

3.1 EXTERNAL KNOWLEDGE RETRIEVER AGENT

The main agent specifies an information need and relevant local context through REQUESTEXTERNAL. The retriever ρ_ψ selects web sources dynamically rather than searching a fixed, pre-indexed corpus. It exposes two modes: OVERVIEWSEARCH returns candidate answers and URLs for exploratory questions; FOCUSEDRETRIEVAL investigates a concrete question, starting from an optional seed URL, and returns a synthesized answer with sources. The main agent can use overview results to formulate a focused request. Neither interface grants the retriever unrestricted repository access. Appendix A.1 specifies messages, outputs, and per-mode procedures. The evaluation applies source-eligibility and answer-leakage controls separately (Appendix B).

3.2 MULTI-ROUND BELIEF-ALIGNED COMMUNICATION

Focused retrieval can request the local information missing from its observations. Instead of returning an answer, the retriever may emit a protocol-level WAIT with a clarification question and resume URL. The main agent obtains the requested context from its trajectory or additional repository inspection, then re-invokes FOCUSEDRETRIEVAL with that context while retaining the URL. Examples include an installed package version, exact API usage, or stack trace. The retriever resumes with the added constraints rather than treating the initial query as complete.

WAIT also supports continuation when a promising search exceeds one call's interaction or time budget. The system retains the current URL and partially collected evidence and returns a continuation command. The main agent follows that command, adding local context when requested. A WAIT is not evidence or a final answer; it is distinct from browser-level `Wait`, which only waits for page state. Appendix A.2 details continuation and stopping rules, including unproductive repeated waits, and Appendix A.1 gives the execution pseudocode.

"Belief-aligned" describes communicating the main agent's current hypotheses and repository constraints, not a separately learned belief-state model. Returned evidence informs debugging rather than supplying an automatically accepted patch. The main agent checks applicability through repository inspection and available tests; this does not certify every retrieved claim (Appendix E).

3.3 OUTCOME-CONDITIONED IN-CONTEXT REINFORCEMENT FOR TOOL USE

We adapt retrieval behavior through prompt memory while keeping all language-model parameters fixed. For each bootstrap task, we collect a *forced-retrieval* trajectory ξ_i^{force}, with explicit retrieval-use guidance, and a *free* trajectory ξ_i^{free}, which exposes the same protocol without that additional guidance. Repository validation supplies binary outcomes y_i^{force} and y_i^{free}. The initial contrastive set is

$$\mathcal{P}^{(0)} = \left\{ (\xi_i^{\text{force}}, \xi_i^{\text{free}}) \;\middle|\; y_i^{\text{force}} = 1,\ y_i^{\text{free}} = 0,\ \textsc{UsedExternal}(\xi_i^{\text{force}}) \right\}. \tag{2}$$

These outcome-conditioned pairs guide behavioral comparison; they do not by themselves isolate the causal effect of retrieval. A teacher distills differences in when to retrieve, how to formulate requests and communicate local constraints, and how to integrate evidence into repository reasoning. The resulting memory $\mathcal{M}^{(0)}$ is intended to encode reusable tool-use lessons rather than task-specific patches.

We prepend the memory to the main-agent prompt and rerun the bootstrap tasks with the full protocol. Successful forced trajectories are compared with unsuccessful adapted trajectories to distill additional lessons. The update $\mathcal{M}^{(k+1)} = \textsc{Compress}(\mathcal{M}^{(k)} \cup \Delta\mathcal{M}^{(k)})$ removes redundancy within a fixed context budget. Refinement continues until bootstrap validation performance or tool-use behavior stabilizes; Appendix A.3 provides the full construction and refinement algorithm.

In the GPT-5-mini experiments, the teacher is also GPT-5-mini. We randomly select 50 SWE-bench Verified instances for bootstrapping and exclude all 50 from the 450-instance evaluation. Only bootstrap outcome feedback is used to construct the memory, which is fixed before evaluation. In-context reinforcement here denotes feedback-driven construction of prompt memory, not a parameter update.

4 Experiments

We design experiments to answer five questions: **(1)** Does adding BRIDGE to existing repository-level software engineering agents improve issue-resolution performance on standard and contamination-resistant benchmarks? **(2)** Does the benefit of BRIDGE extend beyond giving the agent direct access to the same web tools? **(3)** Which components of BRIDGE, including the external retriever, multi-round belief-aligned communication, and outcome-conditioned in-context reinforcement, are responsible for the final performance? **(4)** Is the improvement explained by answer exposure through live retrieval? **(5)** What is the computational trade-off introduced by interactive external knowledge acquisition, in terms of token consumption and trajectory length?

4.1 Experimental Setup

Datasets. To evaluate BRIDGE, we conduct experiments on two repository-level software engineering benchmarks. We primarily evaluate on **SWE-bench Verified** (Jimenez et al., 2024), a widely adopted benchmark for real-world repository-level issue resolution. It contains human-verified bug-fixing instances with reliable validation signals, and we use it as our main evaluation benchmark. In addition, we evaluate on **SWE-rebench** (Badertdinov et al., 2025), a recently introduced contamination-resistant benchmark for software engineering agents. Unlike static benchmarks, SWE-rebench constructs fresh interactive tasks from real-world repositories to reduce contamination and distribution shift. We use the same fixed SWE-rebench subset for all reported comparisons to evaluate whether BRIDGE remains effective under a more dynamic and contamination-resistant setting.

