COMPILE THE CATALOG, NOT THE TASK: TOOLGRAPH AS A TOOL-USE AGENT HARNESS THAT UPDATES, SCALES, AND TRANSFERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-use agents must chain interdependent calls, and which outputs satisfy which later inputs is a property of the deployed tool *catalog*, not of any one task or model. We argue that this structure should be compiled once into a persistent artifact that any frozen agent can read. TOOLGRAPH is a directed, weighted tool-dependency graph built from tool interfaces alone: parameter matching and counterfactual probes, in which an evaluator LLM compares a candidate chain with the same query answered without its proposed predecessor, propose edges, and a Bayesian combination of match and information-gain evidence sets each edge's confidence. Construction reads no task data, reward, or expert label. A frozen planner reads the graph through a top-k injection ranked by personalized PageRank or by a goal-conditioned action value $Q(s, a, g)$ composed from graph paths. We study TOOLGRAPH as an agent harness along three axes. **Update (continual learning):** the agent stays frozen while the graph learns. Create/read/update/delete (CRUD) operations driven by execution outcomes raise completion on held-out τ^2-Bench tasks by 6–7 points for two planners, with the gain saturating by 60–80% of the update stream. **Scale (test-time scaling):** spending test-time compute on an LLM scorer that reads graph evidence outperforms Language Agent Tree Search in all six ToolComp and τ^2-Bench settings while using 6–29$\times$ fewer LLM calls per task. **Transfer:** with no training, subgraphs retrieved from a 618-node TOOLGRAPH compiled on Toolathlon's catalog lift DeepSeek-V3.2 on τ^2-Bench from 80 to 88, while the reverse direction trails the target's own graph. Across three benchmarks and three backbones, the reader matters more than the graph: the same edges help or hurt depending on how the planner reads them.

1 INTRODUCTION

Language-model agents extend generation into interaction by calling tools and reading their results (Yao et al., 2023b; Schick et al., 2023). What makes tool use hard is rarely a single call. It is *chaining*: knowing which tool's output supplies a later tool's input, in what order, and under which preconditions. That dependency structure belongs to the deployed catalog, not to any one task, and it is poorly recovered from documentation. On expert-labelled parameter-level API dependencies, zero-shot closed-source LLMs reach only 42.3–70.0% three-way accuracy (Lee et al., 2026). The problem grows with catalogs of hundreds of MCP tools (Li et al., 2026b; Wang et al., 2026) that also change over time (Huang et al., 2024; Chen et al., 2025).

Current methods supply this structure in one of two ways, and each pays per task or per catalog change. The first writes it into weights: fine-tuning on curated tool data generalizes to unseen APIs and tasks (Qin et al., 2024; Zeng et al., 2024; Xiao et al., 2025), a builder fine-tuned on expert API graphs carries across datasets (Lee et al., 2026), and tool- or graph-token methods internalize the catalog (Wang et al., 2025a; Gao et al., 2026). All need training data and weight access, and a changed catalog means retraining. The second rediscovers structure at test time, by searching over reasoning steps and tool calls (Yao et al., 2023a; Zhou et al., 2024; Zhuang et al., 2024) or by mining transition and dependency graphs from invocation histories (Liu et al., 2024a; Jia & Li, 2026; Li et al., 2026c). Search pays on every request, and a mined graph cannot exist before the trajectories that produce it.

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

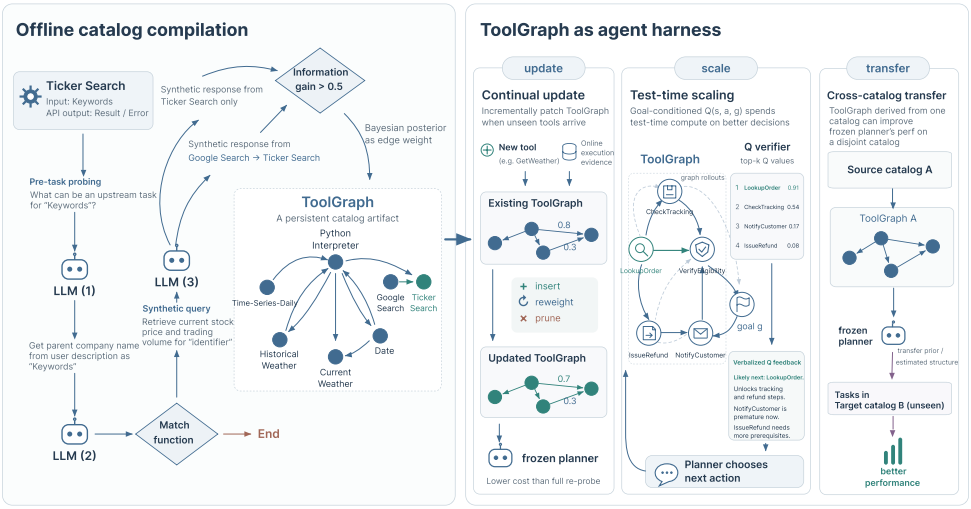

Figure 1: **ToolGraph as a harness around a frozen planner.** *Left:* probes synthesized from tool interfaces compare a candidate chain with and without its proposed predecessor, and edges are weighted by a Bayesian posterior over the estimated information gain. *Right:* the artifact is **updated** from execution outcomes, **scaled** through a goal-conditioned $Q(s, a, g)$ verifier, and **transferred** to a planner on a disjoint catalog.

We take a third position: *compile the catalog, not the task.* TOOLGRAPH is a persistent, weighted tool-dependency graph compiled once from tool interfaces alone. Parameter matching and counterfactual probes, which compare a candidate chain with the same query answered without its proposed predecessor, supply evidence, and a Bayesian combination of that evidence becomes each edge's confidence. No task text, trajectory, reward, or expert label is read. We treat the result as a *harness* around a frozen planner (Figure 1). The planner reads the graph through a top-k injection ranked by personalized PageRank or by a goal-conditioned action value $Q(s, a, g)$ composed from graph paths, and it is the graph, not the planner, that learns. Three questions organize the paper. **Update:** can a frozen agent keep improving at tool use as the graph learns from the outcomes of its own use? **Scale:** can test-time compute spent on reading the graph buy more success than search does? **Transfer:** does a graph compiled on one catalog help on a different one?

Contributions.

- **A task-free harness for continual learning.** We compile TOOLGRAPH from interfaces and counterfactual probes and maintain it with outcome-driven CRUD, so a frozen agent's tool use keeps improving while only the graph changes.
- **One graph, four readers.** Over one frozen graph per catalog, an LLM scorer reading graph evidence is the best of four readers in all nine benchmark–backbone columns (mean 52.0, against 45.6 for PageRank ranking and 45.7 for the strongest baseline), while two purely structural value estimators fall below relevance ranking.
- **Update, scale, transfer.** CRUD raises held-out completion of a frozen agent by 6–7 points, saturating before the stream ends. For test-time scaling, graph-guided scoring beats LATS in all six ToolComp and τ^2 settings at 6–29$\times$ fewer LLM calls. A Toolathlon graph improves DeepSeek-V3.2 on τ^2-Bench by 8 points without training; the reverse direction trails the target's own graph.

2 RELATED WORK

Tool graphs. ToolNet navigates a tool-transition graph (Liu et al., 2024a), ControlLLM searches a pre-built tool graph (Liu et al., 2024b), TaskBench samples tool subgraphs to synthesize tasks

(Shen et al., 2024), and graph networks outperform LLMs at selecting connected tool subgraphs (Wu et al., 2024). Recent systems differ mainly in where edges come from. GTool (Chen et al., 2026), AutoTool (Jia & Li, 2026), and H-EPM (Li et al., 2026c) mine them from invocation trajectories, NaviAgent refines its tool model from interaction feedback (Jiang et al., 2026), Tool Graph Retriever trains a discriminator on labelled pairs (Gao et al., 2025), DTDR fits demonstrations (Patel et al., 2026), and In-N-Out and ToolDial derive graphs from documentation, the former with expert labels (Lee et al., 2026; Shim et al., 2025). ToolExpNet tests relations by simulated trial and error (Zhang et al., 2025b). ToolAtlas builds a persistent tool memory by execution-verified probing that transfers across instances of one service and across agent frameworks (Fang et al., 2026); we instead attach a graded counterfactual confidence to each dependency edge, update it from outcomes, and test transfer to a catalog with no shared tools. STE and DRAFT refine per-tool knowledge through self-driven use (Wang et al., 2024a; Qu et al., 2025); TOOLGRAPH instead estimates which tool's output another needs, as REST API testing did from interface specifications (Atlidakis et al., 2019). Our schema-derived edges and PageRank reading follow Liu et al. (2025a). Tool- and graph-token methods internalize the catalog into weights (Wang et al., 2025a; Gao et al., 2026); we keep the planner frozen, and structure-free retrievers (Qin et al., 2024; Chen et al., 2024) correspond to our flat control.

