SELF-EVOLVING AGENTS VIA LIKELIHOOD-GUIDED TOOL-SPACE OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving agents can continually improve their behavior, while tools define the executable action space through which they interact with the environment. However, exposing the full tool library to a model introduces substantial irrelevant context and can impair tool-use decisions. We study tool-space self-evolution, where each recurring task type maintains a persistent tool space which is constructed from accumulated output experience. We identify three limitations of existing methods: (1) *output-unaware selection*: they rely primarily on tool descriptions or model priors rather than observed tool outputs; (2) *statelessness across requests*: they select tools independently for each request without consolidating prior output experience into persistent task-specific state; (3) *inference cost*: they repeatedly search, rank, or reason over candidate tools for subsequent requests of the same task. We address these limitations through output-aware tool scoring, persistent task-specific tool spaces, amortized tool selection, and reusable configurations across models. We introduce `LOTS` (*Likelihood-Only Tool Scoring*), which evolves an agent's tool space from accumulated output experience while keeping model parameters fixed. After each request, `LOTS` holds the model's generated answer and estimates each tool's contribution by measuring how much the answer likelihood changes when its observed output is removed. These contributions are aggregated within each recurring task to rank tools and update its persistent space. Across three benchmarks, `LOTS` improves task performance while substantially reducing tool context. More importantly, sequential experiments demonstrate that task-specific spaces persist and continue to improve over time, while cross-model experiments show that learned configurations transfer across different models.

1 INTRODUCTION

Recent advances in large language models (LLMs) have enabled increasingly capable autonomous agents that can reason (Hurst et al., 2024), invoke external tools (Deng et al., 2025), and operate in complex environments (Anthropic, 2026). As these systems become more autonomous, an important capability is to improve from accumulated interaction experience rather than treating every request independently (Gao et al., 2025). Such *self-evolution* involves persistent updates to model parameters (Zhai et al., 2025; Acikgoz et al., 2026; Feng et al., 2026; Yang et al., 2026a), contextual state such as prompts and memories (Zhang et al., 2026a), tools and reusable skills (Wang et al., 2023; Yang et al., 2026b), or control architectures (Gao et al., 2025; Ren et al., 2026).

Among these components, the *tool space* is particularly important yet under-explored. It determines the agent's executable action space, while tool descriptions and schemas also constitute part of the context processed by the model at inference time. Unlike experience that can be stored and selectively retrieved when relevant, tool specifications are typically authored in advance and exposed to the model at request time (Schick et al., 2023; Patil et al., 2023; Qin et al., 2024). Consequently, as the global tool library expands, exposing it in full imposes a recurring cost: irrelevant tools occupy context even when they are not needed and enlarge the set of alternatives among which the model must choose. Prior work has shown that irrelevant or increasingly long context can impair language models' ability to identify and use relevant information reliably (Liu et al., 2023). This creates a structural mismatch: the global tool library may continually accumulate capabilities, whereas each recurring task typically requires only a small, task-specific subset. As shown in Figure 1, exposing an expanding global tool space in full increases context overhead and reduces accuracy, even when

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.

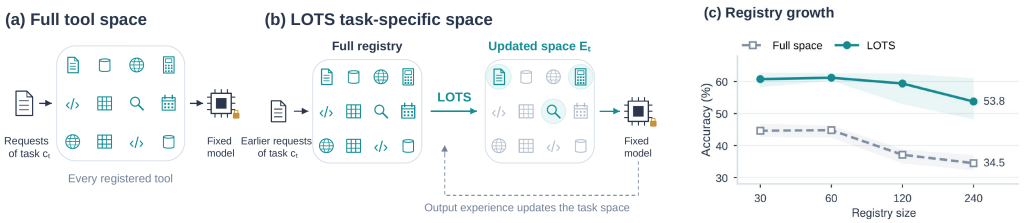

Figure 1: **A task-specific tool space that evolves from the agent's own outputs. (a)** A fixed model is served full registry on every request of a task. **(b)** *Likelihood-Only Tool Scoring* (LOTS) turns the output experience of a task's earlier requests into a compact space E_t. Later requests of the task reuse it, and their outputs update it. **(c)** Accuracy as the registry grows from 30 to 240 tools (TGB, Qwen3.5-9B): serving every tool falls to 34.5%, while the LOTS space stays at 53.8% or above (Appendix F.1).

task requirements remain unchanged. In contrast, maintaining a task-specific tool space substantially reduces context overhead and preserves higher accuracy as the global tool space grows. Additional results in Appendix F.1 (Figure 21) further support this finding.