Implementation Details. We instantiate the main repository-level agent with mini-SWE-agent and augment it with BRIDGE. Our default configuration uses `gpt-5-mini` (OpenAI, 2025) as both the main agent and the external retriever. The component ablations use this default pairing; the temporal-policy comparison retains the original GPT-5 retriever. To examine BRIDGE across different base models, we additionally instantiate the main agent with `gemini-3.0-flash` (Google DeepMind, 2025), retaining the original GPT-5 retriever and the same retrieval interface and communication protocol. The direct-tools comparison uses GPT-5-mini in both roles (Table 2); Appendix A.5 compares GPT-5-mini with the original GPT-5 retriever. We additionally evaluate Claude Haiku 4.5 as the main-agent backbone for both the local-only baseline and BRIDGE; the Claude BRIDGE configuration uses GPT-5-mini as its external retriever.

For BRIDGE's adaptation phase, we construct an in-context reinforcement buffer using 50 randomly selected instances from SWE-bench Verified. On these instances, we run forced-retrieval trajectories, where the agent is encouraged to use the external retriever, and compare them with free trajectories, where the agent has access to the same protocol but receives no additional retrieval-use guidance. We then distill the behavioral differences between successful forced-retrieval runs and failed free runs into compact in-context examples. These 50 instances are used only for constructing the adaptation

buffer and are excluded from the main SWE-bench Verified evaluation. We report final results on the remaining 450 SWE-bench Verified instances and on SWE-rebench.

Additional controls and uncertainty. The repeated-run GPT-5-mini and Gemini experiments, direct-tools controls, and component ablations report three-run means and standard deviations on both benchmarks. The retriever-capacity comparison likewise covers both benchmarks; the temporal-policy comparison uses three runs on the held-out 450 SWE-bench Verified instances. The distillation teacher in the GPT-5-mini experiments is GPT-5-mini, and the memory is fixed before evaluation. SWE-rebench's design addresses training-data contamination; it does not by itself rule out answer exposure through live retrieval, which we examine separately below. Appendix A provides the detailed protocol.

4.2 Main Results

Table 1: **Main results on SWE-bench Verified and SWE-rebench.** BRIDGE improves Pass@1 across GPT-5-mini, Gemini-3-Flash, and Claude Haiku 4.5 backbones on both benchmarks, showing consistent gains from external knowledge acquisition. Values are Pass@1 percentages; uncertainty is reported where available.

Method	SWE-bench Verified	SWE-rebench
GPT-5-mini (local-only)	52.8 ± 1.69	42.8 ± 1.23
GPT-5-mini + BRIDGE	$\mathbf{59.4 \pm 1.32}$	$\mathbf{48.3 \pm 1.23}$
Gemini-3-Flash (local-only)	56.4 ± 1.61	43.2 ± 2.14
Gemini-3-Flash + BRIDGE	$\mathbf{63.8 \pm 1.25}$	$\mathbf{47.3 \pm 1.36}$
Claude-Haiku-4.5 (local-only)	58.3 ± 1.18	45.8 ± 1.52
Claude-Haiku-4.5 + BRIDGE	$\mathbf{62.1 \pm 1.37}$	$\mathbf{50.2 \pm 1.44}$

Table 1 reports the main issue-resolution results on SWE-bench Verified and SWE-rebench. Adding BRIDGE to the base agent consistently improves Pass@1 performance across the evaluated backbones and both benchmarks, with gains of +6.6 and +7.4 percentage points on SWE-bench Verified for GPT-5-mini and Gemini-3-Flash respectively, and smaller but still positive gains on the more contamination-resistant SWE-rebench. These results show that structured external knowledge acquisition reliably improves repository-level issue resolution across model families and evaluation settings. The observed SWE-rebench improvements are 5.5 points for GPT-5-mini and 4.1 points for Gemini-3-Flash.

4.3 Comparison with Direct Web-Tool Access

Table 2: **Comparison with direct web-tool access.** All settings use the GPT-5-mini main agent. The direct-tools baseline and BRIDGE share the same web actions and outcome-conditioned in-context adaptation; BRIDGE here uses GPT-5-mini as the retriever. Values are Pass@1 percentages (mean $\pm$ standard deviation).

Method	SWE-bench Verified	SWE-rebench
GPT-5-mini (local-only)	52.8 ± 1.69	42.8 ± 1.23
Direct web tools + adaptation	54.2 ± 1.21	42.8 ± 0.71
BRIDGE, GPT-5-mini retriever	$\mathbf{59.6 \pm 1.37}$	$\mathbf{48.1 \pm 2.14}$

Direct-tools control. The direct-tools baseline exposes the *same* web actions to the main agent, which controls both the repository environment and web tools without a separate retriever managing them. It uses the same GPT-5-mini main model, evaluation split, outer interaction budget, and outcome-conditioned in-context adaptation protocol as the corresponding BRIDGE comparison. The agent freely chooses whether and how often to browse. This matches tool access and adaptation, while realized computation may differ because tool use is agent-controlled.

BRIDGE achieves 59.6% issue resolution with GPT-5-mini in both roles, improving on local-only by 6.8 percentage points (Table 2). By comparison, direct access to the same web tools obtains 54.2%. The 5.4-point gain of same-model BRIDGE over direct tools, with outcome-conditioned adaptation in both settings, supports the practical value of a dedicated retriever and contextual communication beyond tool availability alone. On SWE-rebench, same-model BRIDGE achieves $48.1 \pm 2.14\%$,

compared with $42.8 \pm 0.71\%$ for direct tools and $42.8 \pm 1.23\%$ for local-only, a 5.3-point improvement over each. Appendix A.5 compares this configuration with the original GPT-5 retriever.

4.4 Ablation Studies

Table 3: **Ablation study of BRIDGE components.** The rows remove outcome-conditioned in-context reinforcement and then the whole external retriever agent. Full BRIDGE achieves the best performance, showing that both prompt-level tool-use adaptation and the interactive retriever protocol contribute to reliable external knowledge acquisition. The final no-`WAIT` row is a separate ablation from full BRIDGE, rather than an additional removal after the local-only setting. Values are Pass@1 percentages with standard deviations.