Continual learning and experience memory. Catalog change is a continual-learning problem (Huang et al., 2024), usually addressed by training (Chen et al., 2025). Training-free agents accumulate external memory instead: insights (Zhao et al., 2024), trial-to-trial memory (Majumder et al., 2023), workflows (Wang et al., 2025c), reasoning strategies (Ouyang et al., 2026), context playbooks (Zhang et al., 2026), cheatsheets (Suzgun et al., 2026), skill libraries (Wang et al., 2024b), and graph-structured experience (Feng et al., 2026; Li et al., 2026a). HippoRAG ranks a persistent knowledge graph by personalized PageRank and treats growing it as non-parametric continual learning (Jiménez Gutiérrez et al., 2024; Gutiérrez et al., 2025). ScaleMCP syncs a flat tool index through CRUD (Lumer et al., 2025). Our unit of memory is a typed dependency edge with a Beta posterior, updated from execution outcomes and read by any frozen planner, so continual learning for tool use needs no change to the agent.

Test-time compute. Agents spend test-time compute on deliberate search (Yao et al., 2023a; Hao et al., 2023), Monte Carlo tree search with LM values (Zhou et al., 2024; Yang et al., 2026), A* over tool plans (Zhuang et al., 2024), or best-first search in a live environment (Koh et al., 2025), all of which need resettable or simulated state that irreversible tool actions complicate (Gu et al., 2025). Compute can also go into verifiers (Cobbe et al., 2021; Lightman et al., 2024; Zhang et al., 2025a; Xia et al., 2025) and its allocation (Snell et al., 2025; Zhu et al., 2025). Goal-conditioned values (Schaul et al., 2015) verbalized to a frozen LLM (Hong et al., 2025) and self-taught lookahead (Mendes & Ritter, 2025) are our nearest neighbours; each fits its value on task trajectories or reward, whereas ours is composed from a graph that saw neither, via a probabilistic relaxed-reachability heuristic (Bonet & Geffner, 2001; Hoffmann & Nebel, 2001).

Generalization across tools and domains. The dominant route is fine-tuning on broad or curated tool data (Qin et al., 2024; Patil et al., 2024; Liu et al., 2024c; 2025b; Xu et al., 2025); AgentTuning and LIMI show that a small curated set of trajectories yields agentic ability on unseen tasks (Zeng et al., 2024; Xiao et al., 2025), though such transfer can memorize rather than generalize (Chu et al., 2025). In-N-Out is the closest precedent for our transfer axis: a graph builder fine-tuned on one benchmark's expert API graphs predicts useful graphs for another, which help a frozen agent (Lee et al., 2026). Training-free transfer through memory is emerging, in cross-domain experience bases (Tang et al., 2025), shared coding memories where abstract insights transfer best and low-level traces often transfer negatively (Kim et al., 2026), and skills reused across websites (Wang et al., 2025b). ToolLIFT argues that trajectory-built tool-level graphs transfer poorly across tool sets and lifts them to function-level workflows for an RL-trained planner (You et al., 2026). We test whether a dependency graph distilled from one catalog, with no model trained, helps a frozen agent on a catalog that shares none of its tools.

3 COMPILING THE CATALOG INTO TOOLGRAPH

Let the catalog $\mathcal{T} = \mathcal{T}_a \cup \mathcal{T}_m$ contain *atomic* tools with declared, typed interfaces and *meta* tools, such as search engines or interpreters, whose output semantics must be established empirically. TOOLGRAPH is a directed weighted graph $G = (\mathcal{T}, E, W)$. Each node stores a record $\Phi(t) = \langle \textit{name}, \textit{class}, \textit{domain}, \textit{intent}, \textit{required entities}, \textit{output description} \rangle$ and an embedding of its intent and output text. An edge $t_i \to t_j$ states that running t_i may supply information t_j needs, and $w_{ij} \in [0, 1]$ is the confidence that this dependency holds.

Edge proposal. Two proposers run over tool pairs, and their evidence is merged per pair. *Parameter matching* asks an LLM, for each pair of atomic tools in a domain, whether one tool's output fields can supply the other's required inputs, with a confidence in $[0, 1]$. It is cheap but reaches only tools with declared, typed interfaces. *Counterfactual probing* covers the rest with three LLM roles, using a different model for evaluation to avoid self-confirmation. For each atomic tool t_i, $\mathcal{L}_{\text{gen}}$ derives tool-agnostic tasks from its interface alone: tasks that would produce each required entity (*pre-tasks*), tasks that would consume each output field (*post-tasks*), and the tool's own task (*same task*). $\mathcal{L}_{\text{match}}$ scores in $[0, 1]$ whether a meta tool t_j could perform each task, and tasks scoring at least θ_{match} become probe queries Q. For each probe, the evaluator $\mathcal{L}_{\text{eval}}$ receives Q, both tool schemas, and a proposed order. It simulates the output of the chain and of the second tool alone, checking for inputs the user could simply supply, for one tool subsuming the other, and for capabilities a tool lacks, and scores how much of Q each resolves, $r(\cdot, Q) \in [0, 1]$. The information gain of an evidence component c (one probed entity or output field) is

$$\widehat{\mathrm{IG}}_c = r(\hat{y}_{\text{chain}}, Q_c) - r(\hat{y}_{\text{base}}, Q_c), \tag{1}$$

and only components with $\widehat{\mathrm{IG}}_c > 0$ are kept. Pairs within one class (atomic–atomic, meta–meta) are probed with a single call that returns both a match score and a gain. The contrast with a baseline that lacks the predecessor is what makes a probe counterfactual: a predecessor's contribution is estimated against its absence rather than inferred from names or types.

Edge confidence. Each retained component c contributes a posterior. Its match score is the prior, $p(e_{ij} \mid c) = s_c$, and the information gain enters through a shifted-sigmoid likelihood $\ell_c = \sigma\big((\widehat{\mathrm{IG}}_c - 0.5)/\tau\big)$:

$$\pi_c = \frac{\ell_c\, s_c}{\ell_c\, s_c + (1 - \ell_c)(1 - s_c)}, \qquad w_{ij} = \frac{1}{|\mathcal{C}_{ij}|} \sum_{c \in \mathcal{C}_{ij}} \pi_c, \tag{2}$$

A parameter-matching component contributes its confidence as its posterior. The edge $t_i \to t_j$ is kept when $w_{ij} > \theta_{\text{prune}}$, and each edge stores its probe queries, simulated outputs, and confidence components, so any value computed from the graph traces back to its evidence.

No task data. Probe queries are synthesized from interfaces alone, and probe outcomes are simulated by the evaluator rather than obtained by calling the tools. No benchmark task, reference trajectory, reward, or environment feedback from any split is read during construction, so edge weights cannot encode the tasks the graph is later evaluated on. This matters because one graph per catalog is reused by every task, backbone, and arm. Figure 2 shows the three compiled graphs; Appendix Tables 7 and 8 give their statistics and construction cost.

4 READING TOOLGRAPH AT INFERENCE

Every TOOLGRAPH arm shares one frozen graph per catalog, one top-k injection interface, and one prompt template per benchmark (Appendix Table 9). The planner proposes n candidate actions per decision; each reader scores (candidate, goal) rows, and the top k rows, with their evidence, are injected. The arms differ in how the rows are ranked and what is said about them. Every arm also exposes the full catalog to the planner, so the injected block is *advice within* the prompt, not a restriction of the action space.

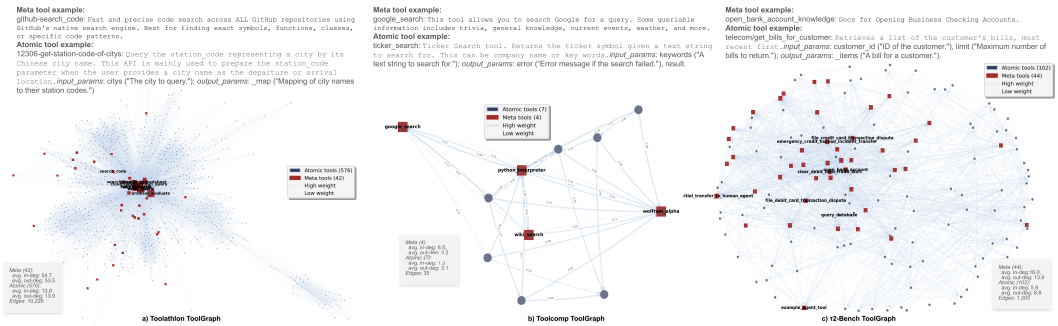

Figure 2: **Compiled ToolGraphs** for Toolathlon (576 atomic and 42 meta tools, 10,226 edges), ToolComp (7 atomic and 4 meta tools, 35 edges), and τ^2-Bench (102 atomic and 44 meta tools, 1,303 edges), with one meta and one atomic tool shown per catalog. Edge shade encodes confidence w_{ij}.

