

Tool-Creating LLM Agents Gain Little from Keeping Their Tools

Marek Šuppa*
Cisco Systems &
Comenius University in Bratislava

Jaroslav Kopčan
Kempelen Institute of
Intelligent Technologies

Abstract

Tool-creating LLM agents are widely reported to benefit from accumulating reusable tool libraries. We test this claim with the *create-and-discard ablation*: build the tool, then throw it away. Across two benchmarks (BigCodeBench-Hard, 148 tasks; τ^2 -airline, 50 tasks), four models, and multiple retrieval variants including embedding-based and dependency-aware retrieval, creating and *discarding* tools matches creating and *keeping* them. Over every Create+Reuse–Create+Discard contrast we report—ten in total, spanning both benchmarks—the effect of keeping the tool has median +0.4pp and mean −0.4pp (range −9.5 to +6.0), and every contrast carrying a paired significance test is null ($p \geq 0.09$). On BigCodeBench-Hard all eight conditions cluster within a 2pp band. On τ^2 -airline, a wrapper-matched decomposition splits the raw +18pp Direct-to-Discard gain into +10pp from the agent harness itself, +6pp from a simple “use your tools” instruction, and only +2pp from the tool-creation structure. Tool creation helps agents engage with available capabilities, not build reusable memory—and agent-framework details alone can inflate reported gains by 10pp. We recommend that tool-creation evaluations report a create-and-discard ablation and a harness-matched baseline.

1 Introduction

A growing body of work equips LLM agents with the ability to create their own tools: given a problem, the agent writes a reusable function (typically Python) that encapsulates a solution strategy, then invokes that function to produce an answer (Qian et al., 2023; Cai et al., 2024; Yuan et al., 2024; Wang et al., 2025b). The resulting tool is added to a library for future retrieval, creating an ever-growing repertoire of skills that the agent can draw on (Wang et al., 2024; Xia et al., 2026; Wang et al., 2025a).

This paradigm has produced impressive empirical gains. CREATOR (Qian et al., 2023) outperforms chain-of-thought on MATH. LATM (Cai et al., 2024) achieves GPT-4-level performance at GPT-3.5 cost through “functional caching.” ASI (Wang et al., 2025b) improves WebArena success rate by 23.5% through programmatic skill induction. The implicit claim is that building tools teaches agents to solve problems better.

But this narrative conflates two distinct mechanisms. When an agent creates a tool, it engages in a structured reasoning process: it must decompose the problem, identify a reusable abstraction, write correct code, and verify the output. This *reasoning elicitation*—the cognitive work of tool creation—may itself be the source of improvement, regardless of whether the resulting tool is ever used again. Alternatively, the value may come from *genuine reuse*: the tool encodes a reusable subroutine that saves reasoning effort on future problems. To our knowledge, no published tool-creation paper has ablated these two sources.

This confound has practical consequences. If tool creation helps mainly through reasoning elicitation, then investing in retrieval infrastructure, library management, and tool indexing is wasted effort—the agent should simply be prompted to “think in terms of tools” without

*Correspondence to marek@suppa.sk.

persisting anything. If genuine reuse is the driver, then the tool library is a critical asset that must be carefully managed. And there is growing evidence that management matters: Li (2026) reported preliminary evidence that skill libraries exhibit a *phase transition* in retrieval accuracy as they grow, with semantic confusability among tools causing sudden performance collapse.

The answer depends on three moderators: *task recurrence*, *model capability*, and *retrieval quality*. Experiment 1 tests persistence on a low-recurrence code benchmark, Experiment 2 tests a higher-recurrence agent benchmark, and the cross-benchmark synthesis tests whether stronger retrieval changes the conclusion.

Contributions. We present controlled ablations on two benchmarks with four models that disentangle the reasoning process from artifact reuse:

1. **Discarding tools matches keeping them** (§4 - 5): Across both benchmarks and all four models, creating and discarding tools performs comparably to creating and persisting them. The benefit comes from the act of designing a tool, not from reusing it later.
2. **Harness details can dominate the reported effect** (§5): on τ^2 -airline the upstream Direct agent makes zero API calls on 35 of 50 tasks. Routing that same baseline through our wrapper adds +10pp before any tool creation, and a 379-character prompt adds +6pp more, against +2pp for the tool-creation structure itself. Agent evaluations that do not harness-match their baseline can credit the intervention with several times the effect it earns.
3. **Stronger retrieval does not rescue persistence** (§6): Replacing lexical retrieval with embedding-based retrieval (text-embedding-3-small) leaves both benchmarks in the same band (40.5% on BCB-Hard, 58.0% on τ^2 -airline). Ground-truth-guided probes are similarly null, suggesting the bottleneck is task diversity rather than retrieval quality (§6.1).

2 Related work

Tool creation for LLM agents. Toolformer (Schick et al., 2023) showed that LLMs can learn *when* to invoke tools. Subsequent work extended this to tool *creation*. CREATOR (Qian et al., 2023) showed gains on MATH via tool creation, and LATM (Cai et al., 2024) amortizes GPT-4 reasoning for GPT-3.5 invocation in a cross-model setup. CRAFT (Yuan et al., 2024), KTCE (Ma et al., 2025), LearnAct (Zhao et al., 2024b), and Wölflein et al. (2025) extend the paradigm with validation, evolutionary search, and multi-agent delegation. None ablated whether gains originate from the tool or the reasoning process that produced it.

Skill libraries and lifelong learning. Voyager (Wang et al., 2024) pioneered the skill library paradigm in Minecraft, and ASI (Wang et al., 2025b) extended it to web navigation. SkillRL (Xia et al., 2026), SAGE (Wang et al., 2025a), and ARISE (Li et al., 2026b) train skill-management policies via RL. All assume accumulated skills are net-positive, and none study when accumulation becomes harmful.

Library degradation and the utility problem. Li (2026) reported preliminary evidence of phase-transition degradation in skill libraries, driven by semantic confusability. ToolLibGen (Yue et al., 2025) proposed clustering and consolidation to manage large tool collections. The Agent Skills survey (Xu & Yan, 2026) noted that industry deployments use human curation pipelines. This degradation problem has classical roots: Minton (1990) showed that saved macros in explanation-based learning can slow search by bloating the operator set—the “utility problem”—which is exactly the dynamic we study here for LLM tool libraries. In program synthesis, DreamCoder (Ellis et al., 2021) demonstrated that library learning *can* yield genuine reuse when abstractions are learned through Bayesian compression. The contrast with our findings highlights that naive accumulation without principled compression does not achieve the same benefit. Concurrent work on SkillsBench (Li et al., 2026a) reports that human-curated skills raise average pass rate by roughly 16 percentage

points, but that skills the model authors for itself provide no benefit on average—models cannot reliably write the procedural knowledge they benefit from reading. SkillsBench and our work are complementary: SkillsBench asks a *quality* question (self-generated vs. curated skills), while we ask a *reuse* question (persisted vs. discarded tools) and also isolates the reasoning-elicitation mechanism through the create-and-discard ablation. To the best of our knowledge, no prior work proposes or evaluates automated admission control for LLM tool libraries.

Trajectory memory. ExpeL (Zhao et al., 2024a) stores and retrieves completed trajectories to learn from experience. Reflexion (Shinn et al., 2023) stores verbal self-reflections rather than full trajectories, representing a lighter-weight form of experience accumulation. Xiong et al. (2025) showed that naive trajectory memory causes error propagation through “experience-following” behavior. CLIN (Majumder et al., 2023) proposed causal abstractions as an intermediate representation. Memex(RL) (Wang et al., 2026) compressed trajectories via structured summaries. No work directly compares trajectory memory against abstracted tools under controlled token budgets.

Code generation as structured reasoning. PAL (Gao et al., 2023) and Program of Thoughts (Chen et al., 2023) demonstrated that writing code—even non-reusable programs—improves reasoning by offloading computation to a runtime, directly relevant to our Discard_{full} condition. Self-Refine (Madaan et al., 2023) parallels the verify-then-revise loop in our pipeline, and LATS (Zhou et al., 2024) shows that structured exploration drives improvement beyond longer thinking. Scaling test-time compute (Snell et al., 2025) motivates our inclusion of o4-mini in the cross-model comparison.

Code generation benchmarks. BigCodeBench (Zhuo et al., 2025) evaluates code generation with diverse function calls and complex instructions, providing a natural testbed for tool creation. LiveCodeBench (Jain et al., 2025) offers contamination-free evaluation with rolling updates. DS-1000 (Lai et al., 2023) targets data science code generation. We use BigCodeBench-Hard as our primary benchmark because tool creation is ecologically natural (composing library calls) and current frontier models have substantial headroom ($\sim 40\text{--}55\%$ direct accuracy).

3 Experimental setup

3.1 Benchmarks

We evaluate on two benchmarks that span the spectrum from low to high subproblem recurrence:

BigCodeBench-Hard (Zhuo et al., 2025). 148 code-generation tasks (the Hard split of BigCodeBench) requiring composition of diverse library function calls under complex natural-language instructions. Each task has a ground-truth test suite for automated verification. Current frontier models achieve 40–55% on this split, providing substantial headroom for tool creation to help, while the full BigCodeBench is approaching saturation.

τ^2 -airline (Yao et al., 2025). 50 agentic customer-service tasks in a simulated airline reservation system, drawn from the τ^2 benchmark suite. Tasks require multi-step tool use: looking up reservations, changing flights, processing refunds, and enforcing policy constraints. Scoring combines action correctness, policy compliance, and natural-language assertions evaluated by an LLM judge. Unlike BigCodeBench-Hard, τ^2 -airline has *high* subproblem recurrence: many tasks share common API-call patterns (e.g., booking lookups, cancellation eligibility checks), making this a natural complement for testing whether tool persistence helps when subtasks genuinely recur. Current frontier models span a wide range on this benchmark (40–66% direct accuracy across our four models), providing both headroom and a natural gradient of model capability.

3.2 Agent architecture

All experiments use a common agent scaffold with a ReAct-style (Yao et al., 2023) action loop. The agent receives a task description, reasons about the approach, and can take actions including `execute(code)`, `create_tool`, `retrieve_tool`, and `invoke_tool`.

Model selection. Our primary experiments use **Claude Sonnet 4.5** (Anthropic, \$3/\$15 per M input/output tokens), a current-generation frontier model with strong tool-creation capabilities and the same model family used by a recent tool-creation precedent in our bibliography (Wang et al., 2025b). To test whether findings generalize across model families and capability levels, we also run on GPT-5 Mini (OpenAI, mid-range), Gemini 2.5 Pro (Google, frontier), and o4-mini (OpenAI, a reasoning model with built-in chain-of-thought). Each tool is stored as a Python function with a short name, natural-language summary, and source body. By default, retrieval uses term-frequency cosine similarity—bag-of-words token counts with no IDF weighting—against this stored text. We also test embedding-based retrieval (`text-embedding-3-small`) and benchmark-aware variants as diagnostic probes in §6.1. BigCodeBench-Hard experiments use provider-default generation parameters (no temperature override), while τ^2 -airline explicitly sets temperature 0.