Method	SWE-bench Verified	SWE-rebench
GPT-5-mini + BRIDGE	59.6 ± 1.37	48.1 ± 2.14
w/o In-Context Reinforcement	55.8 ± 1.48	45.6 ± 1.41
w/o Retriever Agent	52.8 ± 1.69	42.8 ± 1.23
BRIDGE w/o protocol `WAIT`	56.7 ± 1.57	46.2 ± 1.38

We conduct ablation studies to understand the contribution of each core component in BRIDGE. Table 3 orders the variants from the full framework to progressively weaker configurations. The full BRIDGE approach includes the external retriever, the multi-round belief-aligned communication protocol, and outcome-conditioned in-context reinforcement. Removing in-context reinforcement keeps the same retriever and communication protocol, but strips the distilled tool-use lessons from the main agent prompt. Removing the retriever agent further disables external retrieval together with the multi-round communication protocol that lives on top of it, reducing the system to the original local-only base agent.

The results show that each component contributes to the final performance. Without outcome-conditioned in-context reinforcement, performance drops from 59.6% to 55.8% on SWE-bench Verified and from 48.1% to 45.6% on SWE-rebench. This indicates that simply adding the retrieval protocol is not sufficient: the base agent still needs prompt-level adaptation to learn when to invoke retrieval, how to formulate useful requests, and how to incorporate returned evidence into local repair. Further removing the retriever agent drops performance to 52.8% on SWE-bench Verified and 42.8% on SWE-rebench, which is the local-only base agent and forms the endpoint of the ablation. The 3.0-point gap on SWE-bench Verified compares two settings without in-context reinforcement and measures the joint contribution of the retriever and the multi-round communication protocol in the absence of adaptation.

For clarity, the $55.8 - 52.8 = 3.0$ point comparison above is between two settings without in-context reinforcement. It measures the joint contribution of retrieval and its communication protocol in the absence of adaptation, rather than an increment on top of adaptation. The following independent no-`WAIT` comparison isolates protocol-level interaction while retaining adaptation.

Effect of protocol-level `WAIT`. The no-`WAIT` ablation retains the dedicated retriever but disables its protocol-level handoff for clarification and resumption. The main/retriever models, evaluation split, adaptation memory, filtering policy, and overall retrieval-budget limits are held fixed, with only interface-specific instructions adjusted. The retriever can still search and browse within its allocated budget, and ordinary browser-level waiting remains available. This isolates protocol-level interaction without removing retrieval itself. Table 3 reports the corresponding results on both benchmarks.

No-`WAIT` analysis. Disabling protocol-level `WAIT` reduces Pass@1 from $59.6 \pm 1.37\%$ to $56.7 \pm 1.57\%$ on SWE-bench Verified and from $48.1 \pm 2.14\%$ to $46.2 \pm 1.38\%$ on SWE-rebench, corresponding to decreases of 2.9 and 1.9 percentage points, respectively. Unlike removing the retriever entirely, this comparison retains external search and browsing, as well as in-context adaptation. The lower resolution rates on both benchmarks support the value of cross-agent clarification and retrieval resumption beyond external access alone. In full BRIDGE, `WAIT` enables the retriever to request additional repository-side context from the main agent and continue its search with those constraints.

Table 4: **Temporal-policy comparison** on the 450 held-out SWE-bench Verified instances (three-run mean $\pm$ standard deviation). Both BRIDGE conditions use the original GPT-5 retriever; the stricter condition adds LLM-mediated temporal filtering against each problem's publication date.

Setting (GPT-5-mini main agent)	Pass@1 (%)
Local-only	52.8 ± 1.69
BRIDGE, original retrieval policy	59.4 ± 1.32
BRIDGE, stricter temporal policy	59.0 ± 1.28

4.5 Retrieval Validity and External-Knowledge Analysis

Content and temporal checks. Relevant external knowledge is allowed: documentation, API semantics, version behavior, and general mathematical facts may be exactly what the agent lacks. Direct target-task solutions are not the intended information source. Our content judge screens retrieved answers for patches, exact edits, complete fix recipes, and upstream issue/PR/commit content that directly reveals the required fix. The exact prompt is provided in Appendix B. This distinction is important because removing a diff alone does not remove answer exposure: a target-issue discussion may already reveal the root cause or code location.

The stricter policy additionally uses an LLM to assess source dates throughout the retrieval process against the corresponding problem's publication date and filters content judged to postdate that boundary before returning it to the main agent. This is an LLM-mediated temporal judgment, not a fixed domain allowlist or a timestamp-certified search index. The reported temporal-filter intervention rate in the final runs is 17.6%, with interventions attributed to content judged to exceed the problem-date boundary. Appendix B.2 details this policy.

Under this stricter policy, Pass@1 is $59.0 \pm 1.28\%$, close to the original $59.4 \pm 1.32\%$ and above the local-only $52.8 \pm 1.69\%$ (Table 4). This sensitivity check supports that the gain is not eliminated by the tested temporal restriction. It is not proof of zero leakage: date inference can be imperfect, pages can change, and semantically answer-revealing content requires further auditing.

Illustrative clean case. For `sympy__sympy-13878`, the agent must implement predefined cumulative distribution functions (CDFs) for nine continuous distributions whose PDFs cannot always be integrated automatically. BRIDGE retrieves general mathematical references, including Wikipedia's arcsine distribution, MathWorks' gamma distribution, and a Student-t CDF formula involving the hypergeometric function ${}_2F_1$. Both the local-only and BRIDGE trajectories attempt nine CDF implementations, but only the BRIDGE patch uses the required hypergeometric function; it passes the evaluation, while the local-only patch fails. These sources concern general distribution formulas rather than the target SymPy patch. The trace illustrates the intended mechanism: acquiring general mathematical knowledge that the agent can apply to a concrete repository repair.