Relevance ranking. TG-PPR (row "TG top-k PPR" in the tables) ranks tools by query-conditioned personalized PageRank (Page et al., 1999; Haveliwala, 2002), as HippoRAG does over knowledge graphs (Jiménez Gutiérrez et al., 2024). It computes $\boldsymbol{\pi}_q = \alpha \mathbf{v}_q + (1-\alpha)\mathbf{W}^\top \boldsymbol{\pi}_q$ over the row-normalized weighted adjacency $\mathbf{W}$, with personalization $v_q[t] = \lambda_{\text{int}} \operatorname{sim}(q, \textit{intent}_t) + \lambda_{\text{out}} \operatorname{sim}(q, \textit{output}_t) + \lambda_{\text{ent}} \operatorname{sim}(q, \textit{entities}_t)$.

A goal-conditioned action value. An *entity* is a named datum passed between tools, such as `order_id`. Let $E(s)$ be the entities available in state s, and let I_a and O_a be action a's required and produced entities. Action a is executable when $I_a \subseteq e$ for $e = E(s)$; success yields $e^a = e \cup O_a$, and failure leaves e unchanged. Because effects only add entities, this is a probabilistic *delete-free* relaxation, the analogue of the relaxed-reachability heuristics of classical planning (Bonet & Geffner, 2001; Hoffmann & Nebel, 2001). A goal g is a *goal tool*, inferred from the request, whose executability would satisfy it. States in which some goal is executable take value $V_g = +1$, and states with no executable action take $V_g = -1$. Everything else is backed up:

$$\begin{aligned} \bar{Q}_g(e,a) &= \gamma\big[p_a V_g(e^a) + (1-p_a)V_g(e)\big], \qquad V_g(e) = \max_{a:\, I_a \subseteq e} q_g(e,a), \\ q_g(e,a) &= \bar{Q}_g(e,a) - \kappa(1-c_a)\,\big|\bar{Q}_g(e,a)\big|, \end{aligned} \tag{3}$$

where p_a and c_a are the maximum and the mean of a's incoming edge weights; the penalty damps values supported by uncertain edges. We study three estimators, each nested in the next. Q_{dp} solves Eq. 3 by value iteration over the reachable lattice and maps it affinely to $[0,1]$. Q_{emb} is an inductive GraphSAGE model (Hamilton et al., 2017), $\sigma(\text{MLP}[z_a \| z_s \| z_g])$, distilled from Q_{dp} on training-split states. Q_{llm} is an LLM scorer that reads the concrete state, candidate arguments, goal, policy, the verbalized subgraph and edge evidence, with Q_{dp} as a prior, and returns a 0–10 score (five samples, averaged); when several candidates are eligible, it ranks them in a single comparative call instead. A message to the user scores $\gamma V_g(E(s))$, the value of leaving the entity state unchanged. A *verbalizer* turns the top-k action–goal rows and their evidence into feedback and a recommended action. It costs one call per decision, and the planner may ignore it.

Evidence budget. Four knobs control the graph evidence Q_{llm} sees: subgraph radius h, edge threshold θ, per-node annotation depth (N0 edges only; N1 + input/output fields; N2 + structural features; N3 + call history), and the number n of candidates scored per decision, the test-time compute knob.

Cross-catalog reading. To transfer, we change only the graph Q_{llm} reads. For a task in target catalog B, we retrieve the subgraphs of the source graph G_A whose nodes are most semantically related to the task, select the most task-related tools within them, and present those source subgraphs to Q_{llm} *in place of* the target's own subgraph. The planner still sees and calls only the tools of B. Because A and B share no tools, what transfers is dependency structure evidenced on analogous tools, not edges the planner can execute.

5 UPDATING TOOLGRAPH: CRUD FROM EXECUTION OUTCOMES

After deployment the graph is maintained by four operations. READ is the inference of Section 4. UPDATE refines edges from each trajectory's outcome, as labelled by the benchmark's own evaluator (on τ^2, a pass is full reward). On a pass, every consecutive tool pair receives $\alpha_{ij} \mathrel{+}= 1$; on a failure, every traversed edge receives $\beta_{ij} \mathrel{+}= 0.3$; trajectories with an uncertain outcome are skipped. The edge weight is the Beta mean $w_{ij} = \alpha_{ij}/(\alpha_{ij} + \beta_{ij})$. CREATE adds a previously unseen edge once a tool pair has co-occurred in n_{create}=2 passing trajectories, starting at w=2/3. DELETE prunes an edge once $n_{ij} \geq 5$ observations leave $w_{ij} < 0.05$. Adding or removing a *tool* recompiles only that node and its incident edges (Appendix Table 8).

Every operation touches only the graph, never the planner. Continual learning for tool use therefore becomes maintenance of one shared artifact: evidence from the agent's own executions refines edge confidences, new dependencies enter through CREATE, unreliable ones leave through DELETE, and every frozen planner that reads the graph inherits the change.

6 EXPERIMENTAL SETUP

Benchmarks. *ToolComp* (Nath et al., 2025) tests multi-step, multi-tool reasoning over 11 tools, with final answers graded by an LLM judge. We use a random 60/40 split into 296 training and 197 evaluation tasks. With only 11 tools, the injected ranking covers most of the catalog, so ToolComp mainly tests how the graph is read rather than what it retrieves. *τ^2-Bench* (Barres et al., 2026) covers customer-service tasks in the airline, retail, and telecom domains (telecom is dual-control: the user also acts). We use the official train/test split released with the benchmark code, with 178 training and 100 test tasks, weighted 20/40/40 across domains, and the benchmark-native reward: a product of per-task checks on the final database or environment state and, in airline and retail, on required communication or natural-language assertions. *Toolathlon* (Li et al., 2026b) consists of long-horizon tasks over 32 real applications behind MCP servers, scored by execution-based checking scripts. The benchmark exposes 604 MCP tools and 16 local tools; our compiled graph has 618 nodes (576 atomic, 42 meta). We use a 60/40 split into 65 training and 43 evaluation tasks. Training splits are used only by the CRUD stream, for hyperparameter selection, and for the states that distill Q_{emb}; construction never reads them.

Backbones and baselines. The planners are Ministral-3-8B-Instruct-2512 (M) (Liu et al., 2026), Gemma 4 31B (G) (Gemma Team, 2026), and DeepSeek-V3.2 (D) (DeepSeek-AI, 2025). The *verifier* model is held fixed across planners. It writes the Q_{llm} scores, the verbalizer's feedback, and Reflexion's self-reflections; it is Gemma 4 31B unless stated, with Claude Opus 4.6 (Anthropic, 2026) as an ablation. The τ^2 user simulator is Claude Sonnet 4.5. Construction uses Claude Sonnet 4.5 for edge proposal and probe synthesis and Claude Sonnet 4 for probe evaluation, independent of every test-time model. Baselines are *Native* function calling, *ReAct* (Yao et al., 2023b), and single-revision *Reflexion* (Shinn et al., 2023), whose reflection the verifier writes with the verbalizer's budget. *Flat top-k retrieval* ranks tools by request–description embedding similarity on the same code path with graph propagation removed, separating the injection interface from the structure. *LATS* (Zhou et al., 2024) is the search baseline for test-time scaling. Because τ^2 and ToolComp allow one real attempt per task over stateful environments, search below the root is imagined rather than executed. At each decision the planner samples 4 candidate actions (T=0.7), which form the real root frontier. Each of 5 simulations UCT-selects a root child (c=1), rolls out an imagined continuation of up to 5 steps (the planner imagines actions and the verifier imagines tool results), and scores the leaf with the verifier on a 0–10 scale; rollouts valued below 0.5 yield a verbal reflection that later simulations see. The most-visited root action is executed, and the same configuration is used on both benchmarks.

7 RESULTS

7.1 HOW TOOLGRAPH IS READ MATTERS MORE THAN WHAT IT CONTAINS

Table 1 changes one component at a time. *Native to flat retrieval* adds the injection interface alone and lowers the Mean (43.4 → 42.7) and five of six ToolComp and τ^2 cells: with the full catalog visible, an unstructured block only competes for attention. *Flat to TG-PPR* adds dependency structure

Table 1: **Task completion (%) with Gemma 4 31B as verifier.** Flat retrieval and all TOOLGRAPH (TG) arms share one top-k injection interface, one candidate pool, and one prompt template; the TG arms read the same frozen graph per catalog and differ only in how the injected rows are ranked and described. Every arm exposes the full catalog. M/G/D: Ministral 3 8B, Gemma 4 31B, DeepSeek-V3.2. The mean is over all nine cells; bold marks the best arm per column.