3.3 Tool creation pipeline

When the agent creates a tool, it (1) designs a natural-language specification, (2) implements a Python function, and (3) verifies it via syntax checking. The only difference between conditions is whether the validated tool is stored.

3.4 Baselines and controls

We compare eight conditions, each using the same underlying LLM. The conditions are designed to isolate three distinct sources of improvement: (a) the *structured reasoning* that tool creation forces, (b) *same-task executable scaffolding* (running the created tool on the current problem), and (c) *future-task reuse* (persisting the tool for later retrieval).

1. **Direct:** standard ReAct agent with no tool creation or retrieval. Solve each task from scratch.
2. **CoT-Extended:** extended chain-of-thought reasoning, controlling for thinking longer.
3. **Structured-CoT:** same design/implement/verify structure as tool creation but with pseudocode only (no execution).
4. **Create+Discard_{full}:** full design/implement/verify cycle but the tool is never stored or invoked, isolating pure reasoning elicitation.
5. **Create+Invoke_{once}:** tool is created and invoked on the current task but discarded afterward, isolating same-task scaffolding.
6. **Create+Reuse:** the full tool-creation pipeline. The agent creates tools, invokes them on the current task, and adds them to the library for future retrieval. This is the standard paradigm from CREATOR/LATM/ASI.
7. **Trajectory Retrieval:** stores and retrieves complete successful trajectories as few-shot exemplars instead of abstracted tools.
8. **Create+Admit:** identical to Create+Reuse but with a syntax-and-distance admission policy gating library entry.

These conditions form a causal ladder:

Method	BCB-H (all)	95% CI	Δ vs. Direct	Natural reuse	Tokens/task
Direct	38.5	[30.4, 46.6]	—	—	1338
CoT-Extended	40.5	[32.4, 48.6]	+2.0pp	—	1651
Structured-CoT	39.9	[31.8, 48.0]	+1.4pp	—	—
Create+Discard _{full}	39.2	[31.1, 47.3]	+0.7pp	—	3985
Create+Invoke _{once}	39.9	[31.8, 48.0]	+1.4pp	—	—
Create+Reuse	39.9	[31.8, 48.0]	+1.4pp	91.2%	6488
Trajectory Retrieval	35.8	[28.4, 43.2]	−2.7pp	—	—
Create+Admit	39.9	[31.8, 48.0]	+1.4pp	92.6%	—

Table 1: **Experiment 1: Reasoning elicitation ablation on BigCodeBench-Hard (Sonnet).** Aggregate pass rate (%) with 95% bootstrap CIs ($n_{\text{boot}}=10,000$), delta versus Direct, natural reuse rate, and tokens per task.

Condition	Structured reasoning	Same-task exec	Future reuse
Direct	—	—	—
CoT-Extended	(unstructured)	—	—
Structured-CoT	✓	—	—
Create+Discard _{full}	✓	—	—
Create+Invoke _{once}	✓	✓	—
Create+Reuse	✓	✓	✓
Trajectory Retrieval	—	—	(trajectories)
Create+Admit	✓	✓	(gated)

Comparing adjacent rows isolates each factor. The key comparisons are: Structured-CoT vs. Create+Discard_{full} (does code execution during creation matter, even without invocation?), Create+Discard_{full} vs. Create+Invoke_{once} (does same-task invocation help?), and Create+Invoke_{once} vs. Create+Reuse (does future reuse help beyond same-task scaffolding?).

4 Experiment 1: Is the value in the tool or the thinking?

Hypothesis. On tasks without recurring subproblems, the gains from tool creation are attributable primarily to the reasoning process, not the tool artifact. Specifically, Create+Discard will achieve performance within 1pp of Create+Reuse on BigCodeBench-Hard. If the benchmark mainly rewards on-the-fly reasoning rather than persistent helper artifacts, both methods may remain close to Direct.

Method. We run all eight conditions on BigCodeBench-Hard (148 tasks in the current v0.1.2 split) using the Sonnet variant `anthropic.claude-sonnet-4-5-20250929-v1:0` as the primary model. The current public split does not expose an explicit easy/hard subgroup label, so the main Experiment 1 table reports aggregate pass rate, Δ vs. Direct, natural reuse, and tokens/task rather than a per-difficulty breakdown. For the four key conditions (Direct, Create+Discard_{full}, Create+Invoke_{once}, Create+Reuse), we also run on GPT-5 Mini, Gemini 2.5 Pro, and o4-mini to test cross-model generalization. The more recent GPT-5.4 is evaluated on the Direct, Create+Discard, and Create+Reuse conditions (Table 2). For each run we record (a) the solve rate, (b) the token budget consumed by tool-creation reasoning, and (c) the natural reuse rate (what fraction of problems does Create+Reuse actually retrieve and invoke a previously created tool).

Results. All eight Sonnet conditions cluster within a 2pp band (Table 1). Create+Discard (39.2%) is within 0.7pp of Create+Reuse (39.9%, $p=0.93$), and both remain close to Direct (38.5%). CoT-Extended is the strongest at 40.5%, suggesting that any gain on BCB-Hard is modest reasoning elicitation rather than a benefit from preserving the tool artifact. In cost terms, Create+Discard uses $2.9\times$ and Create+Reuse $4.7\times$ the tokens per solved task compared with Direct, for the same accuracy.

Model	Type	Direct	Discard _{full}	Invoke _{once}	Create+Reuse	Reuse-Discard
Sonnet 4.5	frontier	38.5	39.2	39.9	39.9	+0.7pp ($p=.93$)
GPT-5 Mini	mid-range	32.4	33.8	31.8	33.1	−0.7pp ($p=.97$)
Gemini 2.5 Pro	frontier	36.5	35.8	33.1	26.4	−9.5pp ($p=.09$)
o4-mini	reasoning	39.2	35.1	36.5	35.8	+0.7pp ($p=.93$)
GPT-5.4	frontier	41.2	43.9	—	39.2	−4.7pp

Table 2: **Cross-model generalization on BigCodeBench-Hard (148 tasks).** Solve rate (%) for the four key conditions. The rightmost column shows the Reuse-Discard delta with bootstrap p -values. No model shows a statistically significant advantage for persistence. Gemini 2.5 Pro exhibits a notable (though not quite significant at $\alpha=0.05$) *negative* effect of persistence (−9.5pp, $p=0.09$). The final row reports GPT-5.4 (gpt-5.4-2026-03-05), a more recent frontier model (95.3% natural reuse), which shows the same pattern: Create+Reuse again lands below both Create+Discard (−4.7pp) and Direct. GPT-5.4 is evaluated on the Direct, Create+Discard, and Create+Reuse conditions, so its Create+Invoke_{once} cell and paired bootstrap p -value are omitted. All rows share a single provider route, so absolute accuracies here are comparable to each other but not to the OpenRouter-served runs in Table 8.

Interpreting the null band. The persistence null holds across all four models (Table 2), and three-run replication puts it within the within-condition noise floor ($\sigma=0.7$ –2.2pp; Appendix B.1). At $n=148$ with a base rate of $\sim 39\%$, a two-sided McNemar’s test at $\alpha=0.05$ with 80% power detects pairwise effects of roughly 13–15pp, so the 2pp band is evidence against a >13 pp benefit from persistence, not evidence that the true effect is zero. Two finer-grained observations lend support. First, Structured-CoT (39.9%) matches Create+Discard_{full} (39.2%), so even *pseudocode*—without code execution—suffices to elicit the reasoning benefit. Second, Trajectory Retrieval (35.8%) is the only condition below Direct (−2.7pp), which we attribute to retrieval-based anchoring: injecting a full prior trajectory anchors the model on a previous task’s approach rather than reasoning freshly, paralleling the “experience-following” degradation reported by Xiong et al. (2025).

Per-task concordance confirms the null result. Across all eight Sonnet conditions, 67.6% of the 148 tasks have identical outcomes—21.6% solved by every condition, 45.9% by none—yielding Fleiss’ $\kappa=0.737$ (substantial agreement), and pairwise Jaccard indices range from 0.66 to 0.84. The conditions largely solve and fail on the *same* tasks, so the 2pp band reflects task-level stability rather than different conditions succeeding on different subsets.

Reuse is frequent but unhelpful. Create+Reuse retrieves and invokes a prior tool on 81–91% of tasks across models, yet never outperforms Create+Discard. The model invokes every retrieved tool with zero selectivity: solve rates are identical whether a tool is retrieved (40.0%) or not (38.5%). Per-task flip analysis (Appendix B.5) reveals 8 reasoning-elicitation gains partially offset by 7 retrieval-anchoring failures, paralleling the distractor-context effect in retrieval-augmented generation (Shi et al., 2023). The problem is not tool *quality* but task *diversity*: the 13 created tools are well-structured code, but BCB-Hard’s 148 tasks span unrelated domains, so three tools with common keywords dominate all retrievals regardless of functional relevance (Figure 1). A per-reuse audit (Appendix B.10) confirms this: of the 108 reuse events firing at $\tau=0.55$, only 4% fail on tool *quality*, against 27% from irrelevant retrieval and 38% from task difficulty.

Reasoning models partially ignore tool creation. o4-mini ignores the create_tool instruction on 38% of tasks. Compliant tasks benefit (+3.3pp) while non-compliant tasks are harmed (−16.1pp), so the aggregate masks opposite effects (Appendix B.7).

Tool taxonomy. Sonnet produces 172 tools across conditions, 70% generic abstractions, of which only 13.4% are ever invoked by a subsequent task (Appendix B.12). They are good solutions, just not to the problems that come next.

(a) **A stored tool** (`pca_transform_and_plot`, created on task 93; `create_tool` records its name, the one-line summary “Performs PCA transformation on data and creates a scatter plot of the principal components,” and the body below; abridged):

```
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

def pca_transform_and_plot(data, n_components=2):
    """Perform PCA transformation and create a scatter plot.
    Returns: tuple (transformed_data_df, axes_object)."""
    pca = PCA(n_components=n_components)
    transformed_df = pd.DataFrame(
        pca.fit_transform(data),
        columns=[f'PC{i+1}' for i in range(n_components)])
    fig, ax = plt.subplots()
    ax.scatter(transformed_df['PC1'], transformed_df['PC2'])
    return transformed_df, ax
```

(b) **An unrelated later task retrieves it.** On task 139 the agent must *draw histograms of a DataFrame’s numeric columns*. Its `retrieve_tool` query returns three stored “plot” tools, none of which draws a histogram:

```
query: "create histograms for DataFrame numeric columns matplotlib"
1. pca_transform_and_plot (task 93, score 0.182) # PCA scatter plot
2. train_rf_plot_importance (task 37, score 0.175) # random-forest bars
3. kmeans_cluster_plot (task 92, score 0.148) # K-means scatter plot
```

Figure 1: **A worked Create+Reuse retrieval on BCB-Hard (Sonnet 4.5), illustrating why persistence does not help.** Shared surface tokens (*DataFrame*, *plot*, *matplotlib*) make the term-frequency retriever rank three functionally unrelated visualization tools at near-tied scores below 0.2 for a histogram task. The mismatch is systematic: three generic-token tools hold the top rank for 59% of the run’s 135 retrieval events regardless of function (Appendix B.11).