4.6 Performance Analysis: Token Consumption and Trajectory Length

Table 5: **Average token consumption and interaction steps per trajectory.** Token counts are reported in thousands (K). BRIDGE keeps agent-side token usage comparable to the local-only agent while reducing average trajectory length by 16.3%; the additional total token cost is primarily retrieval-side.

Method	**Agent Tokens (K)**	**Retrieval Tokens (K)**	**Total Tokens (K)**	**Avg. Steps**
GPT-5-mini (local-only)	34.6	–	34.6	21.23
GPT-5-mini + BRIDGE	35.7	30.0	65.6	17.77

We measure per-trajectory token consumption and interaction steps to characterize the computational cost of BRIDGE.

Table 5 shows a slight increase in agent-side tokens, from 34.6K to 35.7K per trajectory. The external retriever handles web-facing operations and returns condensed task-relevant evidence, keeping the main agent's context comparable to local-only reasoning. Retrieval increases total tokens from 34.6K to 65.6K per trajectory, while average interaction steps decrease from 21.23 to 17.77, a 16.3% reduction. This is consistent with external evidence reducing some unproductive local exploration.

Overall, BRIDGE improves issue resolution and shortens local trajectories at approximately $1.9\times$ the total token count.

5 RELATED WORK

Repository-level software engineering agents. SWE-bench evaluates issue resolution through repository inspection, editing, and validation (Jimenez et al., 2024). SWE-agent studies the agent–computer interface (Yang et al., 2024), and OpenHands provides a platform for agents interacting with code and execution environments (Wang et al., 2025). Other approaches improve localization and repair, repository representations, coordination, or inference-time exploration (Xia et al., 2025; Li et al., 2025; Ouyang et al., 2025; Tao et al., 2024; Garg & Huang, 2026; Applis et al., 2026; Antoniades et al., 2025); training-oriented methods exploit task environments and verifiable outcomes (Pan et al., 2025; Wei et al., 2025). BRIDGE addresses a complementary question: how should a repository agent acquire and use information that is not available in its local environment?

External knowledge and retrieval. Documentation, release histories, and developer discussions can inform software development (Xavier et al., 2017; Treude & Robillard, 2016; Ajam et al., 2020). Retrieval-augmented generation incorporates external documents into generation (Lewis et al., 2020), but repository repair additionally requires relating evidence to local versions, configuration, and concrete API usage. RepoMem retrieves repository memory for localization (Wang et al., 2026), while SWE-Exp uses previous issue-resolution experience (Chen et al., 2026). BRIDGE instead targets knowledge acquired from the open web during the current repair. OpenHands-Versa gives coding agents browsing capabilities directly (Soni et al., 2026). Our distinction is organizational: a dedicated retriever manages external acquisition, while an interactive protocol communicates the repository-side context needed to interpret its findings. The direct-tools comparison controls for access to the same web actions and for outcome-conditioned adaptation; it evaluates this complete organization, not browsing access alone.

Tool-use adaptation. An external interface introduces decisions about when to retrieve, what context to communicate, and how to use the answer. DSPy optimizes modular language-model pipelines (Khattab et al., 2024); Reflexion uses verbal feedback without gradient updates (Shinn et al., 2023). SWE-Gym and SWE-RL improve software engineering agents through supervised or reinforcement learning with task outcomes (Pan et al., 2025; Wei et al., 2025). Concurrently, Ye et al. (2026) study general tool use by inserting demonstrations into reinforcement-learning rollouts and gradually removing them while optimizing model parameters. In contrast, BRIDGE keeps the underlying models fixed. It contrasts successful forced-retrieval and failed free-run trajectories, distills their behavioral differences, and refines prompt memory for the external-retrieval protocol. This is outcome-conditioned prompt adaptation, not a gradient-based policy update.

6 CONCLUSION

We studied external knowledge acquisition as coordination between repository-local problem solving and external evidence gathering. BRIDGE pairs a repository agent with a dedicated retriever, provides exploratory and focused interfaces, and supports context requests and search continuation through a multi-round protocol. Outcome-conditioned prompt memory, constructed on a disjoint bootstrap partition and fixed before evaluation, adapts the parameter-frozen agent to these interfaces. Together, these components connect external acquisition to the agent's local hypotheses and constraints while retaining repository-side patch implementation and validation.

The evaluation places this organization alongside local-only reasoning and direct web-tool access. On held-out SWE-bench Verified tasks, the matched GPT-5-mini comparison reports a higher issue-resolution rate for BRIDGE than for direct access to the same web tools with the same adaptation. Retriever-capacity comparisons, protocol and adaptation ablations, and temporal-policy sensitivity experiments further characterize the framework's behavior. The computational measurements show a trade-off: in the original GPT-5-mini main/GPT-5 retriever configuration, fewer repository interaction steps accompany higher total reported token use. These findings support coordinated evidence acquisition as a useful complement to repository-local reasoning and motivate designing retrieval interfaces, context exchange, and prompt-level adaptation together in software engineering agents.

AI USE STATEMENT

Generative AI assisted manuscript drafting and revision, translation, literature discovery and citation verification. Language models are also components of the proposed agent and prompt-adaptation procedure, as described in the method. The authors are responsible for verifying all text, claims, and artifacts before submission.

REFERENCES

George Ajam, Carlos Rodríguez, and Boualem Benatallah. API topics issues in stack overflow Q&As posts: An empirical study. In *CLEI*, 2020.

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. SWE-Search: Enhancing Software Agents with Monte Carlo Tree Search and Iterative Refinement. In *ICLR*, 2025.