	ToolComp			τ^2-Bench			Toolathlon			
Method	M	G	D	M	G	D	M	G	D	Mean
No-graph baselines										
Native	57	73	73	40	67	72	0	3	6	43.4
ReAct	47	65	62.4	38	52	63	0	9	9	38.4
Reflexion	51	70	73	49	69	75	0	6	18	45.7
Flat top-k retrieval	54	72	72	34	62	72	0	6	12	42.7
ToolGraph readers (one shared frozen graph per catalog)										
TG top-k PPR	60	75	73	37	67	74	3	12	9	45.6
TG top-k Q_{dp}	56	72	74	38	54	67	3	9	9	42.4
TG top-k Q_{emb}	58	70	68	36	50	66	3	9	6	40.7
TG top-k Q_{llm}	**62**	**78**	**77**	**53**	**73**	**80**	**6**	**18**	**21**	**52.0**

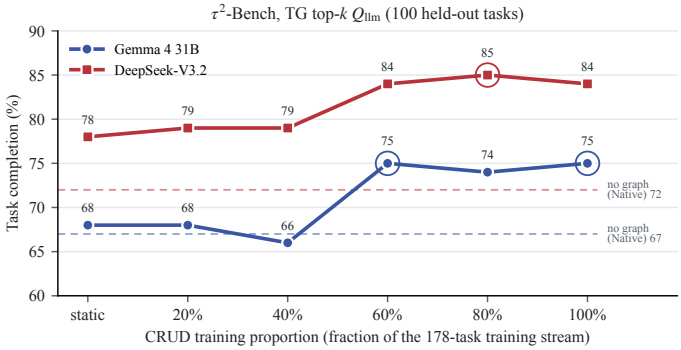

Figure 3: **Continual learning through CRUD** on τ^2-Bench, TG top-k Q_{llm} with a frozen planner (Gemma 4 31B, DeepSeek-V3.2). The graph is updated from execution outcomes on growing fractions of the 178-task training stream and evaluated on the 100 held-out test tasks after each checkpoint. Rings mark each curve's maxima; dashed lines show the no-graph Native planner.

to the ranking and, with it, the verbalizer's feedback, since flat retrieval makes no verifier call. This step therefore bounds rather than isolates the value of structure. It helps in eight of nine cells (Mean $42.7 \rightarrow 45.6$), consistent with the structure carrying signal a frozen planner can use. *TG-PPR to the value estimators* keeps the graph and changes how its edges are combined, and the answer splits. Q_{llm} is the best arm in all nine columns and beats TG-PPR in each of them, by $+2/+3/+4$ on ToolComp, $+16/+6/+6$ on τ^2, and $+3/+6/+12$ on Toolathlon (Mean 52.0, against 45.7 for Reflexion, the strongest baseline). The two structural estimators fall *below* relevance ranking (Mean 42.4 and 40.7). All four TG arms read the same frozen graph, so their differences come from how it is read. Reflexion, whose feedback is written by the same verifier model without graph evidence, trails Q_{llm} by 6.3 points on the Mean. With a stronger verifier (Claude Opus 4.6), Q_{llm} remains the best arm on every backbone (Appendix B).

The structural estimators lose most on τ^2, where Q_{dp} trails TG-PPR by 13 (G) and 7 (D) points against -4 to $+1$ on ToolComp. τ^2 is conversational, and Eq. 3 scores a message $\gamma V_g(E(s))$, strictly below the best admissible tool call whenever the goal is reachable. The relaxation therefore never recommends communicating in a state it values positively, a constraint that neither Q_{llm}, which reads the conversation, nor relevance ranking shares.

Table 2: **Test-time scaling: task completion versus test-time cost** (Scale). LATS (Zhou et al., 2024) spends compute on Monte Carlo tree search; TG top-k Q_{llm} spends it on scoring candidate actions against graph evidence. LLM calls and tokens (K = thousands) are per task. The last block gives the completion gap (Q_{llm}−LATS, points; ToolComp Q_{llm} cells are recorded at integer precision) and the ratio of LATS cost to Q_{llm} cost.

		ToolComp			τ^2-Bench		
Method		Compl.	Calls/task	Tok./task	Compl.	Calls/task	Tok./task
LATS	M	59.4	155.4	307.9K	47	655.7	2,487.2K
	G	75.1	181.2	264.3K	70	463.3	2,042.9K
	D	75.6	170.4	369.0K	75	379.5	1,510.4K
TG Q_{llm}	M	**62**	25.5	117.7K	**53**	23.4	143.0K
	G	**78**	6.2	28.5K	**73**	22.3	108.2K
	D	**77**	15.8	72.8K	**80**	26.0	159.6K
Gap / ratio	M	+3	6.1×	2.6×	+6	28.0×	17.4×
	G	+3	29.2×	9.3×	+3	20.8×	18.9×
	D	+1	10.8×	5.1×	+5	14.6×	9.5×

7.2 UPDATE: CONTINUAL LEARNING FOR TOOL USE WITH A FROZEN AGENT

Updating the graph from its own execution outcomes improves completion on tasks it never saw (Figure 3). From the compiled graph to the end of the stream, Gemma gains 7 points (68 → 75) and DeepSeek 6 (78 → 84), finishing 8 and 12 points above the no-graph Native planner. The gain is not immediate. The first 40% of the stream moves completion by at most 2 points (Gemma 68 → 66, DeepSeek 78 → 79), and most of the gain arrives between 40% and 60%. The benefit then *peaks before the stream is exhausted*: Gemma first reaches its maximum at 60% and only matches it at 100%, and DeepSeek peaks at 80%. Throughout, the planner's weights never change. What improves is the graph it reads, so this is continual learning for tool use carried entirely by the harness, and any planner that reads the updated graph inherits it.

7.3 SCALE: TEST-TIME SCALING WITH GRAPH-GUIDED SCORING

Test-time scaling asks how well a method converts test-time compute into success. LATS spends it on search (expanding, LM-valuing, and reflecting); TOOLGRAPH spends it on scoring a few candidate actions against compiled evidence (Table 2). Q_{llm} completes more tasks in all six ToolComp and τ^2 settings, by 1–3 points on ToolComp and 3–6 on τ^2, while using 6–29× fewer LLM calls and 2.6–19× fewer tokens per task. Even with the amortized construction cost added (88K tokens per τ^2 task; Table 8), Q_{llm} uses 6–11× fewer tokens than LATS on τ^2. The token ratio is largest on τ^2, where LATS spends 1.5–2.5M tokens per task against 108–160K for Q_{llm}: search rediscovers each task's dependencies, while the graph supplies them once per catalog. We did not sweep LATS's budget, so this compares operating points, not full compute–success curves.

Within Q_{llm}, the number of scored candidates n is the compute knob (Appendix Table 5). Going from n=1 to n=4 raises macro-average success from 61.1 to 66.4 on τ^2 (67.7 → 155.2K tokens) and from 66.6 to 69.0 on ToolComp (43.9 → 79.0K). Going on to n=8 adds 0.4 on ToolComp and loses 1.7 on τ^2 at 1.6–1.8× the tokens, so most of the benefit arrives with the first few scored candidates. The other knobs move cost by at most 12% and success by about run-to-run noise (on ToolComp, h=1, 2, 3 retrieve the identical graph yet score 67.3, 69.5, 66.1).

7.4 TRANSFER: A TOOLGRAPH COMPILED ON A DIVERSE CATALOG HELPS A DIFFERENT ONE

Table 3 tests transfer in both directions, and the effect is strongly asymmetric. Reading subgraphs from the Toolathlon graph, compiled over 32 applications, DeepSeek-V3.2 completes 88 of 100 τ^2 tasks: 8 more than with τ^2's own graph and 16 more than without a graph. Gemma matches its in-catalog score. With Toolathlon as the target, subgraphs from the 146-tool τ^2 graph do not improve on Toolathlon's own graph (Gemma 18 → 15; DeepSeek 21 → 9 in a run that hit more timeouts), but they still beat no graph for both planners (3 → 15 and 6 → 9). Every cross-catalog cell stays above

Table 3: **Cross-catalog transfer** (Q_{llm} reader). CROSS-CATALOG reads the *other* catalog's subgraphs in place of the target's own; the planner acts only on target tools, and the catalogs share no tools. Bold: best per column.

	Target: τ^2-Bench			Target: Toolathlon		
Graph read	source	G	D	source	G	D
no graph (Native planner)	—	67	72	—	3	6
IN-CATALOG	τ^2	**73**	80	Toolathlon	**18**	**21**
CROSS-CATALOG	Toolathlon	**73**	**88**	τ^2	15	9
cross − in-catalog		0	+8		−3	−12

its no-graph control, so a graph compiled on one catalog transfers in both directions, even when its source is the narrower catalog.

The asymmetry is consistent with distillation into an artifact: a large, heterogeneous catalog yields many recurring dependency patterns (look-ups before writes, identifiers resolved before use) on tools analogous to a narrower catalog's, and a narrow one yields fewer. This mirrors In-N-Out, where a builder fine-tuned on one benchmark's curated API graphs generalizes to another (Lee et al., 2026), and the finding that abstract knowledge from a larger pool transfers better (Kim et al., 2026); here nothing is trained, and any frozen planner can read the graph. Two caveats bound the claim. Source and subgraph size change together (τ^2 tasks receive larger subgraphs from Toolathlon's graph than from their own) and we run no size-matched control. And the evidence is one catalog pair, two backbones, and one seed: $+8$ on 100 tasks is about 1.5 standard errors of an unpaired difference, a strong directional signal rather than a settled effect size.