Existing work on tool-oriented self-evolution has largely focused on creating new tools (Ding et al., 2026; Tam et al., 2026) or adapting their documentation, while comparatively little attention has been paid to evolving task-specific tool spaces from execution experience (Gao et al., 2025). More related works are in Appendix A. A natural starting point for constructing such a space is tool selection: determining which tools to retain from a candidate set. Existing approaches broadly follow two strategies, *Search-based* and *LLM-based methods* (Patil et al., 2023; Qin et al., 2024; Qu et al., 2024a; Hong et al., 2026; Zheng et al., 2024; Du et al., 2024; Moon et al., 2024; Fei et al., 2025). *Search-based methods* retrieve tools by matching requests to tool descriptions (Qin et al., 2024), but semantic relevance alone does not capture how effectively the serving model can use each tool. *LLM-based methods* use a language model to reason over candidate tools (Kachuee et al., 2025), or train it to select and reconfigure tools during execution (Zou et al., 2025; Zhou et al., 2026). These methods draw on the selector's understanding of tool functionality or model prior. Our focus is complementary: how can execution experience refine a persistent, task-specific tool space in an output-aware manner? To our knowledge, TTO (Lu et al., 2025) is the only prior method using output accuracy for tool selection, but it requires ground-truth labels and costly greedy search over tool compositions (as shown in Fig. 5). In summary, prior methods suffer from one or more of three key limitations: (1) *output-unaware*: Selection is primarily based on tool descriptions or model priors before the tool has run, and therefore does not directly measure whether the tool's observed output actually supported the resulting answer. (2) *statelessness across requests*: A tool-selection decision is typically made independently for each request. Output experience from previous requests of the same recurring task is not consolidated into persistent state that can shape how later requests are served. (3) *costly at inference time*: As the global registry (tool space) expands, tools must be repeatedly searched, ranked, or reasoned over for each new request, even when the deployed agent has already seen many requests from the same task. This motivates our central question:

> *Can an agent evolve a persistent tool space from its own output experience, so that later requests are served more accurately and efficiently?*

To answer this question, we introduce *Likelihood-Only Tool Scoring* (LOTS), which evolves persistent task-specific tool spaces from output experience. LOTS first makes tool optimization *output-aware*: it freezes the model's realized output and measures its likelihood change under a benchmark-specific intervention that removes evidence associated with each tool. It then aggregates these request-level contributions across recurring requests of the same task, ranks the tools by their accumulated evidence, and converts the ranking into a persistent task-specific space through tool pruning. This persistent space is reused to serve subsequent requests, whose new output traces can in turn revise the configuration. As a result, output evidence is retained across interactions and missing tools can be captured in the successive turns. We evaluate this design on TGB, BFCL, and GTA-Atomic: on TGB, a controlled compositional benchmark, LOTS keeps 2.8 to 6.8 of 15 tools on average and improves all eight evaluated models over serving every tool, by 3.8 to 25.0 accuracy points; on

BFCL, LOTS improves all five models; and on GTA-Atomic, tool space fitted on earlier requests improves six of seven models when later requests are executed by a ReAct agent with executable tools. The learned spaces also remain revisable as new traces arrive and can be reused across models. Our contributions are:

• **Output-aware tool optimization.** LOTS values tools from their observed outputs, using the likelihood change of the model's realized answer under leave-one-out intervention. This moves beyond description- or prior-based selection and yields tool spaces that generally improve over the full registry across the evaluated benchmarks and models.

• **Persistent task-specific evolution.** Rather than selecting tools independently for each request, LOTS aggregates output evidence across recurring requests into a persistent task-specific space and is reused by later requests and revised as new traces arrive.

• **Reduced serving context.** LOTS reuses compact task-specific spaces instead of searching or reasoning over the full registry for every request, reducing recurring tool context while maintaining or improving performance; Appendix F reports the one-time construction cost and the number of requests needed to recover it.

• **Tool-space reuse across models.** As an additional benefit, the optimized tool spaces learned from one model can often be reused by models from other scales and families.

2 Problem Formulation

Tool-use interaction. Let $\mathcal{S}$ denote the global tool registry, and $E \subseteq \mathcal{S}$ is the tool set available to the agent. Given a request x, the initial context h_0 contains the request and the specifications of tools in E. At each interaction step ℓ, the model generates an action $a_\ell \sim p_\theta(\cdot \mid h_{\ell-1})$, where a_ℓ is either a terminal response or a tool call (s_ℓ, z_ℓ) specifying a tool $s_\ell \in E$ and its arguments z_ℓ. A tool call produces an observation $o_\ell = s_\ell(z_\ell)$, which is appended to the context: $h_\ell = h_{\ell-1} \oplus (a_\ell, o_\ell)$. The agent continues this action–observation process until it produces the final answer $y_\theta(x; E)$. The tool space therefore determines both the agent's executable action space and the tool information available during inference.

Continual tool-space evolution. We consider a fixed language-model agent p_θ deployed with the global tool registry $\mathcal{S}$. The agent continually self-evolves as a sequence of tasks $c_1, \ldots, c_T$ arrives: for each incoming task c_t, it optimizes a task-specific tool space $E_t \subseteq \mathcal{S}$, whose optimum is

$$E_t^\star = \arg\max_{E \subseteq \mathcal{S}} \mathcal{U}(c_t; p_\theta, E),$$

where $\mathcal{U}$ measures task performance. As new tasks arrive, the agent continually expands its collection of task-specific spaces $\{E_1, \ldots, E_T\}$, and a task's space can be revised as more of its requests are served, improving performance through tool-space optimization while keeping p_θ fixed.