5 Experiment 2: Exploratory case study (τ^2 -airline)

Motivation and benchmark. BigCodeBench-Hard has low subproblem recurrence, so persistence may fail simply because there is nothing to reuse. τ^2 -airline (50 agentic customer-service tasks in a simulated airline system) tests the opposite regime: tasks share common API-call patterns (booking lookups, cancellation checks), and current frontier models span 40–66% direct accuracy.

Method. We run three conditions (Direct, Create+Discard_{full}, and Create+Reuse) on all four models. For Sonnet we also run Create+Admit to test whether admission control rescues the reuse path. Tool-creating conditions inject our harness’s meta-tools (`create_tool`, `retrieve_tool`, `invoke_tool`) alongside the domain-specific airline tools and use a local wrapper that normalizes empty assistant tool-call messages and limits the agent to one tool call per turn to satisfy τ^2 ’s interaction protocol. The original Direct condition uses the upstream τ^2 agent (a different code path). To control for this harness asymmetry, we run a **wrapper-matched Direct** control that routes through the same harness wrapper with only domain tools (no meta-tools, no tool-creation prompt text). Because the benchmark order is fixed and the library grows online across that order, we treat significance-style analyses on τ^2 as exploratory diagnostics rather than definitive causal estimates.

Results. The τ^2 results differ sharply from BigCodeBench-Hard (Table 3). On clean end-to-end Sonnet reruns (all 50 tasks scored, single eval per condition), the raw Direct-to-Discard gap is +18pp (40.0% to 58.0%). However, the wrapper-matched Direct control—which routes Direct through the identical harness wrapper used by tool-creating conditions but without meta-tools or tool-creation prompt text—reaches 50.0%. This decomposes the +18pp into two components:

- **Harness effect (+10pp):** the harness wrapper itself improves performance from 40.0% to 50.0%, likely through message normalization and interaction-protocol handling that the upstream τ^2 agent lacks.

	Direct	Wrapper-Direct	Direct-Active	Struct. CoT	Discard	Reuse	Reuse-Discard
Sonnet 4.5	40.0	50.0	56.0	54.0	58.0	60.0	+2.0pp

Table 3: τ^2 -**airline wrapper-matched results (Sonnet, 50 tasks)**. All conditions run through the same harness wrapper. Wrapper-Direct uses no meta-tools or tool-creation prompt. Direct-Active adds an “always use your tools” instruction but no meta-tools. Structured-CoT uses the design/implement/verify structure with pseudocode only. Relative to upstream Direct (40%), the wrapper adds +10pp, engagement priming (Direct-Active) adds +6pp, and tool creation adds a further +2pp. Structured-CoT (54%) and Direct-Active (56%) are close, confirming that engagement matters more than the design/implement/verify structure. No individual step is significant at $n=50$. Cross-model results (not wrapper-matched) appear in Appendix C.1.

- **Scaffold effect (+8pp), further decomposed by Structured-CoT:** wrapper-matched Structured-CoT reaches 54.0%, splitting the +8pp into +4pp from structured reasoning (pseudocode design/implement/verify) and +4pp from executable tool creation. The full gradient is 50.0% $\rightarrow$ 54.0% $\rightarrow$ 58.0% $\rightarrow$ 60.0% (Wrapper-Direct $\rightarrow$ Structured-CoT $\rightarrow$ Create+Discard $\rightarrow$ Create+Reuse). No individual step reaches significance at $n=50$: Wrapper-Direct vs. Structured-CoT has 3 vs. 5 discordant tasks ($p=0.73$); Structured-CoT vs. Create+Discard has 5 vs. 7 ($p=0.77$); and the overall Wrapper-Direct vs. Create+Discard has 2 vs. 6 ($p=0.29$). The consistent direction across three steps is suggestive of a genuine scaffold gradient, but confirming it requires larger n .

Create+Reuse reaches 60.0%, and the 2pp Reuse-Discard gap is also not significant (McNemar $p=1.000$; paired bootstrap +2.0pp $[-10.0, 14.0]$, $p=0.859$).

Cross-model results (GPT-5 Mini, Gemini, o4-mini) use the upstream Direct baseline, not wrapper-matched, so the harness confound applies. We report them in Appendix C.1 as descriptive only. On Sonnet, Create+Admit (54.0%) matches Structured-CoT but falls below Create+Discard, suggesting admission gating filters out tools that would have helped.

Tool-creation prompting transforms agent engagement. In 35 out of 50 Direct tasks, the agent makes zero API calls despite having full access to domain tools. It asks one clarifying question, then stops responding. In Create+Discard, tasks average 5.1 API calls, yet `create_tool` is called only once across all 50 tasks. The transformation comes from 379 characters of additional system prompt text mentioning “reusable helper functions.” The Direct-Active control confirms this, reaching 56.0% with just a prompt to “use your available tools” and no meta-tools (Appendix C.5). The dominant τ^2 mechanism is engagement priming, not structured reasoning.

Baseline capability. Cross-model τ^2 results (Appendix C.1) suggest weaker-baseline models benefit while stronger ones are flat or hurt: Sonnet (40% Direct) gains +18pp raw, while Gemini (66%) drops 12pp. These comparisons are not wrapper-matched and rest on four data points (Appendix C.6), so we treat them as hypothesis-generating only.

6 Cross-benchmark synthesis

Aggregating every Create+Reuse versus Create+Discard comparison we report—ten contrasts spanning both benchmarks—the effect of persistence has median +0.4pp and mean -0.4 pp (range -9.5 to $+6.0$), the five contrasts carrying paired significance tests are all null, and the two largest effects in the set are negative. This is a descriptive summary rather than a meta-analysis: the contrasts have unequal n and several share a common Create+Discard baseline.

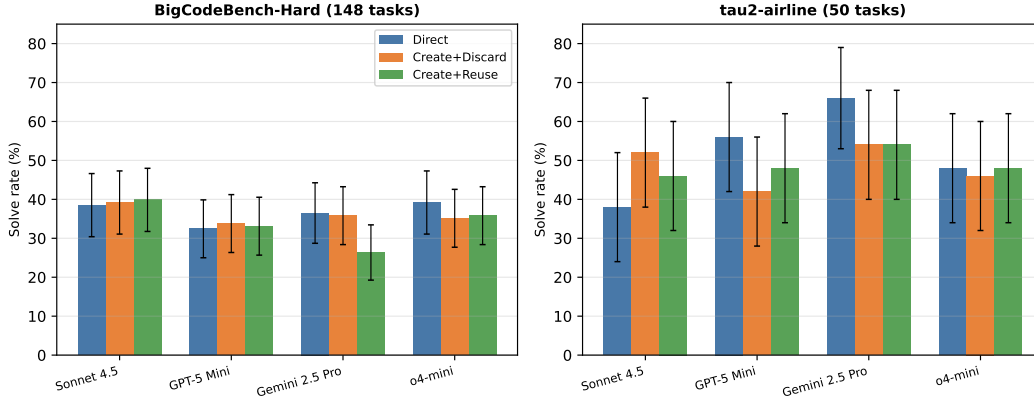


Figure 2: **Side-by-side comparison of the two benchmarks.** On BCB-Hard (left), all conditions cluster within a narrow band. On τ^2 -airline (right), tool creation has large, model-dependent effects. Under standard retrieval, Create+Reuse (green) never clearly beats Create+Discard (orange).

The two benchmarks tell complementary stories (Figure 2). On BigCodeBench-Hard (148 tasks), all conditions cluster within a 2pp band—a null result supported by per-task concordance ($\kappa=0.737$) and three-run replication ($\sigma=0.7\text{--}2.2\text{pp}$). On τ^2 -airline (50 tasks), tool creation produces much larger effects whose direction varies across models. Under our default retrieval configuration, however, no benchmark $\times$ model comparison shows a clear advantage for Create+Reuse over Create+Discard.

6.1 Stronger retrieval does not rescue persistence

Could better retrieval rescue persistence? We probe this along three axes: the *similarity scorer*, the *query construction*, and the retriever’s *selectivity*. Replacing the default lexical scorer with **embedding-based retrieval** (text-embedding-3-small) scores 40.5% on BCB-Hard and 58.0% on τ^2 -airline—both within noise of lexical Create+Reuse (39.9% and 60.0%). Two **benchmark-aware query probes** that keep the lexical scorer but strengthen the query—a ground-truth-guided upper bound on BCB-Hard (39.9%) and a task-metadata probe on τ^2 -airline (60.0%, McNemar $p=1.000$)—are also null.

These first two axes both make retrieval fire *more* often, yet the failure mode we identify is the opposite one: the agent invokes every retrieved tool with zero selectivity (Appendix B.9). Our third axis therefore makes retrieval *more precise*. We add an abstention floor to the embedding scorer—returning no tool when the top candidate’s cosine similarity falls below $\tau = 0.55$ —so the retriever declines on 73–92% of tasks and reuse falls from 81–91% to 8–25%. Across four models spanning three providers, this selective retriever still never beats Direct (Table 8): the within-route Create+Reuse–Direct gap is +0.0pp for Opus 4.5, –1.4pp for Sonnet 4.5, and +0.0pp for Gemini 2.5 Pro and GPT-5 Mini.¹ Removing the over-retrieval does not expose any reuse benefit that low selectivity had been masking.

Neither a stronger scorer, stronger queries, nor a more selective retriever overturns the persistence null (Appendix B.2). Nor does a dependency-aware retriever that indexes tools by typed input/output signatures and is warm-started with the entire library available from the first task (41.9% vs. 39.2% Create+Discard on BCB-Hard, $p=0.424$; Appendix B.3). Library management ablations and harness design details appear in Appendices B.4 and D.

¹These runs are served through OpenRouter rather than the provider route used for our main experiments, so absolute accuracies are not comparable to Table 2, and we report each model’s difference *within* the OpenRouter runs. On BCB-Hard, Direct and Create+Discard differ by under 1pp (Table 1), so Direct is a conservative stand-in for the no-persistence baseline.