8 CONCLUSION

TOOLGRAPH compiles a catalog's dependency structure once, from interfaces and counterfactual probes, into an artifact a frozen planner can read. It improves from its own execution outcomes, beats tree search at 6–29× fewer LLM calls, and carries structure from a large, diverse catalog to one it has never seen; across readers, the one that combines graph evidence with the concrete state does best. Appendix A discusses limitations.

REPRODUCIBILITY STATEMENT

Section 3 and Eqs. 1–2 specify construction, Section 4 and Eq. 3 specify every reader, and Section 5 specifies the CRUD rules. Appendix Table 9 lists all constants, and Appendix Table 6 lists the per-backbone settings. All three benchmarks are public, and we state our splits in Section 6. We will release the compiled graphs, the prompt set, and per-task outcome records as anonymized supplementary material. The prompt templates are in Appendix E.

ETHICS STATEMENT

Tool-use agents act on external systems. Graph construction does not call any tool: probe queries are synthesized from interfaces and their outcomes are simulated by an LLM, so no external action is taken and no personal data is read during construction. Evaluation runs inside the benchmarks' own environments.

LLM USAGE STATEMENT

LLMs are part of the method. They derive probe tasks, match capabilities, synthesize and evaluate probes during construction, act as planners, scorers, verbalizers, and outcome judges at test time, and simulate users in τ^2-Bench, all as described above. We also used AI assistants for literature search, editing prose, and writing plotting code. The authors checked every reference against its source and reviewed all AI-assisted content, and they take full responsibility for the paper.

REFERENCES

Anthropic. System card: Claude Opus 4.6. https://www.anthropic.com/claude-opus-4-6-system-card, February 2026. Accessed 2026-09-26.

Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. RESTler: Stateful REST API fuzzing. In *Proceedings of the 41st IEEE/ACM International Conference on Software Engineering (ICSE)*, pp. 748–758, 2019. doi: 10.1109/ICSE.2019.00083. Preprint: arXiv:1806.09739.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2506.07982.

Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001. doi: 10.1016/S0004-3702(01)00108-4.

Guoxin Chen, Zhong Zhang, Xin Cong, Fangda Guo, Yesai Wu, Yankai Lin, Wenzheng Feng, and Yasheng Wang. Learning evolving tools for large language models. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2410.06617.

Wenjie Chen, Di Yao, Wenbin Li, Xuying Meng, Chang Gong, and Jingping Bi. GTool: Graph enhanced tool planning with large language model. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2508.12725.

Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, Vincent Cohen-Addad, Mohammadhossein Bateni, Chen-Yu Lee, and Tomas Pfister. Re-Invoke: Tool invocation rewriting for zero-shot tool retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 4705–4726. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-emnlp.270.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V. Le, Sergey Levine, and Yi Ma. SFT memorizes, RL generalizes: A comparative study of foundation model post-training. In *International Conference on Machine Learning (ICML)*, 2025. arXiv:2501.17161.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025. URL https://arxiv.org/abs/2512.02556.

Yue Fang, Zhibang Yang, Fangkai Yang, Xiaoting Qin, Liqun Li, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. ToolAtlas: Learning once, reusing everywhere with tool-side memory. *arXiv preprint arXiv:2607.11126*, 2026.

Tao Feng, Chongrui Ye, Tianyang Luo, Jingjun Xu, Xueqiang Xu, Haozhen Zhang, Zhigang Hua, Yan Xie, Shuang Yang, Ge Liu, and Jiaxuan You. ExpGraph: Model-agnostic experience learning with graph-structured memory for LLM agents. *arXiv preprint arXiv:2605.30712*, 2026.

Linfeng Gao, Yaoxiang Wang, Minlong Peng, Jialong Tang, Yuzhe Shang, Mingming Sun, and Jinsong Su. Tool graph retriever: Exploring dependency graph-based tool retrieval for large language models. *arXiv preprint arXiv:2508.05152*, 2025.

Xinyi Gao, Xinyu Ren, Junliang Yu, Tong Chen, Quoc Viet Hung Nguyen, and Hongzhi Yin. GRAFT: Graph-tokenized LLMs for tool planning. *arXiv preprint arXiv:2605.11706*, 2026.

Gemma Team. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026. URL https://arxiv.org/abs/2607.02770.

Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. Is your LLM secretly a world model of the internet? model-based planning for web agents. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. arXiv:2411.06559.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From RAG to memory: Non-parametric continual learning for large language models. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 21497–21515, 2025. arXiv:2502.14802.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. arXiv:1706.02216.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8154–8173. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.507. arXiv:2305.14992.

Taher H. Haveliwala. Topic-sensitive PageRank. In *Proceedings of the 11th International Conference on World Wide Web (WWW)*, pp. 517–526. ACM, 2002. doi: 10.1145/511446.511513.

Jörg Hoffmann. Where "ignoring delete lists" works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, 24:685–758, 2005. doi: 10.1613/jair.1747.

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001. doi: 10.1613/jair.855.

Joey Hong, Anca Dragan, and Sergey Levine. Planning without search: Refining frontier LLMs with offline goal-conditioned RL. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2505.18098.

Jerry Huang, Prasanna Parthasarathi, Mehdi Rezagholizadeh, and Sarath Chandar. Towards practical tool usage for continually learning LLMs. *arXiv preprint arXiv:2404.09339*, 2024.

Jingyi Jia and Qinbin Li. AutoTool: Efficient tool selection for large language model agents. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 40, pp. 31265–31273, 2026. doi: 10.1609/aaai.v40i37.40389. arXiv:2511.14650.

Yan Jiang, Hao Zhou, Lizhong Gu, Tianlong Li, Ruinan Jin, Wanqi Zhou, and Ai Han. NaviAgent: Graph-driven bilevel planning for scalable tool orchestration. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. arXiv:2506.19500.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2405.14831.

Kangsan Kim, Minki Kang, Taeil Kim, Yanlai Yang, Mengye Ren, and Sung Ju Hwang. Memory transfer learning: How memories are transferred across domains in coding agents. *arXiv preprint arXiv:2604.14004*, 2026.

Jing Yu Koh, Stephen Marcus McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. arXiv:2407.01476.

Seungkyu Lee, Nalim Kim, and Yohan Jo. In-N-Out: A parameter-level API graph dataset for tool agents. *Transactions of the Association for Computational Linguistics*, 14:1008–1027, 2026. doi: 10.1162/tacl.a.694.

Jiaxing Li, Lei Song, Rui Dong, and Youyong Kong. Learning from failures: Heterogeneous graph memory for small language model tool-using agents. *arXiv preprint arXiv:2609.28003*, 2026a.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, Graham Neubig, and Junxian He. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. In *International Conference on Learning Representations (ICLR)*, 2026b. arXiv:2510.25726.

Sijia Li, Yuchen Huang, Zifan Liu, Zijian Li, Jingjing Fu, Lei Song, Jiang Bian, Jun Zhang, and Rui Wang. Experience-evolving multi-turn tool-use agent with hybrid episodic–procedural memory. In *International Conference on Machine Learning (ICML)*, 2026c. arXiv:2512.07287.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2305.20050.

Alexander H. Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, et al. Ministral 3. *arXiv preprint arXiv:2601.08584*, 2026. URL https://arxiv.org/abs/2601.08584.

Shengjie Liu, Li Dong, and Zhenyu Zhang. Bridging tool dependencies and domain knowledge: A graph-based framework for in-context planning. In *NeurIPS 2025 Workshop on Bridging Language, Agent, and World Models for Reasoning and Planning (LAW)*, 2025a. Non-archival workshop paper. arXiv:2510.24690.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. ToolACE: Winning the points of LLM function calling. In *International Conference on Learning Representations (ICLR)*, 2025b. arXiv:2409.00920.

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. *arXiv preprint arXiv:2403.00839*, 2024a.

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*, Lecture Notes in Computer Science, pp. 89–105. Springer, 2024b. doi: 10.1007/978-3-031-73254-6_6. arXiv:2310.17796.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. APIGen: Automated pipeline for generating verifiable and diverse function-calling datasets. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024c. arXiv:2406.18518.

Elias Lumer, Anmol Gulati, Vamse Kumar Subbiah, Pradeep Honaganahalli Basavaraju, and James A. Burke. ScaleMCP: Dynamic and auto-synchronizing model context protocol tools for LLM agents. *arXiv preprint arXiv:2505.06416*, 2025.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter 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. *arXiv preprint arXiv:2310.10134*, 2023.

Ethan Mendes and Alan Ritter. Language models can self-improve at state-value estimation for better search. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. Spotlight. arXiv:2503.02878.

Vaskar Nath, Pranav Raja, Claire Yoon, and Sean Hendryx. ToolComp: A multi-tool reasoning & process supervision benchmark. *arXiv preprint arXiv:2501.01290*, 2025.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2509.25140.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the Web. Technical Report 1999-66, Stanford InfoLab, 1999.

Bhrij Patel, Davide Belli, Amir Jalalirad, Maximilian Arnold, Aleksandr Ermolov, and Bence Major. Dynamic tool dependency retrieval for lightweight function calling. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 33649–33672. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1680.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2305.15334.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2307.16789.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. From exploration to mastery: Enabling LLMs to master tools via self-driven interactions. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2410.08197.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning (ICML)*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1312–1320, 2015.