3 Method

We introduce LOTS for continually constructing task-specific tool spaces from an agent's own output experience while keeping the model parameters fixed. As tasks $c_1, \ldots, c_T$ arrive sequentially, each incoming task c_t provides a set of requests Q_t. As shown in Fig. 2, the agent first serves these requests under a candidate tool space and records the resulting tool-use traces. LOTS then (1) measures how much each observed tool output supports the model's realized answer, (2) aggregates this evidence across requests of the same task to obtain a task-level tool ranking, and (3) converts the ranking into a persistent task-specific tool space E_t. The resulting space is stored and reused when serving subsequent requests of c_t. As new tasks arrive, the deployed agent therefore continually expands its collection of task-specific spaces $\{E_1, \ldots, E_t\}$ without modifying the underlying model p_θ. Figure 2 summarizes the procedure; implementation details are given in Appendix B. LOTS can also be applied to a mix of tasks in a batch, with details in Appendix D.1.

3.1 LOTS: Output-Aware Tool Scoring Via Likelihood Guidance

For an incoming task c_t, let $Q_t = \{x_1, \ldots, x_{n_t}\}$ denote the requests used to construct its tool space. Consider a request $x \in Q_t$ served by the agent, and let y_x denote the model's realized output. We write h_x for the recorded context from which y_x is produced, including the request, available tool documentation, tool calls, and observed tool outputs. Our goal is to measure the contribution of

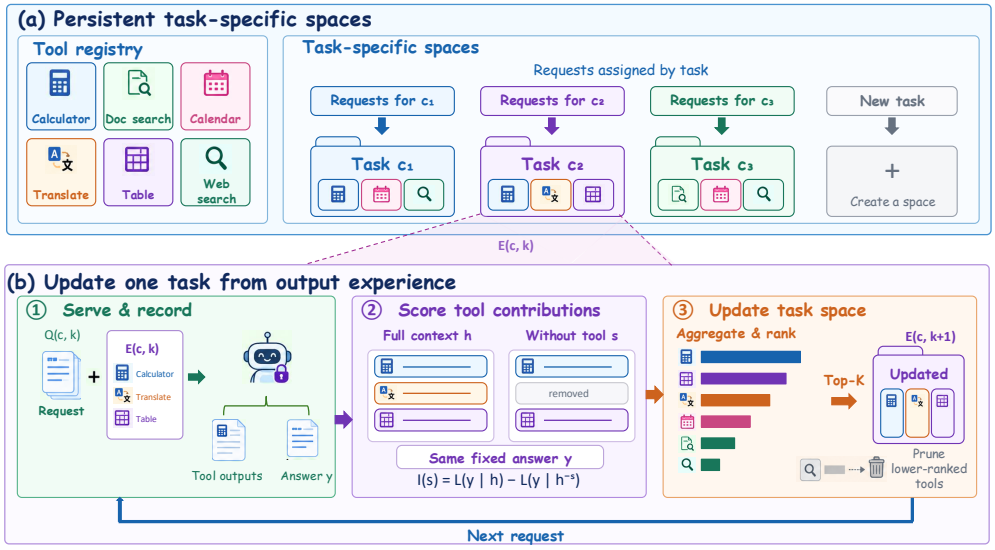

Figure 2: **LOTS evolves persistent task-specific tool spaces from output traces.** **(a)** Requests are assigned to recurring task types, each of which maintains its own space $E_{c,k}$. Requests of task c collected for an update are served under the current space. **(b)** One LOTS update freezes each generated output, scores tools by leave-one-out likelihood, aggregates the scores within the task, and uses the ranking to prune the task's tool space.

each tool from what it *actually produced during execution*. For a tool s, we construct an intervened context h_x^{-s} in which the output evidence attributable to s is removed from the recorded execution. We then keep the realized output y_x fixed and define the request-level contribution of s as

$$I_x(s) = \mathcal{L}_\theta(y_x \mid h_x) - \mathcal{L}_\theta(y_x \mid h_x^{-s}), \tag{1}$$

where

$$\mathcal{L}_\theta(y \mid h) = \frac{1}{|y|} \sum_{j=1}^{|y|} \log p_\theta(y_j \mid h, y_{<j}) \tag{2}$$

is the mean log-likelihood over the scored output tokens. A positive $I_x(s)$ means that removing the observed evidence from tool s reduces the model's support for the same realized output, indicating that the tool contributed useful evidence to that execution.

Two design choices are important. First, LOTS evaluates the *same frozen output* under both contexts, measuring how tool evidence changes model support without additional rollouts. Second, each tool is evaluated alongside the other tools used in the execution, preserving dependencies in compositional tool use. LOTS does not observe correctness, but measures which tool evidence the model relies on. Empirically, correct and incorrect outputs often yield similar task-level rankings. Appendix E examines when this holds or breaks down. Benchmark-specific interventions and scored tokens are also detailed in Appendix B.2.