Leonhard Applis, Yuntong Zhang, Shanchao Liang, Nan Jiang, Lin Tan, and Abhik Roychoudhury. Unified software engineering agent as AI software engineer. In *ICSE*, 2026.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. SWE-rebench: An Automated Pipeline for Task Collection and Decontaminated Evaluation of Software Engineering Agents. In *NeurIPS*, 2025.

Silin Chen, Shaoxin Lin, Yuling Shi, Heng Lian, Xiaodong Gu, Longfei Yun, Dong Chen, Lin Cao, Jiyang Liu, Nu Xia, and Qianxiang Wang. SWE-Exp: Experience-Driven Software Issue Resolution. *arXiv preprint*, 2026.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang (Eric) Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-One: A generalist multi-agent system for solving complex tasks. *arXiv preprint*, 2024.

Spandan Garg and Yufan Huang. Debug2Fix: Can Interactive Debugging Help Coding Agents Fix More Bugs? In *ASE (Industry Showcase)*, 2026.

Google DeepMind. Gemini 3 Flash model card, 2025.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. In *ACL*, 2024.

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? In *ICLR*, 2024.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. In *ICLR*, 2024.

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 *NeurIPS*, 2020.

Hongwei Li, Yuheng Tang, Shiqi Wang, and Wenbo Guo. PatchPilot: A Cost-Efficient Software Engineering Agent with Early Attempts on Formal Verification. In *ICML*, 2025.

OpenAI. GPT-5 system card, 2025.

Siru Ouyang, Wenhao Yu, Kaixin Ma, Zilin Xiao, Zhihan Zhang, Mengzhao Jia, Jiawei Han, Hongming Zhang, and Dong Yu. RepoGraph: Enhancing AI Software Engineering with Repository-Level Code Graph. In *ICLR*, 2025.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. In *ICML*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

Aditya Bharat Soni, Boxuan Li, Xingyao Wang, Valerie Chen, and Graham Neubig. Coding agents with multimodal browsing are generalist problem solvers. In *Findings of EACL*, 2026.

Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. MAGIS: LLM-Based Multi-Agent Framework for GitHub Issue Resolution. In *NeurIPS*, 2024.

Christoph Treude and Martin P. Robillard. Augmenting API documentation with insights from stack overflow. In *ICSE*, 2016.

Boshi Wang, Weijian Xu, Yunsheng Li, Mei Gao, Yujia Xie, Huan Sun, and Dongdong Chen. Improving code localization with repository memory. In *ICLR*, 2026.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. In *ICLR*, 2025.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. SWE-RL: Advancing LLM Reasoning via Reinforcement Learning on Open Software Evolution. In *NeurIPS*, 2025.

Laerte Xavier, Aline Brito, Andre Hora, and Marco Tulio Valente. Historical and impact analysis of API breaking changes: A large-scale study. In *SANER*, 2017.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-based software engineering agents. In *FSE*, 2025.

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 *NeurIPS*, 2024.

Yaoqi Ye, Yiran Zhao, Keyu Duan, Zeyu Zheng, Kenji Kawaguchi, Cihang Xie, and Michael Qizhe Shieh. In-context reinforcement learning for tool use in large language models. *arXiv preprint*, 2026.

A Implementation and Reproducibility Details

A.1 Retriever Interface and Interactive Execution

The main agent invokes the retriever with a message $m_t = (\mu_t, q_t, u_t, c_t)$, where μ_t is either overview search or focused retrieval, q_t is the information need, u_t is an optional seed URL, and c_t contains relevant local context. In the implementation, `engine_search` implements overview search and `search_web` implements focused retrieval. Overview search returns a short set of candidate answers paired with source URLs, $e_t^{\text{overview}} = \{(a_i, u_i)\}_{i=1}^{n}$. It is intended for questions whose relevant source or sub-question is not yet clear. Focused retrieval starts at the supplied URL, or identifies suitable root pages when no URL is provided, and returns a synthesized answer with sources. Neither interface gives the retriever unrestricted access to the repository: local observations must be communicated by the main agent.

The pseudocode below consolidates the interaction described in the main paper. The selected evaluation policy is applied within retrieval and to returned evidence; its content and temporal components are described in Appendix B. Internal policy checks are abstracted into the search and browsing operations below. A blocked response is not treated as evidence. The main agent can reformulate a question about permitted background knowledge or continue with its local evidence.

Listing 1: Overview and focused retrieval.

```
INTERACTIVE_RETRIEVAL(mode, query, seed_url, local_context):
    message = (mode, query, seed_url, local_context)
    if mode == OVERVIEW:
        candidates = overview_search_under_selected_policy(message)
        return apply_policy(candidates)
    state = initialize_browser(message)
    repeat:
        output, state = browse_or_resume_under_selected_policy(
                            message, state)
        if output is ANSWER:
            return apply_policy(output)
        reason, resume_url, next_command = output.WAIT
        if reason says source unavailable or blocked:
            return unavailable(reason)
        if two consecutive WAITs have the same resume_url,
           no new information, and no concrete next target:
            return unavailable("no retrieval progress")
        if reason requests missing repository-side context:
            context = main_agent.obtain_requested_context(reason)
            message = add_requested_context(message, context)
        retain resume_url and saved browsing evidence
        resume using next_command (with requested context added)
```

A.2 Browser Actions and the Meaning of WAIT

The focused retriever runs in a Selenium browser. At each step it receives the current URL, a screenshot, and numbered interactive elements. It can click, type, scroll, read page text, go back, navigate to search, wait for a page, return an answer, or yield control to the main agent. The browser-level `Wait` action waits for page state; the protocol-level `WAIT` instead suspends retrieval and requests continuation or missing task context. These are distinct operations.