7 Discussion

A reasoning scaffold with benchmark- and model-dependent effects. Tool creation is a reasoning intervention whose effect depends on both the task distribution and the model’s baseline capability. On BigCodeBench-Hard (low recurrence), it behaves like noise. On τ^2 -airline (high recurrence), the scaffold gradient parallels the code-as-reasoning paradigm of PAL (Gao et al., 2023) and Program of Thoughts (Chen et al., 2023): structured reasoning helps, executable code helps further, but neither produces reusable artifacts that benefit future tasks. A prompt-only control (“Direct-Active,” 56.0%) suggests much of the scaffold effect is engagement priming rather than the design/implement/verify structure (Appendix C.5).

Practical recommendations. In the evaluated settings: create-and-discard is sufficient on low-recurrence tasks, and on recurring tasks persistent libraries added nothing under lexical, embedding, benchmark-aware, or dependency-aware retrieval. Tool-creation prompting can boost weaker models, but at $2\text{--}5\times$ the token cost of Direct. The controls we recommend for future evaluations are listed in §8.

Limitations.

- **Statistical power and variance:** BCB-Hard ($n=148$) is powered to detect effects $>13\text{pp}$, and three-run replication ($\sigma=0.7\text{--}2.2\text{pp}$) confirms the null band reflects task-level stability. On τ^2 ($n=50$), within-family Holm–Bonferroni correction preserves Direct–Discard and Direct–Reuse significance, but under a global correction across all 16 tests no comparison survives. We treat τ^2 analyses as exploratory given the fixed task order and closely matched (not identical) wrapper.
- **Model and benchmark coverage:** four models across three families, and the inverse-correlation finding rests on four data points. Two benchmarks (code generation, customer service) leave math, web navigation, and embodied planning untested.
- **Retrieval quality:** we test a stronger scorer (text-embedding-3-small), two benchmark-aware query probes, a selective variant with an abstention floor (Table 8), and a dependency-aware retriever over typed signatures warm-started with the full library (Appendix B.3). None overturns the persistence null. Learned re-ranking and hierarchical search remain untested.

8 Conclusion

Our controlled ablations across two benchmarks, four models, three retrieval scorers (lexical, embedding-based, and dependency-aware), and two benchmark-aware query probes show that *creating* a tool may help reasoning but *keeping* it does not help reuse: over ten contrasts the median Create+Reuse–Create+Discard difference is $+0.4\text{pp}$ and the mean is -0.4pp , with no paired test reaching significance. Tool-creation prompting can help weaker models at $2\text{--}5\times$ the token cost, but persistence adds nothing under any retrieval scheme we tested. Evaluations of tool-creation systems should report the controls below, none of which prior work has included.

REPORTING CHECKLIST FOR TOOL-CREATION EVALUATIONS

1. **Create-and-discard.** Rerun with the tool discarded, isolating elicitation from reuse.
2. **Harness-matched baseline.** Route both arms through one wrapper. **10pp**
3. **Natural reuse rate.** How often a persisted tool is invoked, and whether it helps. **81–91%**
4. **Token-normalized accuracy.** Tokens per task beside solve rate. **$2.9\text{--}4.7\times$**

Right column: what each control revealed in our own experiments.

Reproducibility statement

All experiments use publicly available benchmarks. Exact model identifiers, decoding settings, retrieval hyperparameters (including the embedding model used for the semantic-retrieval variant), and compute totals appear in Appendices A.1–C.2. The experiment harness and the archived Inspect logs backing every reported table are released at <https://github.com/NaiveNeuron/winnow-colm-2026>.

Acknowledgement

This work was partially funded by European Union, under the project lorAI - Low Resource Artificial Intelligence, GA No. 101136646.

This research was partially supported by ALFIE - Assessment of Learning technologies and Frameworks for Intelligent and Ethical AI, a project funded by European Union under the Horizon Europe, GA No. 101177912.

It was also partially funded by the EU NextGenerationEU through the Recovery and Resilience Plan for Slovakia under the project No. 09I02-03-V01-00029.

References

- Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL <https://openreview.net/forum?id=qV83K9d5WB>.
- Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Trans. Mach. Learn. Res.*, 2023, 2023. URL <https://openreview.net/forum?id=YfZ4ZPt8zd>.
- Kevin Ellis, Catherine Wong, Maxwell I. Nye, Mathias Sablé-Meyer, Lucas Morales, Luke B. Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *PLDI ’21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, pp. 835–850. ACM, 2021. doi: 10.1145/3453483.3454080. URL <https://doi.org/10.1145/3453483.3454080>.
- Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *International Conference on Machine Learning, ICML 2023, Honolulu, Hawaii, USA, volume 202 of Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 2023. URL <https://proceedings.mlr.press/v202/gao23f.html>.
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL <https://openreview.net/forum?id=chfJJYC3iL>.
- Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-Tau Yih, Daniel Fried, Sida I. Wang, and Tao Yu. DS-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning, ICML 2023, Honolulu, Hawaii, USA, volume 202 of Proceedings of Machine Learning Research*, pp. 18319–18345. PMLR, 2023. URL <https://proceedings.mlr.press/v202/lai23b.html>.
- Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, et al. SkillsBench: Benchmarking how well agent skills work across diverse tasks, 2026a. URL <https://arxiv.org/abs/2602.12670v1>. Version 1. The self-generated-Skills condition discussed here was removed in later revisions.

- Xiaoxiao Li. When single-agent with skills replace multi-agent systems and when they fail. *CoRR*, abs/2601.04748, 2026. doi: 10.48550/ARXIV.2601.04748. URL <https://doi.org/10.48550/arXiv.2601.04748>.
- Yu Li, Rui Miao, Zhengling Qi, and Tian Lan. ARISE: Agent reasoning with intrinsic skill evolution in hierarchical reinforcement learning, 2026b. URL <https://arxiv.org/abs/2603.16060>.
- Xukun Liu, Zhiyuan Peng, Xiaoyuan Yi, Xing Xie, Lirong Xiang, Yuchen Liu, and Dongkuan Xu. ToolNet: connecting large language models with massive tools via tool graph, 2024a. URL <https://arxiv.org/abs/2403.00839>.
- Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Ziheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, and Wenhai Wang. ControlLLM: augment language models with tools by searching on graphs. In *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29 - October 4, 2024, Proceedings, Part XII*, volume 15070 of *Lecture Notes in Computer Science*, pp. 89–105. Springer, 2024b. doi: 10.1007/978-3-031-73254-6_6. URL https://doi.org/10.1007/978-3-031-73254-6_6.
- Zhiyuan Ma, Zhenya Huang, Jiayu Liu, Minmao Wang, Hongke Zhao, and Xin Li. Automated creation of reusable and diverse toolsets for enhancing LLM reasoning. In Toby Walsh, Julie Shah, and Zico Kolter (eds.), *Thirty-Ninth AAAI Conference on Artificial Intelligence, AAAI 2025, Philadelphia, PA, USA, February 25 - March 4, 2025*, pp. 24821–24830. AAAI Press, 2025. doi: 10.1609/AAAI.V39I23.34664. URL <https://doi.org/10.1609/aaai.v39i23.34664>.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegraffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36, NeurIPS 2023, New Orleans, LA, USA, December 10-16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.
- Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter A. Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. CLIN: A continually learning language agent for rapid task adaptation and generalization. *CoRR*, abs/2310.10134, 2023. doi: 10.48550/ARXIV.2310.10134. URL <https://doi.org/10.48550/arXiv.2310.10134>.
- Steven Minton. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42(2–3):363–391, 1990. doi: 10.1016/0004-3702(90)90059-9. URL [https://doi.org/10.1016/0004-3702\(90\)90059-9](https://doi.org/10.1016/0004-3702(90)90059-9).
- Cheng Qian, Chi Han, Yi Ren Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. CREATOR: tool creation for disentangling abstract and concrete reasoning of large language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023, Findings of ACL*, pp. 6922–6939. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EMNLP.462. URL <https://doi.org/10.18653/v1/2023.findings-emnlp.462>.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36, NeurIPS 2023, New Orleans, LA, USA, December 10-16, 2023*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.
- Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *Proceedings of the 40th International Conference on Machine Learning, ICML 2023*,

- volume 202 of *Proceedings of Machine Learning Research*, pp. 31210–31227. PMLR, 2023. URL <https://proceedings.mlr.press/v202/shi23a.html>.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.
- Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL <https://openreview.net/forum?id=4FWAwZtd2n>.
- UK AI Security Institute. Inspect AI: A framework for large language model evaluations, 2024. URL <https://inspect.aisi.org.uk/>. Open-source evaluation framework. https://github.com/UKGovernmentBEIS/inspect_ai.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Trans. Mach. Learn. Res.*, 2024, 2024. URL <https://openreview.net/forum?id=ehfRiF0R3a>.
- Jiong Xiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library, 2025a. URL <https://arxiv.org/abs/2512.17102>.
- Zhenting Wang, Huancheng Chen, Jiayun Wang, and Wei Wei. Memex(RL): Scaling long-horizon LLM agents via indexed experience memory, 2026. URL <https://arxiv.org/abs/2603.04257>.
- Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. *CoRR*, abs/2504.06821, 2025b. doi: 10.48550/ARXIV.2504.06821. URL <https://doi.org/10.48550/arXiv.2504.06821>.
- Georg Wölflein, Dyke Ferber, Daniel Truhn, Ognjen Arandjelovic, and Jakob Nikolas Kather. LLM agents making agent tools. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 26092–26130. Association for Computational Linguistics, 2025. URL <https://aclanthology.org/2025.acl-long.1266/>.
- Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning, 2026. URL <https://arxiv.org/abs/2602.08234>.
- Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Zirui Liu, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. How memory management impacts LLM agents: An empirical study of experience-following behavior, 2025. URL <https://arxiv.org/abs/2505.16067>.
- Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward, 2026. URL <https://arxiv.org/abs/2602.12430>. Accepted at the Agent Skills ’26 Workshop, ACM Conference on AI and Agentic Systems 2026.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

- Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R. Narasimhan. τ -bench: A benchmark for tool-agent-user interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL <https://openreview.net/forum?id=roNSXZpUDN>.
- Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi Fung, Hao Peng, and Heng Ji. CRAFT: customizing llms by creating and retrieving from specialized toolsets. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL <https://openreview.net/forum?id=G0vdDSt9XM>.
- Murong Yue, Zhiwei Liu, Liangwei Yang, Jianguo Zhang, Zuxin Liu, Haolin Chen, Ziyu Yao, Silvio Savarese, Caiming Xiong, Shelby Heinecke, and Huan Wang. ToolLibGen: Scalable automatic tool creation and aggregation for LLM reasoning, 2025. URL <https://arxiv.org/abs/2510.07768>.
- Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (eds.), *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Vancouver, Canada, February 20-27, 2024*, pp. 19632–19642. AAAI Press, 2024a. doi: 10.1609/AAAI.V38I17.29936. URL <https://doi.org/10.1609/aaai.v38i17.29936>.
- Haiteng Zhao, Chang Ma, Guoyin Wang, Jing Su, Lingpeng Kong, Jingjing Xu, Zhi-Hong Deng, and Hongxia Yang. Empowering large language model agents through action learning. *CoRR*, abs/2402.15809, 2024b. doi: 10.48550/ARXIV.2402.15809. URL <https://doi.org/10.48550/arXiv.2402.15809>.
- Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62138–62160. PMLR, 2024. URL <https://proceedings.mlr.press/v235/zhou24r.html>.
- Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widayarsi, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, James Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, and et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL <https://openreview.net/forum?id=YrycTjllL0>.