3.2 Task-Level Tool-Space Construction

The score $I_x(s)$ measures the contribution of tool s to a single request. To construct a tool space for the recurring task c_t, LOTS aggregates these request-level contributions across the task's output experience. Let $Q_t(s) = \{x \in Q_t : s \text{ is exposed and scoreable for } x\}$ denote the requests that provide evidence for tool s. Its task-level contribution

$$\bar{I}_t(s) = \frac{1}{|Q_t(s)|} \sum_{x \in Q_t(s)} I_x(s). \tag{3}$$

Thus, evidence collected from otherwise independent requests is consolidated into a shared task-level estimate rather than discarded after each interaction. A tool that is not exposed on a request contributes no evidence for that request. Let $A_t \subseteq S$ denote the tools eligible for optimization. LOTS ranks them according to $\bar{I}_t(s)$ and defines the high-value set

$$E_t = \mathrm{TopK}_{s \in A_t} \bar{I}_t(s), \tag{4}$$

where K specifies the tool-space budget, with its protocol described in Appendix C.2. Fix a common menu S with $|S| = m$ and the recorded outputs. Let $F(A)$ denote the mean fixed-output likelihood when only the evidence of $A \subseteq S$ is retained, so that $\bar{I}_t(s) = F(S) - F(S \setminus \{s\})$, and define

$$\widehat{F}(A) = F(S) - \sum_{s \notin A} \bar{I}_t(s).$$

The following result relates top-K selection to maximizing F under bounded tool interactions, a detailed proof is provided in Appendix E.1.

Theorem 1 (Selection under bounded interactions). *Let $r = m - K$. Suppose $|\Delta_{s,u} F(B)| \le \beta$, for every $B \subseteq S$ with $K \le |B| \le m - 2$ and distinct $s, u \notin B$, where $\Delta_{s,u} F(B) = F(B \cup \{s,u\}) - F(B \cup \{s\}) - F(B \cup \{u\}) + F(B)$ is the second-order difference (Sundararajan et al., 2020). Then E_t in equation 4 maximizes $\widehat{F}(A)$ over $|A| = K$, and, for every such A, $|F(A) - \widehat{F}(A)| \le \binom{r}{2}\beta$. Consequently, for $A^\star \in \arg\max_{|A|=K} F(A)$, $0 \le F(A^\star) - F(E_t) \le r(r-1)\beta$. Both bounds are zero for $r \le 1$.*

3.3 Continual Task-Specific Tool-Space Evolution

The constructed space E_t is not a request-local selection. It is stored as persistent deployment state associated with task c_t and reused for all requests of the same task. When the next task arrives, LOTS independently constructs its corresponding space from that task's own output experience. After observing tasks $c_1, \ldots, c_t$, the deployed agent therefore maintains $\mathcal{E}_t = \{E_1, \ldots, E_t\}$, and the continual update is

$$\mathcal{E}_{t-1} \longrightarrow \mathcal{E}_t = \mathcal{E}_{t-1} \cup \{E_t\}. \tag{5}$$

Earlier task spaces remain unchanged when a new task arrives. Consequently, new task knowledge is incorporated by expanding the collection of task-specific tool spaces rather than by modifying p_θ or repeatedly reconstructing a tool configuration for every request. In this sense, the deployed agent self-evolves through its tool-space state:

$$c_t \longrightarrow Q_t \longrightarrow \{I_x(s)\} \longrightarrow \bar{I}_t(s) \longrightarrow E_t.$$

If additional experience from an existing task becomes available, its stored space can be revised using the same scoring and aggregation procedure. Denoting successive versions by $E_t^{(0)}, E_t^{(1)}, \ldots$, a new batch of requests from c_t produces

$$E_t^{(k)} \xrightarrow{\text{new output experience}} E_t^{(k+1)}. \tag{6}$$

This within-task refinement is optional to the continual task-arrival setting: the core mechanism only requires that each incoming task constructs and retains its own persistent space. Later batches of a task are served under its current space, their traces update the running scores, and the space is recomputed from them. Details of the running-score updates are given in Appendix B.3.

Algorithm 1: Task-specific tool-space construction

Input: Task c_t, request batch Q_t, candidate space $E_t^c \subseteq \mathcal{S}$, budget K, fixed model p_θ.

1. Serve. Execute Q_t under the fixed space E_t^c and record traces (h_x, y_x).

2. Score. For each request x and exposed, scoreable tool s, keep y_x fixed and compute $I_x(s) = L_\theta(y_x \mid h_x) - L_\theta(y_x \mid h_x^{-s})$. Here h_x^{-s} removes s's calls and observations; where a later tool consumes s's output, the chain is re-executed without s (Appendix B.2).