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 (NeurIPS)*, 2023. Oral. arXiv:2302.04761.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. TaskBench: Benchmarking large language models for task automation. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024. arXiv:2311.18760.

Jeonghoon Shim, Gyuhyeon Seo, Cheongsu Lim, and Yohan Jo. ToolDial: Multi-turn dialogue generation method for tool-augmented language models. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2503.00564.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2303.11366.

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 *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2408.03314.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.eacl-long.333. arXiv:2504.07952.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, Ge Zhang, Jiaheng Liu, Xingyao Wang, Sirui Hong, Chenglin Wu, Hao Cheng, Chi Wang, and Wangchunshu Zhou. Agent KB: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*, 2025.

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. LLMs in the imaginarium: Tool learning through simulated trial and error. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10583–10604. Association for Computational Linguistics, 2024a. doi: 10.18653/v1/2024.acl-long.570. arXiv:2403.04746.

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. *Transactions on Machine Learning Research*, 2024b. arXiv:2305.16291.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. ToolGen: Unified tool retrieval and calling via generation. In *International Conference on Learning Representations (ICLR)*, 2025a. arXiv:2410.03439.

Zhenting Wang, Qi Chang, Hemani Patel, Shashank Biju, Cheng-En Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Shah, Yujia Bao, and Eugene Siow. MCP-Bench: Benchmarking tool-using LLM agents with complex real-world tasks via MCP servers. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2508.20453.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. In *Conference on Language Modeling (COLM)*, 2025b. arXiv:2504.06821.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911, 2025c. arXiv:2409.07429.

Xixi Wu, Yifei Shen, Caihua Shan, Kaitao Song, Siwei Wang, Bohang Zhang, Jiarui Feng, Hong Cheng, Wei Chen, Yun Xiong, and Dongsheng Li. Can graph learning improve planning in LLM-based agents? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2405.19119.

Yu Xia, Jingru Fan, Weize Chen, Siyu Yan, Xin Cong, Zhong Zhang, Yaxi Lu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. AgentRM: Enhancing agent generalization with reward modeling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19277–19290. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.945.

Yang Xiao, Mohan Jiang, Jie Sun, Keyu Li, Jifan Lin, Yumin Zhuang, Ji Zeng, Shijie Xia, Qishuo Hua, Xuefeng Li, Xiaojie Cai, Tongyu Wang, Yue Zhang, Liming Liu, Xia Wu, Jinlong Hou, Yuan Cheng, Wenjie Li, Xiang Wang, Dequan Wang, and Pengfei Liu. LIMI: Less is more for agency. *arXiv preprint arXiv:2509.17567*, 2025.

Zhangchen Xu, Adriana Meza Soria, Shawn Tan, Anurag Roy, Ashish Sunil Agrawal, Radha Poovendran, and Rameswar Panda. TOUCAN: Synthesizing 1.5M tool-agentic data from real-world MCP environments. *arXiv preprint arXiv:2510.01179*, 2025.

Shuo Yang, Soyeon Caren Han, Yihao Ding, Shuhe Wang, and Eduard Hovy. ToolTree: Efficient LLM tool planning via dual-feedback monte carlo tree search and bidirectional pruning. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2603.12740.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 11809–11822, 2023a. arXiv:2305.10601.

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 *International Conference on Learning Representations (ICLR)*, 2023b. arXiv:2210.03629.

Xiuhui You, Jiayi Luo, Zichao Shen, Qingyun Sun, and Ziwei Zhang. ToolLIFT: Lifting tool-specific trajectories into function-level graphs for generalizable tool planning. *arXiv preprint arXiv:2608.03468*, 2026.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.181. arXiv:2310.12823.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *International Conference on Learning Representations (ICLR)*, 2025a. arXiv:2408.15240.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2510.04618.

Zijing Zhang, Zhanpeng Chen, He Zhu, Ziyang Chen, Nan Du, and Xiaolong Li. ToolExpNet: Optimizing multi-tool selection in LLMs with similarity and dependency-aware experience networks. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 15706–15722. Association for Computational Linguistics, 2025b. doi: 10.18653/v1/2025.findings-acl.811.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936. arXiv:2308.10144.

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 *International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62138–62160, 2024.

King Zhu, Hanhao Li, Siwei Wu, Tianshun Xing, Dehua Ma, Xiangru Tang, Minghao Liu, Jian Yang, Jiaheng Liu, Yuchen Eleanor Jiang, Changwang Zhang, Chenghua Lin, Jun Wang, Ge Zhang, and Wangchunshu Zhou. Scaling test-time compute for LLM agents. *arXiv preprint arXiv:2506.12928*, 2025.

Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A. Rossi, Somdeb Sarkhel, and Chao Zhang. ToolChain*: Efficient action space navigation in large language models with A* search. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.13227.

A Limitations

The CRUD experiment updates the graph over a fixed catalog. Adding and deleting tools is costed (Table 8) but not exercised in the update stream, and we do not report Create and Delete firing counts. The Update step rewards or penalizes the edges between tools that appear in a trajectory according to whether that trajectory passed. We have not run a control that updates the same edges regardless of outcome, so we cannot yet separate how much of the CRUD gain comes from learning which tool transitions *succeed* from how much comes from learning which transitions the training tasks merely *use*. Transfer covers one catalog pair. Like any delete relaxation, ours ignores consumed resources and negative interactions (Hoffmann, 2005); our entity model also ignores conditional effects and cannot separate calls to one tool with different arguments. Probe outcomes are simulated and scored by an LLM rather than obtained by executing the tools, so an edge weight is evidence of a dependency rather than proof. The construction models and the verifier are stronger than some planners, so the graph and the scorer may also carry knowledge from those models.

B Ablations

Verifier. Replacing the Gemma verifier with Claude Opus 4.6 on τ^2 (Table 4) leaves flat retrieval, which makes no verifier call, unchanged in all three cells, as it should. The stronger verifier lifts Q_{llm} to $61/79/88$ (Mean 76.0), and it stays the best arm on every backbone. It also reverses the Gemma-verifier ordering of Q_{dp} and TG-PPR on G and D, so that Q_{dp} beats TG-PPR on all three planners. Q_{dp} makes no model call, so that reversal must enter through the verbalizer, the only verifier-written component on the Q_{dp} path.

Evidence budget. On τ^2, the sweep over h, θ, annotation depth, and n (Table 5) is non-monotone in every knob, and every peak sits at or next to the anchor. Pruning edges to save tokens does not pay there: raising θ to 0.9 removes over half the edges, saves 9.5K tokens, and costs 2.5 points. On ToolComp's small graph, every $\theta > 0$ scores slightly above $\theta = 0$, within the replicate spread.

Table 4: **Verifier ablation on τ^2-Bench with Claude Opus 4.6 as verifier**; compare the τ^2 block of Table 1. Flat retrieval makes no verifier call and is identical across the two verifiers. Bold marks the column best.

Method	M	G	D	Mean
Reflexion	57	64	71	64.0
Flat top-k retrieval	34	62	72	56.0
TG top-k PPR	38	65	75	59.3
TG top-k Q_{dp}	46	72	81	66.3
TG top-k Q_{emb}	32	65	80	59.0
TG top-k Q_{llm}	**61**	**79**	**88**	**76.0**

Table 5: **Evidence-budget sweep for Q_{llm}** on the evaluation splits, reported as analysis; the configurations used elsewhere were selected on the training split (Table 6). Each block varies one knob and holds the rest at the anchor (h=1, θ=0.0, N0, n=4; marked *). Success is a macro average over the three planners; nodes/edges give the mean retrieved subgraph; Tok.K is test-time tokens per task, in thousands. Bold marks the best level per block and benchmark.

	τ^2-Bench			ToolComp		
Level	nodes/edges	Succ.	Tok.K	nodes/edges	Succ.	Tok.K
1 — subgraph radius h						
h=0	7.7/7.0	64.4	136.9	6.4/11.5	**69.8**	78.4
h=1 *	24.1/159.5	**65.8**	155.2	11/35	67.3	79.0
h=2	24.3/160.3	64.5	155.2	11/35	69.5	79.0
h=3	24.3/160.3	63.6	155.2	11/35	66.1	79.0
2 — edge threshold θ						
θ=0.0 *	24.1/159.5	**65.8**	155.2	11/35	67.3	79.0
θ=0.5	24.1/149.4	65.3	153.9	11/28	**69.8**	78.9
θ=0.6	24.1/138.4	63.6	152.7	11/28	69.2	78.9
θ=0.7	24.1/131.4	63.0	152.0	11/27	68.4	78.9
θ=0.8	24.1/125.0	63.6	151.3	11/27	67.8	78.9
θ=0.9	24.1/73.7	63.3	145.7	11/15	68.9	78.7
3 — per-node annotation depth						
N0 edges only *	24.1/159.5	65.8	155.2	11/35	67.3	79.0
N1 + input/output fields	24.1/159.5	**66.4**	158.5	11/35	68.6	79.8
N2 + structural features	24.1/159.5	64.2	161.1	11/35	69.2	80.5
N3 + runtime history	24.1/159.5	63.3	163.7	11/35	**70.7**	81.2
4 — scored candidates n (test-time compute)						
n=1 (verbalizer only)	24.1/159.5	61.1	67.7	11/35	66.6	43.9
n=4 *	24.1/159.5	**66.4**	155.2	11/35	69.0	79.0
n=8	24.1/159.5	64.7	271.8	11/35	**69.4**	125.8

Replicate spread. The anchor reads 65.8 in blocks 1–3 and 66.4 in block 4 on τ^2 (67.3 versus 69.0 on ToolComp) for a nominally identical configuration. On ToolComp, h=1, 2, 3 retrieve the identical whole graph and score 67.3, 69.5, and 66.1. Effects of a few points in blocks 1–3 are within this spread.