A Experimental details

A.1 Hyperparameters

Table 4 lists all hyperparameters for the reported runs.

Exact model identifiers. The archived runs use the following provider identifiers: anthropic.claude-sonnet-4-5-20250929-v1:0, gpt-5-mini, gemini-2.5-pro, and o4-mini-global. Table 2 additionally reports GPT-5.4 (release 2026-03-05) on that same route, and the selective-retrieval runs of §6.1 add Opus 4.5 served through OpenRouter; exact provider strings for every run are recorded in the released run metadata. In the main text we use the shorter display names Sonnet 4.5, GPT-5 Mini, Gemini 2.5 Pro, o4-mini, GPT-5.4, and Opus 4.5, and tables use these display names throughout.

Parameter	Value	Notes
LLM (primary)	Claude Sonnet 4.5	Anthropic, \$3/\$15 per M tokens
LLM (secondary)	GPT-5 Mini	OpenAI, \$0.25/\$2 per M tokens
LLM (third family)	Gemini 2.5 Pro	Google, \$1.25/\$10 per M tokens
LLM (reasoning)	o4-mini	OpenAI, \$1.10/\$4.40 per M tokens
Temperature (BCB-Hard)	provider default	No override; stochastic sampling
Temperature (τ^2 -airline)	0	Deterministic decoding (explicit)
τ^2 message limit	30	Max simulated conversation turns per sample
Retrieval scorer (default)	lexical cosine	Bag-of-words term-frequency cosine (no IDF)
Retrieval scorer (embedding)	text-embedding-3-small	OpenAI, 1536-dim embeddings
Retrieval text (default)	name + summary + body	Shared by retrieval and admission similarity
Retrieval top- k	3	Number of tools/trajectories retrieved
Inspect AI version	0.3.200	Version recorded in archived eval bundles
τ^2 task version	2-A	Archived eval metadata
τ^2 comparability version	2	Archived eval metadata
Creation-time verification	syntax check	Standard create_tool validation in current runs
Optional held-out verification	supported but unused	Hidden-case execution path for admission-controlled conditions
Tool revision attempts	agent-controlled	Additional create_tool calls allowed within sample budget
Admission distance δ	0.3 default	Clean reported Create+Admit rows use 0.3; Exp. 3 sweeps 0.50 and 0.90

Table 4: Full hyperparameter specification for the current harness and reported runs.

A.2 Compute costs

Table 5 summarizes observed compute usage.

Experiment	Asst. turns	Input (M)	Cached Input (M)	Output (M)	Reason. (M)	Total (M)	Est. cost (USD)
Exp. 1	6232	7.38	0.92	4.31	4.58	17.19	78.53
Exp. 2	2235	37.77	0.00	0.77	0.00	38.55	124.90
Exp. 3	8104	14.26	0.00	3.08	0.00	17.34	89.03

Table 5: Compute totals for the reported experiment bundles: Exp. 1 is the completed BigCodeBench-Hard sweep bundle (20 runs), Exp. 2 is the clean tau2-airline rerun bundle (7 runs), and Exp. 3 is the BigCodeBench-Hard admission and library-management sweep bundle (18 runs). Assistant-turn counts and explicit input, cached-input, output, and reasoning token totals are measured directly from the archived Inspect traces. Estimated USD uses contemporaneous public list prices for each model family, with reasoning tokens charged at the output-token rate.

A.3 Pilot validation

Pilot 1: Headroom check. We evaluate the Sonnet variant anthropic.claude-sonnet-4-5-20250929-v1:0 on BigCodeBench-Hard (148 tasks) using the **Direct** solver within Inspect AI (UK AI Security Institute, 2024). This simultaneously validates the harness setup (Docker sandboxing, scorer correctness) and measures the

baseline pass rate. The viable range is 40–65%; our pilot yields 38.5%, slightly below but leaving substantial headroom.

Pilot 2: Tool creation feasibility. We run **Create+Reuse** on a 30-task subsample of BigCodeBench-Hard. The condition solved 16/30 tasks (53.3%). Every sample queried `retrieve_tool`, 20/30 received at least one stored-tool hit, and all 20 hit-bearing samples invoked a prior-task tool before proceeding.

B BCB-Hard supplementary results

B.1 Three-run replication

To verify that the null band is not an artifact of single-run noise, we repeat the four key conditions (Direct, Create+Discard, Create+Invoke_{once}, Create+Reuse) three times on BCB-Hard with Sonnet using the same provider-default settings. The run-to-run standard deviation is 0.7–2.2pp across conditions (Table 6), confirming that the ~ 2 pp cross-condition band is comparable to the within-condition noise floor. Critically, the mean values across three runs preserve the same pattern: all four conditions fall within a 2.5pp band (38.7–41.2%), and Create+Reuse ($39.9\% \pm 1.8$) never clearly exceeds Create+Discard ($41.2\% \pm 0.7$).

Condition	Mean	σ	Range	Δ vs. Direct
Direct	40.1	1.6	39.2–41.9	—
Create+Discard _{full}	41.2	0.7	40.5–41.9	+1.1pp
Create+Invoke _{once}	38.7	2.2	37.2–41.2	−1.4pp
Create+Reuse	39.9	1.8	38.5–41.9	−0.2pp

Table 6: **Three-run replication on BCB-Hard (Sonnet, 148 tasks per run).** Mean solve rate (%), standard deviation, and range across three independent runs with provider-default generation parameters. The null band holds across replications.

B.2 Retrieval variant probes

The failure of standard persistence could reflect either (a) tools are not useful for reuse, or (b) the default retriever fails to surface the right tool. We probe this along two axes: the *similarity scorer* (embedding-based vs. lexical) and *query construction* (benchmark-aware vs. agent-generated).

Embedding-based retrieval. We replace the lexical cosine scorer with text-embedding-3-small (OpenAI, 1536-dimensional embeddings), keeping all other harness components identical. On BCB-Hard, Create+Reuse with embedding retrieval scores 40.5% (vs. 39.9% lexical); on τ^2 -airline it scores 58.0% (vs. 60.0% lexical). Both results fall within the existing noise band and do not alter the persistence conclusion. This rules out the concern that the lexical retriever alone explains the null result.

Benchmark-specific probes. On BCB-Hard, the retriever queries with benchmark target text (including the reference solution) unavailable to the agent. On τ^2 -airline, the retriever uses task metadata as the query and upweights source text in the lookup representation.

Selective retrieval (abstention floor). The probes above all make retrieval fire more broadly; a complementary question is whether making it *more selective* helps. We add a minimum-similarity floor ($\tau = 0.55$) to the embedding scorer so it returns no tool when the top candidate is not a close match, and re-run Create+Reuse against a matched Direct baseline on four models (Table 8). The floor makes the retriever abstain on 73–92% of tasks, collapsing the reuse rate from the default 81–91% to 8–25%, yet the persistence null holds at every capability tier, including frontier Opus 4.5. Because these runs are served

Benchmark	Retrieval variant	Solve (%)	Interpretation
BCB-Hard	Embedding (text-embedding-3-small)	40.5	Within noise of lexical Create+Reuse (39.9%)
BCB-Hard	Ground-truth-guided target-text query	39.9	Within noise of standard Create+Reuse
τ^2 -airline	Embedding (text-embedding-3-small)	58.0	Within noise of lexical Create+Reuse (60.0%)
τ^2 -airline	Task-metadata query + source-text lookup	60.0	Tied with standard Create+Reuse

Table 7: **Retrieval variant probes on Sonnet.** Embedding-based retrieval does not materially change Create+Reuse performance on either benchmark. On BCB-Hard, even a ground-truth-guided upper bound (99.3% reuse rate, 100% precision) does not improve over standard Create+Reuse. On τ^2 -airline, both the embedding and benchmark-aware variants stay within the existing noise band (paired oracle-use: 0.0pp $[-14.0, 14.0]$, $p=1.000$).

through OpenRouter rather than the provider route of the main experiments, we report the within-OpenRouter Create+Reuse–Direct difference rather than absolute accuracies; on BCB-Hard Direct is within 1pp of Create+Discard (Table 1) and so is a conservative stand-in for the no-persistence baseline.

Model	Direct (%)	Create+Reuse @ $\tau=0.55$ (%)	Δ (pp)	Reuse (%)	Abstain (%)
Opus 4.5 (frontier)	44.6	44.6	+0.0	25	75
Sonnet 4.5	40.5	39.2	−1.4	20	80
Gemini 2.5 Pro	29.1	29.1	+0.0	20	73
GPT-5 Mini	33.8	33.8	+0.0	8	92

Table 8: **Selective (abstaining) embedding retrieval on BCB-Hard.** Adding a similarity floor ($\tau=0.55$) to the embedding scorer makes the retriever decline to return a tool on 73–92% of tasks, cutting the reuse rate from the 81–91% observed under the default retriever to 8–25% (*Reuse*: fraction of tasks that invoke a persisted tool; *Abstain*: fraction on which retrieval returns nothing). Even so, Create+Reuse never beats Direct at any tier, including frontier Opus 4.5. These runs are served through OpenRouter, Δ is the Create+Reuse–Direct difference within them, and it is not comparable to the absolute numbers in Table 2. All eight runs (four Direct, four Create+Reuse) completed 148/148 tasks with no errored samples.

B.3 Dependency-aware retrieval with a warm-started library

The probes in Appendix B.2 vary the similarity scorer, the query, and the retriever’s selectivity, but they leave two design choices fixed: the library is indexed as free text (name, summary, body) and it grows *online* as the run proceeds. Graph-based tool-selection systems such as ControlLLM (Liu et al., 2024b) and ToolNet (Liu et al., 2024a) instead prune the candidate set by the *dependency structure* among typed tool signatures and can operate over a library that is fully available from the first task. We implement a retriever in this family and pair it with a warm-start protocol that removes the online-ordering confound, then test whether either change overturns the persistence null on Sonnet.

Structured tool signatures. When a tool enters the library we extract, in addition to its name/summary/body, a typed signature: input and output types, the state it *requires* before it can run, the state it *produces*, and, on τ^2 -airline, the domain API endpoints it chains (lookup $\rightarrow$ modify $\rightarrow$ confirm). This replaces the free-text index with the structured description a dependency-aware retriever consumes.