On a protocol-level `WAIT`, the system retains the current URL, partially collected evidence, and the clarification question. The returned `resume_url` and `next_command` allow the main agent to continue from that state. The main agent follows the continuation command unless the retriever requests additional local context, in which case it adds the requested context while retaining the resume URL. The prompt discourages unproductive repetition: after two consecutive `WAIT` results at the same URL with no new information or concrete next target, the agent stops relying on that retrieval path. The retriever's preference to answer within three steps is a prompt instruction, not a hard budget; no equality of realized browsing calls or tokens is assumed.

Main-agent setup and controlled direct access. The repository agent follows mini-SWE-agent v2.0.0 and executes each benchmark instance in its official Docker environment. It inspects the local repository, implements changes, and runs available tests. The direct-tools control receives the same underlying web-search and browser actions, the same outcome-conditioned in-context adaptation, the same GPT-5-mini main model, the same evaluation split, and the same outer interaction budget. It has no dedicated retriever managing these actions: the main agent chooses whether, when, and how to browse. This controls tool availability and adaptation, but not realized computation. Forcing the agent to make a prescribed number of searches would change the agent-controlled direct-use setting.

A.3 Outcome-Conditioned In-Context Adaptation

All language-model parameters remain fixed. We randomly select 50 SWE-bench Verified instances for constructing the tool-use memory and exclude them from the 450-instance evaluation. For each bootstrap task, a forced-retrieval trajectory encourages use of external knowledge; a free trajectory exposes the same protocol without the additional retrieval-use guidance. Repository validation supplies the outcome. Initial contrastive pairs contain a successful forced trajectory that uses external retrieval and a failed free trajectory. The GPT-5-mini teacher compares these trajectories and distills behavioral lessons: when to retrieve, how to formulate the request, what local context to communicate, and how to use the answer. It is not a stronger GPT-5 teacher and does not provide inference-time patches.

Listing 2: Construction and refinement of tool-use memory.

```
BUILD_MEMORY(bootstrap_tasks):
    for task in bootstrap_tasks:
        forced[task] = run_with_forced_retrieval(task)
        free[task] = run_without_added_retrieval_guidance(task)
        evaluate both trajectories using repository validation
    pairs = {(forced[t], free[t]): forced[t] succeeds,
             free[t] fails, and forced[t] uses external retrieval}
    M = compress(teacher_distill_tool_use_differences(pairs))
    repeat:
        adapted = run_BRIDGE(bootstrap_tasks, memory=M)
        failures = pairs of successful forced trajectories and
                   corresponding unsuccessful adapted trajectories
        lessons = teacher_distill_tool_use_differences(failures)
        M = compress(M union lessons, fixed_context_budget)
    until bootstrap validation outcomes or tool-use behavior stabilize
    return M
```

The memory is intended to encode reusable retrieval behavior rather than concrete task patches. This design does not establish independence from the chosen bootstrap sample: sensitivity to alternative random samples has not been measured. The evaluated comparison with direct tools includes adaptation in both conditions, so its difference should not be attributed to the presence versus absence of adaptation. The no-`WAIT` condition in Table 3 separately targets protocol-level interaction.

A.4 Bootstrap Task Manifest

Table 6 retains the 50 recorded bootstrap identifiers. These tasks are excluded from the SWE-bench Verified evaluation, which uses the remaining 450 instances.

A.5 Retriever Capacity

BRIDGE achieves 59.4% issue resolution with the original retriever and 59.6% with GPT-5-mini in both roles, improving on local-only by 6.6 and 6.8 percentage points, respectively (Table 7). The improvement is retained when the retriever uses the same GPT-5-mini model as the main agent. This extends the benefit of BRIDGE beyond the original stronger-retriever configuration; the small difference between the two BRIDGE variants does not establish a ranking between retriever models.

Table 6: The 50 bootstrap instances, using the identifiers recorded in the original experiment manifest.

Instance ID		
sympy__sympy-22080	scikit-learn__13142	matplotlib__25332
django__django-15315	pytest-dev__5809	scikit-learn__13135
django__django-11333	matplotlib__23299	pydata__xarray-6744
matplotlib__20826	django__django-16560	pydata__xarray-6461
django__django-11532	django__django-15103	sympy__sympy-15017
django__django-16642	sympy__sympy-16792	django__django-13417
django__django-14855	django__django-14007	matplotlib__24870
sphinx-doc__8721	psf__requests-2317	django__django-15368
pylint-dev__4604	django__django-11880	django__django-11095
sympy__sympy-13615	django__django-16136	django__django-15554
django__django-13089	django__django-16661	pydata__xarray-6992
django__django-15987	sympy__sympy-17139	django__django-15863
django__django-14725	sphinx-doc__8595	django__django-13363
sympy__sympy-14248	sympy__sympy-14531	sympy__sympy-13852
pytest-dev__7982	django__django-10880	django__django-14017
django__django-15280	sympy__sympy-19346	pylint-dev__4661
django__django-11265	sphinx-doc__9229	

Table 7: **Retriever-capacity comparison.** Both settings use the GPT-5-mini main agent with the same interaction protocol and in-context adaptation; only the retriever model differs. Values are Pass@1 percentages (mean $\pm$ standard deviation).

Method	SWE-bench Verified	SWE-rebench
BRIDGE, GPT-5 retriever (original)	59.4 ± 1.32	48.3 ± 1.23
BRIDGE, GPT-5-mini retriever	59.6 ± 1.37	48.1 ± 2.14

B Retrieval Policy, Leakage Controls, and Audit Boundary

B.1 Content-Based Answer Filter

Internal information comprises the model's parametric knowledge and everything directly accessible within the benchmark environment, including the issue, checked-out code and documentation, dependencies, and execution feedback. External information is obtained outside that environment during problem solving. This access-based definition is separate from the distinction between permitted evidence and benchmark leakage. Relevant documentation is useful precisely because it informs the repair; however, task-specific post-hoc discussions may reveal the answer without reproducing a literal patch.