3. Aggregate. Let $Q_t(s)$ contain the requests providing scores for s, and $A_t = \{s \in E_t^c : |Q_t(s)| > 0\}$. Compute $\bar{I}_t(s) = \frac{1}{|Q_t(s)|} \sum_{x \in Q_t(s)} I_x(s)$.

4. Prune. Set $E_t = \mathrm{TopK}_{s \in A_t} \bar{I}_t(s)$, retaining all eligible tools if $|A_t| < K$.

5. Deploy. Store E_t for subsequent task-c_t requests; repeat with new evidence when updating.

4 Experiments

We evaluate task performance (§4.2), evolution across and within tasks (§4.3), and serving efficiency as well as cross-model reuse (§4.4). Scoring analyses and ablations are presented in Appendix E.

4.1 Experimental Setup

Following §2, each task is a TGB family, a BFCL API class, or a GTA-Atomic task type. We evaluate the top-K spaces of the LOTS ranking over a predefined grid and keep the smallest K attaining the highest fitting accuracy (Appendix C.2). Likelihood scoring does not use correctness labels,

Table 1: **TGB dataset accuracy (%).** One tool space fitted per task family. Search-based baselines include BM25, Dense, and Tool2Vec; the LLM-based router selects tools per request. † marks methods that read correctness labels on the fitting requests. Bold marks the best result in each column.

	Qwen2.5-7B	Qwen2.5-14B	Llama3.1-8B	Mistral-7B	Qwen3.5-9B	Qwen3-8B	Phi4-14B	Gemma4-12B
No tools	10.8	11.4	12.9	10.0	10.5	10.4	11.0	12.8
Keep all	69.6	88.0	77.7	46.4	50.9	56.5	79.7	86.7
Search-Based								
Random	23.4	19.3	41.3	11.5	27.5	10.1	24.3	20.0
BM25	19.3	36.3	40.0	13.4	24.9	10.0	39.3	37.5
Dense	20.5	36.3	37.6	17.2	18.5	18.5	11.1	38.8
Tool2Vec	38.1	45.3	57.4	24.2	23.9	37.9	55.6	47.5
LLM-Based								
LLM as router	31.7	62.5	39.4	29.3	36.1	44.7	61.7	66.9
Jev	60.3	70.8	68.2	51.5	56.6	49.5	81.7	71.3
Beam Search†	26.6	38.1	19.6	21.2	23.8	36.8	64.1	30.0
TTO†	48.6	88.0	65.4	63.1	63.7	63.0	72.1	86.8
LOTS	**82.9**	**91.8**	**84.8**	**65.8**	**65.2**	**81.5**	**92.1**	**91.4**

whereas only budget selection in these experiments uses fitting labels.

Benchmarks. We evaluate LOTS on *three* benchmarks. **TGB (controlled composition)**, provides a controlled compositional setting for studying tool dependencies and distractors, and contains 4,000 requests across ten families and fifteen tools (seven task tools and eight distractors); Appendix B.1 gives its construction and the family chains (Table 4), and Appendix D.6 follows one TGB task through the whole procedure. Each task uses 80 requests for fitting and 320 for evaluation, enabling controlled studies of tool dependencies, distractors, and space updates. **BFCL-v4 multi-turn (multi-turn calling)** (Patil et al., 2025) evaluates multi-turn function calling with task-specific documentation allocation. Success requires completing the required call sequence correctly; each API class uses 10 records for fitting and 40 for evaluation. **GTA-Atomic (executable ReAct agent)**, the atomic-task split of GTA (Wang et al., 2024b), evaluates ReAct task execution through the official harness on three annotated task types. Table 3 reports accuracy over the 112 evaluation requests of the three types pooled, from one serving session per model. Appendices B.1 and B.2 detail the datasets, tool libraries, and scoring interventions.

Models. We evaluate Qwen2.5 (Qwen Team, 2024) (3B/7B/14B), Qwen3 (Yang et al., 2025) (4B/8B), Qwen3.5 (Qwen Team, 2026) (2B/4B/9B), Llama3.1 (Grattafiori et al., 2024) (8B), Mistral (Jiang et al., 2023) (7B), Phi-4 (Abdin et al., 2024) (14B), and Gemma-4 (Team et al., 2026) (12B) instruction-tuned models, with thinking disabled and greedy decoding to reduce sampling variability. Methods share fitting and evaluation splits within each model and benchmark.

Baselines. *Keep all* serves the full registry and *No tools* an empty menu. Search-Based: *Random* draws K tools; *BM25* (Robertson & Zaragoza, 2009) ranks tool descriptions by lexical similarity to the fitting requests; *Dense* uses BGE description embeddings (Xiao et al., 2024) and *Tool2Vec* (Moon et al., 2024) uses the query embedding generated by the tool. LLM-Based: *LLM as router* (Shen et al., 2023) gives the serving model the request and the name and description of every registry tool, asks it for the list of tools the request needs, and serves exactly that list; *Jev* (TypeSafe, 2026) is an external relevance model that scores every tool of a request with an independent probability from the request and the tool descriptions, and serves the top-K tools. We additional adapt two output-aware methods: *TTO* (Lu et al., 2025) starts from the best single tool on fitting accuracy and adds the tool with the largest gain while accuracy strictly increases. *Beam Search* (Lowerre, 1976) keeps the five best subsets at each size. TTO and Beam Search need ground-truth labels and is costly in loop tool space composition. Appendices B.1 and B.4 give implementation details.