Progress-aware retrieval. At each step we infer the available state and current subgoal from the trajectory so far, restrict candidates to tools whose required state is already satisfied (the dependency pruning), and score the survivors by a weighted sum of dependency match, subgoal advancement, and semantic similarity. A top candidate below the abstention threshold triggers an explicit “no relevant tool” action rather than a forced invocation.

Warm-start and preloaded-library protocols. To give reuse its best case and neutralize task ordering, we run Create+Reuse once to populate the library (Pass 1, whose accuracy we discard as confounded by online growth), freeze the resulting library, and re-run from task 1 with the *complete* library available and the progress-aware retriever above (Pass 2). A simpler *preloaded-library* control freezes the same Pass-1 library but keeps the default lexical retriever, separating the effect of full availability from the effect of dependency-aware scoring.

Results. Table 9 reports Pass 2. On BCB-Hard the dependency-aware, warm-started retriever scores 41.9% (62/148) against Create+Discard at 39.2% and lexical Create+Reuse at 39.9%—a +2.7pp difference over Discard that is not significant (paired two-sided exact $p=0.424$; 9 wins, 5 losses, 134 ties) and whose bootstrap interval (−2.0 to +7.4pp) spans zero. The preloaded-library control scores 39.9% (+0.7pp over Discard, $p=1.000$). On τ^2 -airline the warm-started retriever scores 58.0%, exactly matching Create+Discard and below lexical Create+Reuse (60.0%), with the preloaded control at 56.0%. Neither full library availability nor dependency-aware scoring moves either benchmark outside the null band established by the simpler retrievers.

Benchmark	Condition	Solve (%)	Δ vs Discard (pp)	p	Reuse (%)	Tokens /task
BCB-Hard	Create+Discard (baseline)	39.2	—	—	—	—
BCB-Hard	Dependency, warm-start	41.9	+2.7	0.424	90.5	6360
BCB-Hard	Preloaded, lexical	39.9	+0.7	1.000	91.2	6301
τ^2 -airline	Create+Discard (baseline)	58.0	—	—	—	—
τ^2 -airline	Dependency, warm-start	58.0	+0.0	1.000	—	128,474
τ^2 -airline	Preloaded, lexical	56.0	−2.0	1.000	—	133,039

Table 9: **Dependency-aware, warm-started retrieval on Sonnet.** Pass-2 accuracy after the library is frozen and made fully available from the first task. *Dependency, warm-start* uses the progress-aware retriever over typed tool signatures; *Preloaded, lexical* keeps the default lexical scorer over the same frozen library. Δ is against the scaffold-matched Create+Discard baseline (paired two-sided exact p); *Reuse* is the fraction of BCB-Hard tasks that invoke a persisted tool. Neither condition beats Create+Discard on either benchmark, and the bootstrap 95% CI for the dependency − Discard delta spans zero on both (−2.0 to +7.4pp on BCB-Hard; −12.0 to +12.0pp on τ^2 -airline). Both BCB-Hard conditions completed 148/148 tasks; both τ^2 conditions completed 50/50.

Retrieval fires but does not transfer. The mechanism counters explain the flat result. On BCB-Hard the frozen library is surfaced on 99.3% of tasks and a persisted tool is actually invoked on 90.5%—so surfacing and reuse are not the bottleneck—yet only 32.1% of those invocations are scored as useful to the solution (retrieval-use precision), essentially unchanged from the default retriever. Indexing the library by dependency structure and making it fully available surfaces *more* tools without making them more transferable, consistent with this paper’s account that the binding constraint is task diversity rather than retrieval quality (§6.1).

B.4 Library management

If the problem with tool persistence is library bloat rather than persistence per se, then curation—controlling *which* tools enter and how many are kept—might recover value. We test this on BCB-Hard (Sonnet) in a *separate set of runs* from Experiment 1, with two axes of library management. (The ungated Create+Reuse baseline in these runs achieves 40.5%, slightly above the 39.9% from the Experiment 1 run, consistent with the single-run variance discussed in the limitations.)

Admission control (distance threshold). Create+Admit gates tool entry on a minimum cosine-distance threshold δ (Table 10): a new tool is admitted only if it is at least δ distant from every existing tool in the current lexical-similarity space. We test two thresholds:

δ	Solve rate (%)	Final library size	Tools rejected
0.50	41.2	9	0
0.90	39.2	3	22

Table 10: **Admission-distance ablation on BCB-Hard (Sonnet, separate runs from Experiment 1).** At $\delta=0.50$ the gate is permissive and nearly all tools are admitted (9 tools, 0 rejected); performance (41.2%) is comparable to the ungated baseline in these runs (40.5%, Table 11). At $\delta=0.90$ the gate is aggressive (3 tools, 22 rejected) and performance drops to 39.2%.

Bounded-library strategies. We also test eviction policies that cap library size, preventing unbounded growth:

Strategy	Solve rate (%)	Reuse precision	Library size	Tools admitted	Tools evicted
Ungated (Create+Reuse)	40.5	40.6%	10	10	0
FIFO (cap=10)	41.9	40.2%	8	8	0
LRU (cap=10)	43.2	42.3%	10	10	0
FIFO (cap=5)	41.2	48.2%	5	13	8
LRU (cap=5)	41.9	43.9%	5	12	7

Table 11: **Library management strategies on BCB-Hard (Sonnet).** LRU (cap=10) achieves 43.2%—the highest solve rate in the Experiment 3 batch and 2.7pp above ungated Create+Reuse. Tighter caps (cap=5) improve reuse precision (48.2% for FIFO) but do not further improve solve rate. All strategies remain within the broader noise band of the experiment.

The LRU (cap=10) result is suggestive: a small, curated library may perform slightly better than an uncurated one, consistent with the classical utility problem (Minton, 1990) where a leaner operator set reduces search overhead. However, the 2.7pp improvement over ungated Create+Reuse does not reach significance at $n=148$ (within the ~ 8 pp CI half-width). Within this Experiment 3 batch, it exceeds the matched ungated Create+Reuse baseline (40.5%) but does not yet establish a clear advantage over the broader reasoning-first story from Experiment 1. Library management can mitigate degradation from bloat, but our current evidence does not show that it reliably unlocks value that was otherwise unavailable.

B.5 Condition-flip summary

The majority of BCB-Hard tasks are unaffected by condition: 86 tasks are too hard for Sonnet under any condition (all three fail), and 40 are easy enough to pass regardless. The remaining 22 tasks—those whose outcome changes between at least two of Direct, Create+Discard, and Create+Reuse—are where the experimental manipulation actually matters. Table 12 categorizes these flips.

Flip category	Count	Direction
Reasoning helped (Discard pass, Direct fail)	8	+
Reuse helped (Reuse pass, Direct fail)	10	+
Creation hurt (Direct pass, Discard fail)	5	−
Retrieval hurt (Discard pass, Reuse fail)	7	−

Table 12: **Task-level outcome flips on BCB-Hard (Sonnet, 148 tasks).** Retrieval anchoring (7 tasks) partially offsets the reasoning benefit (8 tasks).

What reasoning elicitation looks like in practice. Task 458 (JSON normalization) fails in Direct but passes in Create+Discard: designing `normalize_dict_values` forces the model to enumerate input types explicitly. Task 587 (AES encryption) follows the same pattern—designing `aes_encrypt_file` forces correct padding and key-derivation reasoning. In both cases the discarded tool’s design step, not its code, drove the improvement.

B.6 Retrieval anchoring case studies

Of the seven “retrieval hurt” flips in Table 12, five share a common pattern: the task is straightforward enough for Sonnet to solve without assistance, but the retrieval pipeline injects a tool whose keyword overlap misleads the model’s approach. We call this *retrieval anchoring*—the model treats the retrieved tool as an authoritative starting point rather than evaluating whether it is functionally relevant. In each case below, the retrieval query (derived from the task description) has high cosine similarity to a stored tool’s summary in text-embedding-3-large space, yet the tool solves an unrelated problem:

- **Task 532** (histogram + normal distribution): Query “count duplicate values DataFrame histogram normal distribution”; retrieves `fft_signal_processor_0012` (from task 267) because “distribution” is near “signal processing” in embedding space. The model anchors on FFT logic.
- **Task 637** (student grades): Query “generate student grades dataframe courses average”; retrieves `kmeans_cluster_plot_0007` (from task 92) due to “dataframe” keyword overlap.
- **Task 760** (random DataFrame with names/dates): Retrieves `train_rf_plot_importance_0004` (from task 37).
- **Task 916** (boxplot + histogram visualization): Retrieves `train_rf_plot_importance_0004`.
- **Task 917** (ARIMA forecasting): Retrieves `fft_signal_processor_0012`; ARIMA and FFT both process temporal data but are algorithmically unrelated.

B.7 o4-mini compliance analysis

Table 13 splits o4-mini’s Create+Discard results by whether the model actually called `create_tool`. The 56 skipped tasks are ones o4-mini solves well in Direct (48.2% baseline); the extra system prompt text appears to perturb the model’s hidden chain-of-thought without providing compensating structure.

o4-mini subset	Tasks	Direct	Discard	Δ	Net flips
Created tool (5+ msgs)	92	33.7%	37.0%	+3.3pp	+3
Skipped tool (3 msgs)	56	48.2%	32.1%	−16.1pp	−9
All tasks	148	39.2%	35.1%	−4.0pp	−6

Table 13: **o4-mini compliance split on BCB-Hard Create+Discard.** When o4-mini follows the tool-creation instruction, it benefits (+3.3pp); when it skips, performance drops sharply (−16.1pp). The aggregate −4.0pp masks opposite effects.

B.8 Cross-model per-task flips

The aggregate solve-rate deltas (§5) mask substantial per-task churn: many tasks flip in both directions, with the net effect being a small residual. Table 14 decomposes each model’s 148 tasks into four outcome categories. For all models, the “both fail” category dominates (79–87 tasks), confirming that a large core of BCB-Hard tasks resists all conditions. The interesting signal is in the asymmetry between “Direct only” (tasks hurt by tool creation) and “Discard only” (tasks helped). Sonnet shows a clean +2 net (6 hurt, 8 helped); GPT-5 Mini and Gemini are roughly balanced; o4-mini is the outlier with 17 hurt vs. 11 helped, driven by the compliance issue detailed in §B.7.

Model	Both pass	Both fail	Direct only	Discard only
Sonnet	49	86	6	8
GPT-5 Mini	37	87	11	13
Gemini	41	82	13	12
o4-mini	41	79	17	11

Table 14: **Per-model task concordance on BCB-Hard (Direct vs. Create+Discard, 148 tasks each).** o4-mini has the highest “Direct only” count (17 tasks hurt), 9 of which are unique to o4-mini.