The original content filter uses GPT-5.4 to judge a candidate response together with the task description. It discards responses classified as answer-revealing before they are returned to the main agent. The following is the supplied system prompt, with line wrapping normalized for presentation.

Listing 3: Executed content-judge system prompt.

```
You are a strict answer filter for a SWE-bench web-search tool.
The main coding agent may use web search for public documentation, API
    semantics,
release notes, errors, and high-level source context.
It must NOT receive direct answers to the current coding task from
    search_web.

Filter if the answer contains any of:
- a direct patch, final code edit, exact replacement code, or complete
    fix recipe for the task;
- concrete diff hunks, exact changed lines, or file-by-file patch
    instructions for the target issue;
- an upstream PR/commit answer that directly tells the agent what to
    change for this instance.
```

```
Do NOT filter safe context such as official API semantics, error
    explanations,
docs summaries, version behavior, or broad hints that do not reveal the
    direct
patch/final solution.
Return JSON only.
```

The output fields are `should_filter`, `category`, and `reason`. The supplied category labels are `direct_patch_or_final_solution`, `exact_changed_lines_or_diff`, `safe_context_or_docs`, and `unclear`. This prompt documents the executed policy; it is not a guarantee that all indirect solution evidence is recognized. In particular, knowing the target defect location and root cause from its eventual issue-resolution thread can already provide substantial answer information. Literal-diff rejection alone is therefore insufficient for attributing gains to general external knowledge.

B.2 Stricter Temporal Policy and Aggregate Intervention

The stricter evaluation adds an LLM-mediated temporal check during the retrieval process. The judge assesses source publication time relative to the corresponding problem's publication time and filters content judged to postdate the problem. The LLM considers the retrieval process, rather than applying a fixed domain allowlist or relying on a search API's hard timestamp cutoff. The temporal check complements, rather than replaces, the content boundary above. It aims to reduce access to post-hoc task-specific evidence while retaining useful background knowledge.

Under this stricter policy, BRIDGE obtains $59.0 \pm 1.28\%$ over three runs, compared with $59.4 \pm 1.32\%$ under the original policy and $52.8 \pm 1.69\%$ for local-only. The improvement persists under the added temporal check. The reported temporal-filter intervention rate in the final runs is **17.6%**, with interventions attributed to content judged to postdate the problem. This aggregate is not a measured distribution over the content-judge categories.

General documentation can change after its original publication, and LLM date judgments can be incorrect or uncertain; consequently, this policy is a sensitivity analysis rather than a certified historical-web evaluation. A source-level audit with independently checked timestamps would further characterize these limitations. SWE-rebench's effort to reduce training contamination likewise does not remove live-retrieval leakage by itself.

C PROMPT DETAILS

The following retrieval-specific main-agent excerpt is retained from the submitted prompt; typography and wrapping are normalized. The full base-agent instructions additionally cover local inspection, editing non-test source files, execution, verification, and patch submission. These instructions describe the agent implementation, not a permission to treat arbitrary retrieved text as executable instructions.

Listing 4: Main-agent retrieval instructions (excerpt).

```
Global search rules:
1. Never search private local paths or repo-only strings such as `/
    testbed`, `/workspace`, `/root`, or absolute paths.
2. Form queries as: project name + API symbol + error keyword or
    behavior keyword.
3. Prefer official sources: official docs, official GitHub, release
    notes.
4. If results are poor, rewrite and simplify rather than repeating the
    same query.
5. After each search, immediately translate what you learned into a
    concrete local action.
6. Do not keep searching in a loop. Search, extract the takeaway, then
    inspect, reproduce, edit, or test.
7. For time-sensitive questions, prefer sources that show versions or
    dates.
8. If `search_web` returns `WAIT`, do not treat it as evidence. It is a
    continuation request, not a final answer.
9. If `search_web` returns `WAIT`, your next command MUST be the
    provided `next_command`, unless the `WAIT` reason explicitly asks
    you for missing task context. In that case, add that context to the
    query and keep the provided `resume_url` as the second argument.
10. Continue this `WAIT` -> `search_web` loop until `search_web` returns
     a real answer, or until the returned reason clearly says the
    information is unavailable/blocked rather than merely step-limited.
11. Do not enter an infinite loop: if two consecutive `WAIT` results
    point to the same `resume_url` with no new information and no
    concrete next target, stop relying on web search and explain that
    the source could not be retrieved.
```

The retriever prompt below is reproduced from the submitted implementation. Browser `Wait` and protocol `WAIT` retain their distinct capitalization and meanings. The preference for source URLs improves inspectability but is not itself a source-validity check; the separate evaluation policy governs which evidence is returned.

Listing 5: Focused-retriever prompt.

```
{{task}}
You are a web-browsing assistant running inside a Selenium browser.
You will be given:
1) The current URL.
2) A screenshot of the current page.
3) A list of numbered interactive elements shown on the screenshot (each
     has a numeric label).
Your job is to search the web and answer the user's query with sources.
You may also pause and hand control back to the main coding agent when
    the search needs more context or when the current search budget is
    not enough.
You must respond with EXACTLY this format:
Thought: <short reasoning>
Action: <one action>
Allowed actions (choose ONE each step):
- Click [N]
- Type [N]; [TEXT]
- Scroll [WINDOW]; [up/down]
- ReadText
- GoBack
```

```
- Google
- Wait
- WAIT; [WHY_THE_MAIN_AGENT_SHOULD_CONTINUE_OR_CLARIFY]
- ANSWER; [FINAL_ANSWER_WITH_SOURCES]
Rules:
- Prefer to answer in <= 3 steps when possible.
- When you provide ANSWER, include a few source URLs inline (at least 2
    when possible).
- Use ReadText when the page contains substantial code, prose, tables,
    or diffs that are hard to read from the screenshot alone.
- Keep the final answer concise and complete. Prefer 3-6 short bullets
    or a short paragraph with sources.
- Use `WAIT; ...` only when you should pause this browser search and
    return control to the main agent.
- Use `WAIT; ...` if you need missing context from the main agent, or if
     the current step budget is running out but the current page/search
    path still looks promising.
- Do not use `WAIT; ...` when the evidence strongly indicates the
    information does not exist or is blocked after reasonable attempts.
    In that case, provide `ANSWER; ...` explaining what you tried and
    why no reliable answer was found.
- If you use `WAIT; ...` because of step budget, say what has been
    learned, what remains to inspect, and why continuing from the
    current URL is likely useful.
```