Table 6: **Per-backbone Q_{llm} settings** used in Tables 1, 2, and 3, selected by a sweep on each benchmark's training split.

	ToolComp			τ^2-Bench			Toolathlon		
Knob	M	G	D	M	G	D	M	G	D
Radius h	1	1	1	1	3	2	1	3	2
Edge threshold θ	0	0	0.5	0.5	0.5	0.8	0.5	0.6	0.8
Annotation depth	N2	N3	N1	N0	N1	N1	N0	N1	N1
Candidates n	4	4	4	4	8	8	4	8	8

C GRAPH STATISTICS AND CONSTRUCTION COST

Table 7: **Compiled graph statistics.** Evidence source partitions the retained edges by proposer.

	Nodes			Edges		Evidence source			
Catalog	atomic	meta	total	cand.	kept	param.	probe	both	θ_{prune}
ToolComp	7	4	11	40	35	0	28	7	0.10
τ^2-Bench	102	44	146	1,508	1,303	59	1,041	203	0.30
Toolathlon	576	42	618	11,829	10,226	394	7,145	2,675	0.30

Average in-/out-degree, atomic and meta: ToolComp $1.3/3.1$ and $6.5/3.2$; τ^2 $5.9/6.8$ and $16.0/13.9$; Toolathlon $13.8/13.9$ and $54.7/53.5$. Evaluated τ^2 domains (intra-domain nodes/edges): airline $14/79$, retail $16/93$, telecom $43/309$. The Toolathlon evidence columns sum to $10{,}214$; the other 12 kept edges were added by hand for two general-purpose local tools.

Table 8: **Construction and maintenance cost** in LLM calls. Construction is a one-time, catalog-level cost. The two blocks come from two separate construction runs with different instrumentation: a pipeline-level count over all three catalogs, and a phase-level count, with token estimates, over two. They disagree in both directions, so we report both rather than reconcile them.

	Toolathlon	ToolComp	τ^2-Bench
Pipeline-level count			
Construction	1,240	980	1,105
Add one tool / delete one tool	12 / 8	10 / 7	11 / 7
Phase-level count			
Schema edge evaluation	–	21	1,266
Blind task derivation	–	7	102
Capability matching	–	28	4,488
Information gain (probe evaluation)	–	65	2,400
Total calls (tokens)	–	121 (120K)	8,256 (8.8M)
Add one tool / delete one tool	–	14 / 7	84 / 7
Amortized construction tokens per evaluation task	–	0.6K	88K

D SETTINGS

Table 9: **Settings.**

Component	Parameter	Value	Role
Edge construction	τ	0.15	Sigmoid temperature of the likelihood
	θ_{match}	0.30	Capability-matching threshold
	n_e	3	Task variants per required entity or output field
	Models	Sonnet 4.5 (gen., match, schema); Sonnet 4 (eval.)	Temperature 0
Pruning	θ_{prune}	0.10 ToolComp; 0.30 τ^2, Toolathlon	Edge instantiation threshold
CRUD	Initial counts	$\alpha_0{=}s\,w$, $\beta_0{=}s(1{-}w)$; $s{=}10$ schema, 5 probe	Compiled edges enter with weight w
	Update increments	pass $\alpha{+}=1.0$; fail $\beta{+}=0.3$	Beta-mean edge weight
	Delete rule	$n_{ij}\geq 5$ and $w_{ij}<0.05$	Prunes edges that prove unreliable
	n_{create}	2; new edges start at $\alpha{=}1$, $\beta{=}0.5$	Passing co-occurrences before an edge is created
Q_{dp}	γ, κ	0.95, 0.3	Discount; confidence penalty
	Tolerance / sweep cap	10^{-6} / 10^4	Value iteration
	State cap	10^5	Enumerated entity sets; solver raises on overflow
Q_{emb}	Architecture	2 GraphSAGE layers (mean), hidden 256, dropout 0.1	Inductive node encoder
	Training	Adam, lr 10^{-3}, wd 10^{-5}, batch 64, 200 epochs	MSE to normalized Q_{dp}
Q_{llm}	Samples	5 at temperature 1.0	Integer score in $[0, 10]$, averaged, $/10$; one comparative ranking call when several candidates are eligible
Injection / PPR	k	Toolathlon: top 10 (≤ 2 per server); ToolComp: all tools ranked; τ^2: top 4 candidates, subgraph ≤ 50 edges	Injected rows
	α	0.85 ToolComp, τ^2; 0.65 Toolathlon	PPR teleport probability
	$(\lambda_{\text{int}}, \lambda_{\text{out}}, \lambda_{\text{ent}})$	ToolComp $(0.1, 0, 0.9)$; τ^2 $(0.3, 0.7, 0)$; Toolathlon $(0.9, 0.1, 0)$	Personalization mixture

E PROMPTS

Construction prompts are sent at temperature 0; placeholders in braces are filled at run time, and doubled braces reach the model as literal JSON braces. Tool schemas are rendered as the tool's name, description, and the names, types, and descriptions of its input and output parameters.

Blind task derivation ($\mathcal{L}_{\text{gen}}$), system prompt.

```
You are a task analyst. Given an atomic tool's schema, derive blind tasks -- tasks that make
    NO reference to any specific tool.

For each input parameter, generate 3 DIVERSE upstream tasks that would produce the value
    needed for that parameter. Vary the method (search, lookup, computation, user input, API
     call, etc.).

For each output field, generate 3 DIVERSE downstream tasks that would naturally consume that
    field.
```

```
Also state the canonical same-task: what single task does this tool solve?

Return ONLY valid JSON in this exact structure:
{{
  "pre_tasks": [
    {{"entity": "<param_name>", "variants": ["task1", "task2", "task3"]}}
  ],
  "same_task": "<canonical task description>",
  "post_tasks": [
    {{"field": "<field_name>", "variants": ["task1", "task2", "task3"]}}
  ]
}}
```

Blind task derivation ($\mathcal{L}_{\text{gen}}$), user prompt.

```
Analyze this atomic tool and derive blind tasks.

{tool_schema}

For each INPUT parameter, generate 3 diverse upstream tasks that would produce the value
     needed for that parameter. Make them tool-agnostic -- do NOT name any specific tool.

For each OUTPUT field, generate 3 diverse downstream tasks that would naturally consume that
     field's value.

Also state the canonical same-task this tool solves.
```

Capability matching ($\mathcal{L}_{\text{match}}$), system prompt.

```
You are a capability assessor. Given a meta tool and a set of derived tasks, judge whether
     the meta tool could handle each task.

Score each task from 0.0 to 1.0:
- 1.0 = the meta tool is perfectly suited for this task
- 0.5 = plausible but not ideal
- 0.0 = completely unrelated

For pre_tasks: could the meta tool perform the upstream task to PRODUCE the needed entity?
For post_tasks: could the meta tool perform the downstream task that CONSUMES the output
     field?
For same_task: does the meta tool solve essentially the same task?

IMPORTANT: OMIT any entry where all variant_scores are 0.0. Only include entries with at
     least one non-zero score. This keeps the output compact.

Return ONLY valid JSON:
{{
  "pre_scores": [
    {{"entity": "<name>", "variant_scores": [0.8, 0.5, 0.1], "best_score": 0.8, "best_variant
     ": 0, "reasoning": "..."}}
  ],
  "same_score": 0.1,
  "post_scores": [
    {{"field": "<name>", "variant_scores": [0.3, 0.7, 0.1], "best_score": 0.7, "best_variant
     ": 1, "reasoning": "..."}}
  ]
}}
```

Capability matching ($\mathcal{L}_{\text{match}}$), user prompt.

```
## Meta Tool
{meta_tool_schema}

## Atomic Tool (for context -- the tasks below were derived from this tool's schema)
Name: {atomic_name}
Description: {atomic_description}

## Derived Tasks
PRE-TASKS (upstream tasks that produce inputs for the atomic tool):
  {entity}[{i}]: {task}
SAME-TASK: {same_task}
POST-TASKS (downstream tasks that consume outputs of the atomic tool):
  {field}[{i}]: {task}

Score each task: can the meta tool handle it? (0.0-1.0)
```