B.9 Retrieval precision

The Create+Reuse pipeline has three stages where retrieval could fail to help: (1) no tool is returned (miss), (2) a tool is returned but the model ignores it, or (3) a tool is returned and used but does not improve the outcome. Table 15 shows that stage (2) never occurs—the model invokes a retrieved tool on every single hit, almost always picking the top-ranked candidate (96% position 0) with no evaluation of alternatives. The bottleneck is entirely at stage (3): invocation does not help.

Comparing Create+Reuse to Create+Discard on the 135 hit tasks isolates the pure retrieval effect: 6 tasks are hurt by the retrieved tool, 7 are helped, yielding a net of +1 task (+0.7pp)—statistically indistinguishable from zero.

	Solved	Failed	Total
Hit + Invoked	54	81	135
Hit + Not invoked	0	0	0
No hit	5	8	13
Total	59	89	148

Table 15: **Retrieval precision matrix for Create+Reuse (Sonnet, BCB-Hard).** Solve rate with retrieval ($54/135 = 40.0\%$) is essentially identical to solve rate without ($5/13 = 38.5\%$).

B.10 Reuse failure-mode audit

The precision matrix above shows that invoking a retrieved tool does not raise the solve rate; this audit asks *why*, on the subset of tasks where reuse actually fires under the selective ($\tau = 0.55$) retriever of §6.1 (8–25% of tasks per model, 108 events in total). One signal is objective: although the harness explicitly instructs the agent to “reuse or adapt” the retrieved tool and forbids creating a new one on that turn, the submitted solution actually incorporates the tool’s logic in only 47 of the 108 cases (44%)—more than half the time the agent inspects the tool and then writes an independent solution.

To characterize the outcomes we classify each of the 108 events—from the task specification, the full body of the invoked tool, the submitted solution, and the pass/fail result—into

four mutually exclusive categories: the retrieved tool is functionally *irrelevant* (shares only surface keywords with the task); *relevant* and the task is *solved*; relevant but failed because the *tool's own logic* is wrong or mis-abstracted for the task (a better tool would plausibly have helped); or relevant but failed because the *task is hard* for reasons the tool does not address. The classification is LLM-assisted with a fixed rubric and was spot-validated against the raw traces; the quality-versus-task-hard boundary is the most subjective and should be read as approximate.

Model (reuses)	Irrelevant tool	Relevant, solved	Rel., failed (quality)	Rel., failed (task hard)	Used tool
Sonnet 4.5 (30)	8	7	1	14	11
Gemini 2.5 Pro (29)	11	6	2	10	13
Opus 4.5 (37)	6	17	1	13	15
GPT-5 Mini (12)	4	4	0	4	8
Total (108)	29 (27%)	34 (31%)	4 (4%)	41 (38%)	47 (44%)

Table 16: **Failure-mode audit of the 108 reuse events under selective ($\tau = 0.55$) retrieval.** The first four columns partition each reuse and sum to the per-model count; *Used tool* is an orthogonal count of how often the submitted solution actually incorporated the invoked tool. Only 4% of reuses fail because of tool quality; functional irrelevance (27%) and task difficulty (38%) dominate.

Only 4 of 108 reuses (4%) fail because of tool quality: the created tools are competent code—all pass verification—and when one fails to help it is almost never because it is buggy. The dominant modes are functional *irrelevance* (27%, where the retriever surfaces a keyword-similar but operationally different tool) and *task difficulty* the tool does not address (38%). Genuine transfer—a relevant tool actually used and decisive for the solve—is rare; most “relevant, solved” cases are solved independently of the tool. The bottleneck is therefore reusability and match, not tool quality: created tools are locally competent but task-specific, and seldom fit the problems that come next.

B.11 Retrieval magnets

The Create+Reuse library saturates at 13 tools after the first 13 sequentially processed tasks; subsequent tasks always find an existing tool within the admission-control similarity threshold. Three tools dominate all retrievals:

- `kmeans_cluster_plot`: retrieved by 61 tasks (41%)
- `pca_transform_plot`: retrieved by 51 tasks (34%)
- `execute_commands_from_csv`: retrieved by 51 tasks (34%)

These tools surface because their names contain common data-science keywords (“DataFrame,” “plot,” “CSV”) that have high embedding similarity to many task descriptions, not because they are functionally relevant. Per-tool solve rates hover between 29% and 46%, indistinguishable from the 40% overall baseline.

Magnets are an artifact of un-thresholded retrieval. The magnet effect above is measured under a retriever that always returns its top candidate. Isolating the abstention floor of §6.1 on the embedding scorer—holding the scorer and provider fixed and varying only τ —shows the magnets are a mechanical consequence of never abstaining rather than evidence of transfer. At $\tau = 0.0$ the concentration is extreme: for Sonnet a *single* created tool is retrieved for 99% of the 148 tasks, and for Gemini the three most-retrieved tools account for 54% of all retrievals, so a persisted tool is invoked on 91–99% of tasks. Adding the $\tau = 0.55$ floor spreads retrieval across 19–28 distinct tools, none exceeding 4% of tasks, and cuts reuse to 20% (Table 17). Solve rate changes by at most 3.4pp across this swing—within the noise band—so dismantling the magnets neither helps nor hurts: the dominant tools are

surfaced by keyword overlap, invoked without selectivity, and irrelevant to whether the task is solved. (Runs use the OpenRouter provider route; see the within-route caveat for Table 8.)

Model	Retriever	Reuse (% tasks)	Distinct tools surfaced	Top-tool share	Solve (%)
Sonnet 4.5	Embedding, $\tau=0.0$	99	1	99%	42.6
	Embedding, $\tau=0.55$	20	19	4%	39.2
Gemini 2.5 Pro	Embedding, $\tau=0.0$	91	12	52%	32.4
	Embedding, $\tau=0.55$	20	28	2%	29.1

Table 17: **The abstention floor dissolves retrieval magnets (BCB-Hard, OpenRouter route).** Holding the embedding scorer and provider fixed, adding a $\tau = 0.55$ similarity floor collapses the reuse rate from 91–99% to 20% and spreads retrieval from a handful of magnet tools—a single tool retrieved by 99% of Sonnet’s tasks—across 19–28 distinct tools. *Top-tool share* is the fraction of tasks that surface the single most-retrieved tool. Solve rate stays within the noise band throughout, confirming the magnets contribute no transfer.

B.12 Tool taxonomy

Table 18 provides per-condition tool statistics. Library tools (Create+Reuse) are substantially larger than discarded tools (median 35 vs. 15 lines), with richer parameterization (mean 2.3 vs. 2.0 parameters) and universal docstring coverage.

Condition	Tools	Generic	Utility	Task-specific	Med. lines
Create+Reuse	13	12	1	0	35
Create+Discard	148	99	45	4	15
Create+Admit	11	10	1	0	37
Overall	172	121	47	4	17

Table 18: **Tool taxonomy per condition (Sonnet, BCB-Hard).** Heuristic classification based on parameter count, line count, and hardcoded-value detection. Library tools (Create+Reuse, Create+Admit) skew larger and more generic than discarded tools.

C τ^2 -airline supplementary results

C.1 Cross-model results (not wrapper-matched)

Table 19 reports cross-model rows that use the upstream τ^2 Direct agent, *not* wrapper-matched Direct, so the +10pp harness effect identified for Sonnet may apply.

Model	Type	Direct	Discard	Reuse	Reuse– Discard
GPT-5 Mini	mid-range	56.0	42.0	48.0	+6.0pp
Gemini 2.5 Pro	frontier	66.0	54.0	54.0	0.0pp
o4-mini	reasoning	48.0	46.0	48.0	+2.0pp

Table 19: **τ^2 -airline cross-model results (not wrapper-matched).** The Discard–Direct deltas for GPT-5 Mini (–14pp) and Gemini (–12pp) are confounded by the harness asymmetry. Without wrapper-matched baselines for these models, we treat these rows as descriptive only.

GPT-5.4 on a τ^2 slice is likewise flat: Direct 12/20 (60.0%), Create+Discard 10/20 (50.0%), and Create+Reuse 10/20 (50.0%), a Reuse–Discard delta of 0.0pp.

C.2 Per-task results and table provenance

The archived Inspect logs released with this paper (see the Reproducibility statement) provide per-sample solve outcomes, tool-creation events, retrieval events, and admission decisions for the reported experiments. Rather than duplicate those traces in the PDF, we export compact aggregate tables in the paper and keep the underlying sample-level records in that release. On τ^2 -airline, the task factory is called with `shuffle=False`, so memory-based conditions build their libraries online over a fixed benchmark order.

The τ^2 Sonnet rows in Table 3 come from four single-bundle, 50/50 complete reruns: one each for Direct, Create+Discard_{full}, Create+Reuse, and Create+Admit. The separate benchmark-aware retrieval probe in Table 7 comes from one additional single-bundle 50/50 complete Sonnet rerun using the current benchmark-aware query/lookup variant.

For the March 30 cross-model sweep, the Direct and Create+Discard_{full} rows for GPT-5 Mini, Gemini 2.5 Pro, and o4-mini each come from one clean 50/50 eval bundle. The Create+Reuse row is a canonical aggregate after provider failures on the original tail chunks: 5 source bundles for GPT-5 Mini, 6 for Gemini, and 5 for o4-mini. All three aggregated rows cover 50/50 samples with no duplicates and no unresolved samples. When a rerun overlaps an earlier chunk, the canonical export keeps the latest score for that sample id.

C.3 Paired contingency counts

Table 20 reports per-task contingency counts for the clean Sonnet rerun.

Comparison	Both pass	Both fail	1st only	2nd only	McNemar	Bootstrap
Dir. vs. Discard	17	18	3	12	$p=.035$	+18pp [4, 32], $p=.018$
Dir. vs. Reuse	17	17	3	13	$p=.021$	+20pp [6, 34], $p=.010$
Disc. vs. Reuse	25	16	4	5	$p=1.00$	+2pp [−10, 14], $p=.859$

Table 20: **Paired per-task analysis for the clean Sonnet τ^2 rerun (50 common tasks).** Dir. = Direct, Disc. = Create+Discard, Use = Create+Reuse. “1st/2nd only” count discordant tasks where exactly one condition succeeds. Bootstrap deltas are second minus first. The clean rerun supports tool creation over Direct, but not Create+Reuse over Create+Discard.

C.4 Engagement metrics

Table 21 reports per-condition engagement metrics from the archived March 29 run.

Condition	Solve %	Avg turns	Avg API calls	Zero-tool tasks
Direct	38%	2.3	0.6	35/50 (70%)
Create+Discard	52%	9.8	5.1	5/50 (10%)
Create+Reuse	46%	8.9	4.9	—
Create+Admit	38%	7.7	3.9	—

Table 21: **Agent engagement metrics on τ^2 -airline (Sonnet, 50 tasks, earlier archived run).** Solve rates here are from the archived March 29 run (before the clean rerun in Table 3); engagement metrics (turns, API calls, zero-tool counts) are structural properties that are consistent across runs. In Direct, 70% of tasks have zero airline API calls; the agent asks clarifying questions but never acts. Create+Discard reduces zero-tool tasks to 10% and increases average API calls by $8\times$.