D ILLUSTRATIVE EXTERNAL-KNOWLEDGE CASE

We examine `sympy__sympy-13878`, in which BRIDGE passes the benchmark evaluation and the observed local-only trajectory fails. The task asks for predefined cumulative distribution functions (CDFs) for continuous distributions for which obtaining a CDF by integrating the density is unsuccessful. Both trajectories attempt nine `_cdf` implementations. The BRIDGE trace retrieves general mathematical references rather than the target SymPy issue, pull request, or patch: a Wikipedia page provides the arcsine CDF, a MathWorks reference provides gamma-distribution information, and an academic reference provides a Student-t CDF containing the Gauss hypergeometric function ${}_2F_1$.

For example, the arcsine CDF on $[a, b]$ is

$$F(x) = \frac{2}{\pi} \arcsin\sqrt{\frac{x-a}{b-a}},$$

and the retrieved Student-t expression uses

$$F_\nu(x) = \frac{1}{2} + \frac{x\,\Gamma((\nu+1)/2)}{\sqrt{\pi\nu}\,\Gamma(\nu/2)} {}_2F_1\left(\frac{1}{2}, \frac{\nu+1}{2}; \frac{3}{2}; -\frac{x^2}{\nu}\right).$$

These are general distribution identities, not instructions naming a target repository change. BRIDGE uses the retrieved hypergeometric expression through SymPy's `hyper`, together with functions such as `asin`, `lowergamma`, `uppergamma`, and `erf`. The observed local-only implementation does not use `hyper` and fails the evaluation; BRIDGE passes. No `engine_search` result in the supplied trace was blocked by the filter.

Trace property	Local-only	BRIDGE
CDF implementations attempted	9	9
External evidence	None	General mathematical references
Student-t implementation	No `hyper` term	Retrieved ${}_2F_1$ expression via `hyper`
Task-specific issue/PR/patch retrieved	Not applicable	None in the supplied trace
Benchmark outcome	Failed	Passed

Table 8: An illustrative trajectory comparison for `sympy__sympy-13878`. It supports the intended external-knowledge mechanism but is not an aggregate source audit or a controlled causal intervention.

The trace supports a concrete interpretation: retrieved mathematical knowledge supplies a formula used in a successful implementation when the compared local-only trajectory lacks that expression. It does not prove that this single expression is the unique cause of the outcome, quantify how often this mechanism occurs, or establish that all retrieved sources across the benchmark are leakage-free. This case is illustrative rather than an aggregate source audit.

E Verification, Scope, and Remaining Limitations

The main agent treats retrieved information as evidence for a repository-level hypothesis, not as a correctness certificate. It implements the resulting change and runs available unit and repository tests. Such executable validation checks the changed program's behavior, but not every retrieved assertion or its applicability to all dependency versions. A test suite can be incomplete, and a plausible web source can be incorrect, stale, or adversarial.

The controlled GPT-5-mini experiments address simpler explanations based on tool availability, prompt adaptation, and a stronger retriever. They do not independently isolate every feature of the retrieval protocol or establish generality across unevaluated model families. The original removal ablation jointly removes the retriever and its communication protocol. The repeated-run GPT-5-mini and Gemini experiments and component ablations report means and standard deviations. The Gemini SWE-rebench difference and adaptation-ablation difference are descriptive, not statistically established effects.

Retrieval adds computation. The original token accounting reports approximately $1.9\times$ total model tokens despite similar main-agent tokens and fewer local interaction steps. These measurements do not establish lower total latency or monetary cost. Bootstrap-sample sensitivity and dependence on live-web availability are also unmeasured. More complete source logging, human audit of temporal decisions, version-aware retrieval, selective triggering, and caching are natural next steps.

Finally, web-facing agents can encounter prompt-injection content, unreliable instructions, and code whose licensing or security implications require review. Retrieved text should not override repository-task boundaries. Sandboxed execution, evidence logging, human review of high-impact changes, and restrictions on irreversible actions remain necessary safeguards; leakage filtering alone is not a general security defense.

F Limitations

Retrieval adds inference cost; fewer repository steps need not reduce total compute. Bootstrap-task sensitivity remains unmeasured. Temporal/content judges mitigate but cannot rule out answer exposure, especially with ambiguous dates or altered historical pages. The mathematical-reference case illustrates utility, not prevalence. Broader backbone evaluation, paired uncertainty estimates, and source-level temporal auditing would strengthen the evidence.

G Reproducibility Statement

Section 4 specifies the evaluation split, comparison controls, repeated-run reporting, and token accounting. Appendix A describes the interaction and adaptation procedures; Appendix B provides the content judge and temporal-policy details.

H Ethics Statement

Web-enabled coding agents can retrieve incorrect, malicious, or answer-bearing material. Retrieved text is evidence to be checked against the local task, not an instruction source, and candidate patches require repository-side validation. Benchmark evaluation additionally requires separating useful background knowledge from target-instance solutions. We therefore report both the implemented filtering policy and its limitations instead of treating filtering as proof of uncontaminated evaluation.