4.2 Effectiveness of LOTS in Tool Space Optimization

Our central claim is that LOTS constructs a persistent task-specific tool space, which improves task performance. We evaluate on three benchmarks that cover synthetic compositional tool use, multi-turn function calling, and ReAct execution with real tools.

TGB: task-specific spaces remove distractors. LOTS improves over the full fifteen-tool menu on all eight models in Table 1. The gains are largest on models most affected by irrelevant tools: accuracy rises from 0.565 to 0.815 on Qwen3-8B and from 0.464 to 0.658 on Mistral-7B. Across task

Table 2: **BFCL v4 `multi_turn` accuracy.** For each task, the first 10 records fit the space and its remaining 40 records evaluate it (160 records in total). † marks methods that read correctness labels on the fitting records. Bold marks the best result in each column; underlining marks the second best. Results are averaged over three seeds.

Menu	Qwen3-4B	Qwen3.5-4B	Qwen3.5-9B	Gemma4-12B	Phi-4-mini-3.8B
No tools	0.00	0.00	0.00	0.00	0.00
Keep all	17.71 ± 0.95	14.37 ± 0.63	23.12 ± 0.63	24.79 ± 0.36	3.33 ± 0.36
Search-Based					
Random	10.21 ± 1.06	2.71 ± 1.06	13.96 ± 0.29	13.54 ± 0.29	3.54 ± 0.29
BM25	18.13 ± 0.51	12.08 ± 0.29	27.08 ± 0.78	25.83 ± 1.93	3.33 ± 0.29
Dense	25.62 ± 1.77	13.96 ± 0.29	27.71 ± 0.29	26.04 ± 0.29	3.75 ± 0.00
Tool2Vec	24.58 ± 0.95	10.42 ± 0.95	23.12 ± 0.36	23.54 ± 1.77	3.54 ± 0.36
LLM-Based					
LLM as router	12.71 ± 1.18	5.83 ± 0.00	9.79 ± 0.29	21.25 ± 0.88	3.54 ± 0.29
Jev	22.50 ± 0.00	8.96 ± 0.36	22.50 ± 0.62	$\mathbf{28.96 \pm 1.57}$	$\mathbf{5.21 \pm 0.36}$
TTO†	20.83 ± 0.36	12.29 ± 1.57	22.79 ± 0.72	24.17 ± 0.36	4.79 ± 0.36
LOTS	$\mathbf{28.12 \pm 0.72}$	$\mathbf{15.62 \pm 0.95}$	$\mathbf{29.38 \pm 0.62}$	28.75 ± 1.25	5.00 ± 0.62

families, LOTS retains between roughly three and seven tools on average from the original 15 tool registry. Additionally, Table 7 (Appendix C.2) shows how each family's budget is chosen and which tools it keeps. Thus, the constructed spaces are the right ones: on Qwen2.5-7B every family's space contains its gold tool chain, and for five of the ten families it is exactly that chain.

BFCL: task-specific spaces benefit multi-turn tool calling. BFCL evaluates success over complete multi-turn records, where an incorrect required call can cause the entire record to fail. Beam search is computationally prohibitive in this setting, so we exclude it from the BFCL experiments. LOTS gives the best space on three of the five models and is within 0.21 points of the best, Jev, on the other two: on Qwen3-4B it reaches 28.12% against 25.62% for Dense, the strongest baseline there, and on Gemma4-12B it reaches 28.75% against 28.96% for Jev. The LLM router is below LOTS on all five models, by 1.5 to 19.6 points, which reflects the difficulty of selecting tools before a multi-turn interaction unfolds. These gains of LOTS therefore reflect improvements in completing multi-turn interactions, beyond isolated tool-call decisions.

GTA-Atomic: task-specific spaces improve later execution by a ReAct agent. We finally evaluate fitted spaces inside the ReAct execution loop of GTA-Atomic, where the agent calls executable tools over several turns. For example, a correct output requires reading text from an image and computing with it. The constructed tool space is OCR together with a computation tool (Calculator or Solver). LOTS improves over Keep all on six of the seven models, with the largest gains on Qwen3-8B, from 10.71 to 22.32, and Qwen2.5-7B, from 7.14 to 14.29, where it ties Beam Search; Qwen2.5-14B is the one model where Keep all stays ahead (8.04 against 7.14). Appendix E.2 enumerates all tool spaces of a six-tool universe to locate the LOTS pair among them.