Probe evaluation ($\mathcal{L}_{\text{eval}}$), system prompt.

```
You are an information gain estimator. Given a probe query and two tools, estimate how much
     chaining them together reduces uncertainty compared to using either alone.

CRITICAL -- apply these three checks IN ORDER. If ANY check fails, ig_score MUST be 0 or
     negative.

CHECK 1 -- Trivial user input: Could the user simply provide the needed input directly?
  Parameters like city_name, country_code, month, keywords, symbol are values the user
     ALREADY KNOWS when they initiate a request. No tool is needed to "find" them.
  - "Look up ISO country code" -> the user just types "US". ig_score = 0.
  - "Extract city name from query" -> the user already said which city. ig_score = 0.
  - "Find the month for the query" -> the user specifies the month. ig_score = 0.
  ONLY score positively if the first tool resolves something the user genuinely does NOT know
      and CANNOT trivially provide (e.g., a computed result, a database record, live data
     feed).

CHECK 2 -- Subsumption: Does the first tool already do what the second tool does?
  If tool A can ALREADY compute/solve what tool B does, chaining A->B is pointless.
  - python_interpreter -> calculator: Python already evaluates math. ig_score = 0.
  - wolfram_alpha -> calculator: Wolfram already computes. ig_score = 0.
  A general-purpose tool (interpreter, search engine, computational engine) that can do
     EVERYTHING the specialized tool does makes the chain redundant.

CHECK 3 -- Capability reality: Can the first tool ACTUALLY perform the claimed task?
  Do not give credit for capabilities a tool does NOT have.
  - python_interpreter cannot look up stock ticker symbols -- it has no API access.
  - google_search returns web page links, not structured data extraction.
  - Processing error messages is not a meaningful data dependency.
  If the first tool lacks the ACTUAL capability to produce the needed output, ig_score = 0.

You will be given:
- A probe query Q (the task to be accomplished)
- Tool A (meta tool) with its schema
- Tool B (atomic tool) with its schema
- A proposed chain direction

Estimate:
- baseline_resolves: how well the second tool alone resolves Q given typical user input
     (0.0-1.0)
- chain_resolves: how well the full chain resolves Q (0.0-1.0)
- ig_score: chain_resolves minus baseline_resolves (NEGATIVE if chain is redundant)

Return ONLY valid JSON:
{{
  "baseline_resolves": 0.4,
  "chain_resolves": 0.85,
  "ig_score": 0.45,
  "simulated_baseline_output": "what the second tool returns given direct user input",
  "simulated_chain_output": "what the chain returns when the first tool runs before",
  "reasoning": "why chaining helps or doesn't"
}}
```

Probe evaluation ($\mathcal{L}_{\text{eval}}$), user prompt.

```
## Probe Query
{probe_task}

## Meta Tool
{meta_tool_schema}

## Atomic Tool
{atomic_tool_schema}

## Direction: {direction}
{chain_description}
{baseline_description}

Simulate the outputs and estimate information gain.

where, for direction "pre":
  Chain: meta tool '{meta}' runs first to gather info, then atomic tool '{atomic}' runs using
      that info.
  Baseline: atomic tool '{atomic}' runs alone.
and for direction "post":
  Chain: atomic tool '{atomic}' runs first, then meta tool '{meta}' processes the result.
  Baseline: meta tool '{meta}' runs alone on the query.
```

Same-class probe (atomic–atomic and meta–meta pairs, single call).

```
You are evaluating whether running Tool A before Tool B creates a useful information flow.

Tool A (runs first):
{tool_a_brief}

Tool B (runs second):
{tool_b_brief}

Evaluate this directed pair A -> B:

1. **match_score** (0.0-1.0): How likely is it that A's output provides useful information/
     context that helps parameterize, enable, or improve B's execution?
   - 0.0 = completely unrelated, no information flow
   - 0.3 = weak/indirect relationship
   - 0.6 = moderate relationship, A's output partially informs B
   - 0.9+ = strong dependency, A's output directly feeds into B

2. **ig_score** (0.0-1.0): If a user task requires BOTH tools, how much does running A first
     (vs B first or independently) improve the overall result?
   - 0.0 = order doesn't matter
   - 0.5 = slight benefit to running A first
   - 0.9+ = critical that A runs before B

3. **reasoning**: 1-2 sentences explaining the information flow.

Consider:
- Does A's output contain data that B needs as input?
- Does A's output provide context that helps formulate B's parameters?
- In a multi-tool workflow, would A naturally precede B?
- Even if no direct data dependency, does A provide information that a human/agent would use
     to decide how to call B?

Respond with ONLY this JSON:
{{
  "match_score": <float>,
  "ig_score": <float>,
  "reasoning": "<string>"
}}
```

Parameter matching (schema edge evaluation), system prompt.

```
You are a tool dependency analyst. Given two tools A and B with their schemas, determine if
     there is a directed dependency edge between them.

An edge A -> B means: "A should be invoked BEFORE B because A's output produces data that B
     needs as input."
An edge B -> A means the reverse.

Think about:
1. Does A's output contain fields/entities that B requires as input? (even if named
     differently)
2. Is there a realistic workflow where a user would call A first, then use A's result to call
      B?
3. Could A's output be DIRECTLY passed (or trivially extracted) to satisfy B's input?

Do NOT create an edge just because two tools are in the same domain. The dependency must be
     concrete: output data flows from one tool's result into another tool's parameters.

Return ONLY a JSON object with this exact structure:
{
  "A_to_B": {
    "has_edge": true or false,
    "confidence": 0.0 to 1.0,
    "param_matches": [{"from_output": "field_name", "to_input": "param_name", "reasoning": "
     brief"}],
    "reasoning": "one sentence explaining the dependency or why none exists"
  },
  "B_to_A": {
    "has_edge": true or false,
    "confidence": 0.0 to 1.0,
    "param_matches": [{"from_output": "field_name", "to_input": "param_name", "reasoning": "
     brief"}],
    "reasoning": "one sentence explaining the dependency or why none exists"
  }
}
```

Parameter matching (schema edge evaluation), user prompt.

```
## Tool A
{tool_a_schema}
```

```
## Tool B
{tool_b_schema}

Analyze whether A's output can feed into B's input (A->B edge) and whether B's output can
     feed into A's input (B->A edge).
```

Q_{llm} scorer (sections are omitted when empty).

```
You are a value function for a tool-using agent. You estimate how well an action advances
     toward successful task completion, considering both structural evidence from a causally-
     probed tool dependency graph AND policy/conversation context. Chain rates come from
     interventional experiments (not log co-occurrence).

DOMAIN POLICY:
{policy_text}

USER'S STATED GOAL:
{user_ask}

CONVERSATION SO FAR:
{history}

CURRENT STATE:
{state}

INFERRED TARGET (structural hypothesis):
{inferred_goal}

SUBGRAPH CONTEXT:
{subgraph}

CANDIDATE ACTION:
{candidate}

STRUCTURAL EVIDENCE (from tool dependency graph):
{tool_feature_blocks}

ALTERNATIVE ACTIONS AVAILABLE:
{alternatives}

GRAPH DP ESTIMATE: {q_dp_prior} (path reachability score)

Considering ALL evidence above (policy, conversation, state, and structural graph features),
     rate how well this action advances toward successful task completion on a scale of 0-10:
  10 = optimal action given policy, state, and graph structure
  7-9 = strong choice, on critical path, policy-compliant
  4-6 = feasible but suboptimal (indirect, or better alternatives exist)
  1-3 = poor choice (policy violation, wrong timing, or structurally blocked)
  0 = impossible, harmful, or completely unrelated
A correct clarification or refusal counts as high when policy requires it.
Respond with ONLY a JSON object: {"score": <integer 0-10>, "reason": "<one sentence>"}
```

Verbalizer.

```
You are a critic for a tool-using agent. Given Q-values across multiple goals, recommend the
     best action and explain why.

USER'S REQUEST:
{user_ask}

CONVERSATION SO FAR:
{history}

CURRENT STATE:
{state}

INFERRED GOAL: {inferred_goal} (confidence: {goal_confidence})

Q-VALUES (likelihood of each action reaching each goal):
| candidate \ goal | {goal_1} | {goal_2} | ... |
| {candidate_1} | {q_11} | {q_12} | ... |

GRAPH EVIDENCE (tool dependency subgraph -- topology and features from the causally-probed
     ToolGraph):
{graph_evidence}

{SCORER} RANKING ({scorer_description}):
  rank {r}: {tool} (score {score})
```

```
Respond as JSON:
{
  "recommendation": {"action": "...", "reason": "..."},
  "feedback": "<what the agent should know for this step>",
  "should_refine": true/false
}
```

Block injected into the planner prompt at the next decision (τ^2).

```
[Graph-grounded action analysis from previous step]
Scorer ({scorer_id}) ranked candidates:
  #{rank}: {tools} (score {score})          (top 4 candidates)
Inferred goal: {inferred_goal} (confidence {goal_confidence})
Structural Q-values toward goal:
  {tool}: Q_dp={q}                          (top 5 tools)
Critic assessment: {feedback}
```