C.5 Prompt-only control (Direct-Active)

A prompt-only control, **Direct-Active**, instructs the agent to “always attempt to use your available airline tools” without exposing the create/retrieve/invoke meta-tools or the design/implement/verify structure. Direct-Active reaches 56.0% (28/50), landing between Structured-CoT (54.0%) and Create+Discard (58.0%). The pairwise comparison with Create+Discard has 7 vs. 8 discordant tasks (McNemar $p=1.000$; paired bootstrap +2.0pp [−18.0, 14.0], $p=0.904$).

This suggests that much of the scaffold effect observed in the main text comes from *engagement priming*—simply encouraging the agent to use its available tools—rather than from the specific design/implement/verify structure of tool creation. The full wrapper-matched gradient is:

Condition	Solve %	Δ vs. W-Direct
Wrapper-Direct	50.0	—
Structured-CoT	54.0	+4.0pp
Direct-Active	56.0	+6.0pp
Create+Discard	58.0	+8.0pp
Create+Reuse	60.0	+10.0pp

No pairwise step in this gradient reaches significance at $n=50$. The monotonic pattern is consistent with each additional layer of prompting structure contributing a small increment, but the dominant factor is engagement priming rather than any specific reasoning structure.

C.6 Baseline capability vs. scaffold effect

Figure 3 plots the relationship between each model’s Direct baseline and the Create+Discard effect on τ^2 -airline.

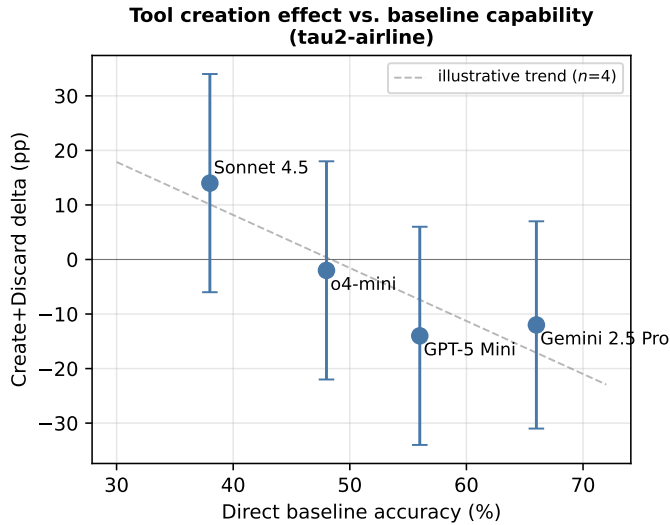


Figure 3: **Baseline capability vs. Create+Discard effect on τ^2 -airline (4 models).** Lower Direct accuracy coincides with larger gains from Create+Discard, consistent with the reasoning-scaffold interpretation. With only four models, confirmation across more models is needed.

C.7 Conversation examples

The behavioral difference between Direct and Create+Discard on τ^2 -airline is dramatic at the individual task level. Two representative examples:

Task 3 (baggage allowance inquiry).

Direct (FAIL): 5 agent messages, 0 tool calls. Agent asks for user ID, customer provides it, agent fails to respond. Scorer: “Agent did not communicate the required info.”

Create+Discard (PASS): 15 agent messages, 3 tool calls. Agent calls `get_reservation_details` and `get_user_details`, discovers Silver membership, communicates correct baggage count (4 suitcases), then transfers to human agent when the customer disputes membership level. Scorer: “Agent communicated all required info.”

Task 39 (cancel all upcoming flights).

Direct (FAIL): 5 agent messages, 0 tool calls. Conversation terminates after the customer provides user ID.

Create+Discard (PASS): 35 agent messages, 12 tool calls. Agent retrieves all 7 reservations, processes cancellations per policy, and confirms each with the customer.

Of the 35 zero-tool Direct tasks, 14 still pass because their target behavior is restraint (e.g., “agent should refuse to proceed with the cancellation”)—an agent that does nothing accidentally satisfies this criterion. The remaining 21 fail because they require database-modifying actions that the agent never attempts.

D Harness design details

This appendix expands on the harness design choices summarized in §3.

D.1 Existing harness paradigms

Prior work uses three broad paradigms:

Prompt-chained pipelines. CREATOR (Qian et al., 2023) and LATM (Cai et al., 2024) use fixed multi-stage pipelines: a creation prompt produces a tool, a separate invocation prompt applies it. This maximizes reproducibility but limits the agent’s ability to adaptively decide *when* to create or retrieve.

ReAct-style action loops. Voyager (Wang et al., 2024), ASI (Wang et al., 2025b), and most recent systems use a ReAct loop where tool creation is one action among many. More flexible but introduces the confound that the agent decides when to create tools.

RL-trained policies. SkillRL (Xia et al., 2026), SAGE (Wang et al., 2025a), and ARISE (Li et al., 2026b) train the tool-management policy via RL. This conflates the library’s value with the RL optimization, making it hard to isolate tool contributions.

D.2 Condition instruction policy

All solvers prepend the same base system prompt:

```
You are running inside the Winnow experiments harness.  
Follow the condition-specific instructions exactly and return a  
benchmark-ready solution. When Inspect tools are available, call them  
explicitly rather than only describing a helper in prose.
```

Each solver then prepends a condition block beginning with Condition: <label> and Description: <description>, followed by one exact condition-specific instruction sentence. The reported runs use the following policy text:

- **Direct:** “Solve directly without inventing reusable memory artifacts.”
- **CoT-Extended:** “Use extra deliberate reasoning budget, but do not rely on tools or memory.”

- **Structured-CoT**: “Organize the response as design, implement, and verify before answering.”
- **Create+Discard_{full}**: “If a helper would help, call `create_tool` to register it and inspect the verification feedback, but do not rely on persistent memory.”
- **Create+Invoke_{once}**: “If a helper would help, call `create_tool` and then `invoke_tool` during this task only. Nothing persists after the sample.”
- **Create+Reuse**: “Retrieve relevant stored helpers when they exist. If retrieval returns a plausible prior-task helper, invoke it before creating a new helper. Only `create_tool` when the retrieved implementations are clearly insufficient for this task.”
- **Trajectory Retrieval**: “Lean on analogous past trajectories instead of abstract helper functions.”
- **Create+Admit**: “Retrieve relevant stored helpers when they exist. New tools may be used on this task, but only correct, non-redundant helpers stay in shared memory. Invoke a retrieved prior-task helper before creating a new one when the retrieved match is plausible.”

All answer modes also append final-answer requirements. For reusable-tool conditions, these include “Query `retrieve_tool` before creating a near-duplicate helper” and “If `retrieve_tool` returns hits, invoke the strongest prior-task candidate before calling `create_tool` unless it is clearly irrelevant.” Tool-creating conditions are also prompted to prefer concise helper names and summaries so future retrieval can find them. For τ^2 -airline, the wrapper appends one additional sentence ahead of the domain policy: “You also have access to `create_tool`, `retrieve_tool`, and `invoke_tool` for building reusable helper functions that wrap common multi-step operations.”

The exposed tool interfaces are:

```
create_tool(name: str, summary: str, body: str)
retrieve_tool(query: str, k: int = 3)
invoke_tool(tool_id: str)
```

The Create+Admit prompt uses the phrase “correct, non-redundant” helpers. In the reported syntax-mode runs, this operationally means passing creation-time syntax validation plus the admission filter’s non-redundancy rule, not a full functional proof of correctness.

D.3 Execution and validation environments

Task-level code execution uses Docker containers via Inspect’s built-in sandboxing. On BigCodeBench-Hard, final submitted answers are scored by the upstream `verify()` harness inside Docker. On τ^2 -airline, the reported runs keep the upstream task, user simulator, and judge-based scoring pipeline.

Helper creation-time validation is lighter-weight. In the reported runs, `create_tool` performs a syntax/function-definition check and records whether the helper passed that validation. The runtime also supports an optional held-out execution path for admission-controlled conditions, but that path is not used in the reported BCB-Hard or τ^2 rows.

D.4 Retrieval configuration sensitivity

The default retriever computes cosine similarity over bag-of-words term-frequency vectors: each text is lowercased, tokenized into alphanumeric tokens via regex `[a-z0-9_]+`, and represented as a raw token-count vector (no IDF or subword normalization). The indexed text for each tool is the concatenation of its name, summary, and source body; the same representation is reused by the admission filter’s distance computation. This is a deliberately simple baseline; the embedding variant described below tests whether a stronger scorer changes the outcome.

We also test an **embedding-based retrieval** variant that replaces the lexical scorer with `text-embedding-3-small` (OpenAI, 1536-dimensional embeddings). Embeddings are cached

per text so each tool description is embedded once. All other harness components—tool creation, invocation, admission control, scoring—remain identical to the lexical runs. The embedding variant scores 40.5% on BCB-Hard and 58.0% on τ^2 -airline, both within noise of the lexical baseline (Table 7).

The benchmark-specific probes in §6.1 keep the lexical similarity function but change the query source and/or lookup text. On BCB-Hard, the upper bound uses benchmark target text unavailable to the agent. On τ^2 -airline, the diagnostic probe uses task metadata as the query and upweights source text in the lookup representation.

D.5 Harness as a confound

The harness design is itself a potential confound. Our admission policy uses lexical similarity and syntax validation as proxies for semantic novelty and correctness; both are imperfect. We nevertheless view the setup as informative because all BCB-Hard conditions share the same runtime, and because the goal here is a conservative, easy-to-audit baseline rather than an optimized library-management policy.

One benchmark-specific asymmetry affects interpretation. On τ^2 -airline, Direct / CoT-Extended / Structured-CoT delegate to the upstream τ^2 agent, whereas tool-creating conditions preserve the same dataset, user simulator, and scorer but add the harness’s meta-tools through a local wrapper. That wrapper also normalizes empty messages and enforces at most one assistant tool call per turn to stay within τ^2 ’s interaction protocol. Combined with the fixed task order, this means the τ^2 results should be interpreted as evidence from a closely matched harness, and significance-style analyses there should be read as exploratory rather than definitive.

We have probed retrieval along two axes—a stronger scorer (embedding-based, text-embedding-3-small) and stronger queries (benchmark-aware probes)—neither of which changes the persistence conclusion. Learned re-ranking and structured skill descriptions remain untested.

E Disclosure of LLM use

In accordance with COLM’s policy on disclosure of LLM use, we report that LLM-based code-generation agents were used during development of the experiment harness and analysis scripts, with the code being reviewed by at least one of the authors. LLMs were also used to improve grammar and readability of the manuscript, as the authors are not native English speakers.