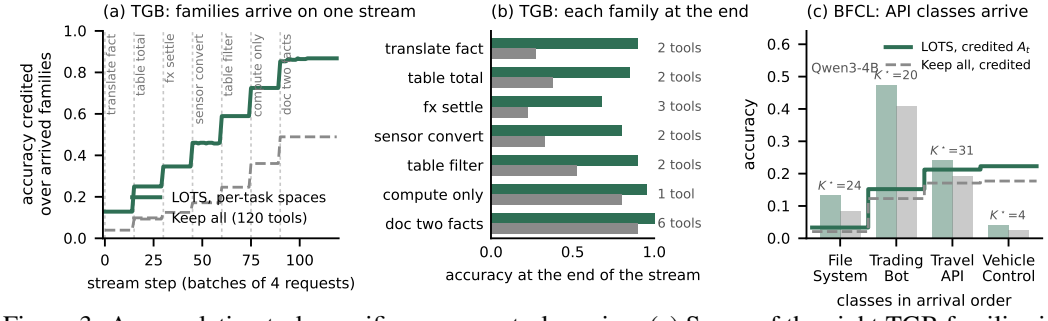

Figure 3: Accumulating task-specific spaces as tasks arrive. (a) Seven of the eight TGB families in this stream arrive on one request stream (Qwen2.5-7B, registry of 120 tools); a step is a batch of four requests, and dotted lines mark arrivals. The curve is credited accuracy Acc_t, it rises in a jump when a family's space is added. (b) Final accuracy and space size for each family. (c) Four BFCL API classes arrive in sequence (Qwen3-4B). Bars give each class's accuracy on its own evaluation records; lines give Acc_t over the four classes, which ends at the mean of the bars.

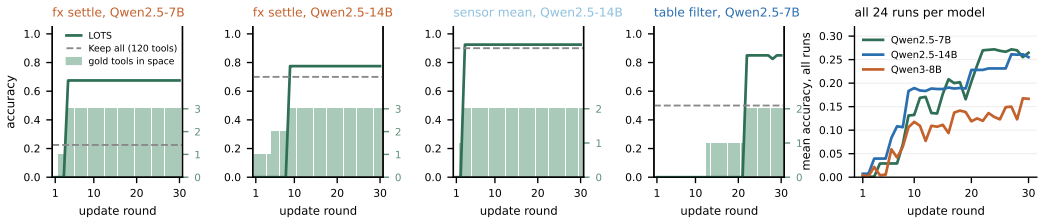

Figure 4: Improving an existing task space through successive request batches. TGB, 120-tool registry. Left: four family updates showing accuracy on the task's fixed evaluation requests. Right: mean accuracy after each update over all 24 runs per model.

Additionally, Appendix E.2 shows why output-based, full-context leave-one-out scoring identifies these spaces while description-based and Shapley-thresholded selection do not.

4.3 Persistent Task-Specific Evolution as Tasks Arrive

Across tasks, we ask whether LOTS can construct and retain useful task-specific spaces as new task types arrive (Eq. 5). Within a task, we ask whether later output experience of the same task can improve its stored space, $E_t^{(k)} \to E_t^{(k+1)}$ (Eq. 6).

Across tasks: constructing and retaining separate spaces. When a new task c_t arrives, LOTS constructs its space E_t from that task's output traces. The deployed agent accumulates a collection of reusable configurations, with each task's experience updating only its own space. Figure 3(a–b) follows seven of the eight families in this stream. At each step, the active family with the fewest completed rounds receives a batch of four requests. After t task types have arrived, we report credited accuracy $A_t = \frac{1}{T}\sum_{j=1}^{t} \mathrm{Acc}(c_j; p_\theta, E_j)$, where T is the total number of task types and each E_j is the corresponding task's current space. This metric measures performance contributed by the tasks encountered so far. LOTS remains above Keep all throughout the stream and finishes at 0.87 versus 0.49. Table 12 in Appendix D reports the full eight-family stream on seven models and three seeds; Figure 3(c) also shows the BFCL setting: each arriving API class fits a documentation configuration on its first records and reuses it on subsequent records. On Qwen3-4B, all four classes improve over Keep all, with final credited accuracy of 22.3% versus 17.7%.

Within a task: improving a space with subsequent experience. We next fix the task type and examine how its space changes across successive requests. We limit initial tool exposure: the registry contains 120 tools, but each request receives at most 16. The initial space may therefore lack tools needed by the task. Figure 4 shows that later experience can improve spaces that are incomplete. The sharp increases occur when an update adds a missing tool and completes a required tool chain, enabling the agent to solve multiple evaluation requests that previously lacked the necessary tools. Across all 24 runs per model, mean accuracy also increases over successive updates. Together, the two studies demonstrate how LOTS accumulates spaces across tasks and updates individual spaces.

Table 3: **GTA-Atomic accuracy (%).** † marks methods that read correctness labels on the fitting requests. Bold denotes the best result in each column; underlining denotes the second best.

Method	Qwen2.5-3B	Qwen2.5-7B	Qwen3-8B	Qwen2.5-14B	Qwen3.5-9B	Phi-4-14B	G4-12B
No tools	1.79	2.68	4.46	4.46	6.25	4.46	2.68
Keep all	7.14	7.14	10.71	**8.04**	40.18	11.61	24.11
Search-Based							
Random	6.25	5.36	11.61	6.25	16.96	8.93	14.29
BM25	1.79	2.68	6.25	3.57	14.29	2.68	14.29
Dense	0.00	1.79	1.79	1.79	12.50	0.89	5.36
Tool2Vec	0.00	0.00	6.25	3.57	16.96	0.00	11.61
LLM-Based							
LLM as router	7.14	1.79	15.18	4.46	25.00	8.04	25.00
Jev	5.36	9.82	12.50	6.25	35.71	8.04	24.11
Beam Search†	7.14	**14.29**	18.75	4.46	33.93	14.29	25.89
TTO†	6.25	12.50	16.96	2.68	33.93	15.18	27.68
LOTS	**13.39**	**14.29**	**22.32**	7.14	**45.54**	**16.07**	**29.46**

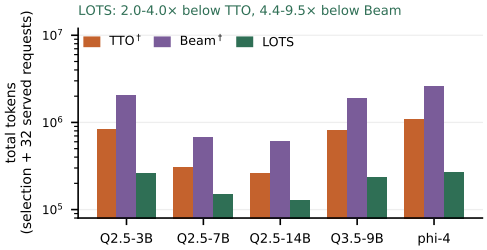

Figure 5: Total token cost on GTA-Atomic `read_arith`: tokens spent on tool selection plus the served evaluation requests, log scale. † methods read correctness labels.

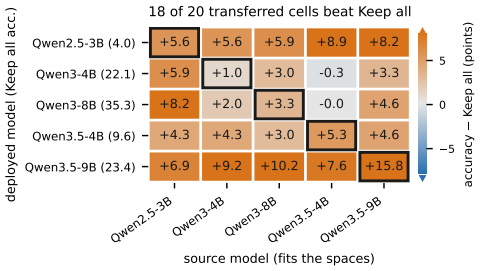

Figure 6: Cross-model tool-space reuse on BFCL. Rows: execution models; columns: sources; cells: accuracy relative to each row's Keep all baseline; outlined cells: own rankings.

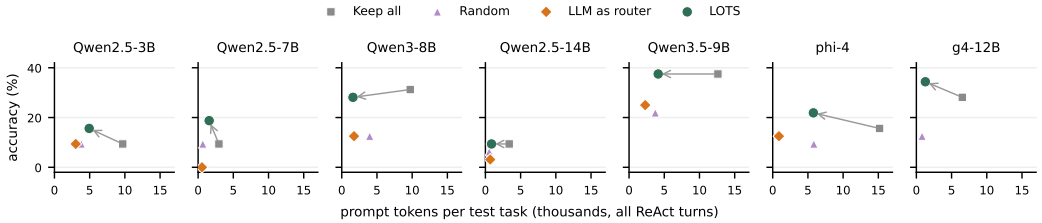

Figure 7: **Accuracy against serving cost on GTA-Atomic `read_arith`.** Prompt tokens per test task over all ReAct turns against accuracy. The arrow runs from Keep all to `LOTS`.

4.4 Serving Efficiency and Cross-Model Reuse

In this section, we examine the resulting token savings and whether a space fitted with one model can be reused by another.

Construction and serving cost. Once constructed, a task-specific space serves subsequent requests without repeating tool search or ranking over the global registry. Figure 5 reports the combined token cost of tool selection and serving the evaluation requests on GTA-Atomic `read_arith`. Under this accounting, `LOTS` uses 2.0-4.0× fewer tokens than TTO and 4.4-9.5× fewer than Beam Search. On TGB, task-specific pruning reduces serving prompt tokens by 33-46% relative to the full fifteen-tool menu (Appendix F). Figure 7 further compares accuracy with serving prompt tokens on GTA-Atomic `read_arith`, showing where smaller spaces improve accuracy and efficiency. Appendix F details the cost and the number of requests needed to recover the construction cost.

Reuse across models. We next test whether changing the serving model requires refitting the tool space. On BFCL, we fit task-specific documentation rankings from a source model's traces and use them with a different serving model. Transferred rankings outperform the serving model's Keep all baseline in 18 of the 20 cross-model pairs in Figure 6. This suggests that the rankings capture task-level preferences that can remain useful across models. Reusing an existing ranking avoids collecting and scoring a new set of traces from the serving model. ince likelihood access is needed only at fitting time, a space fitted with an open-weight model could also serve closed API models, a direct extension of this reuse.

5 Conclusion

We introduced `LOTS`, a likelihood-guided approach to tool-space self-evolution that turns an agent's own execution experience into persistent, task-specific tool spaces. By measuring how observed tool outputs support a fixed realized answer, `LOTS` converts request-level traces into reusable tool configurations that can be refined as new experience arrives. Across compositional tool use, multi-turn function calling, and ReAct execution, these learned spaces consistently reduce unnecessary tool context while improving or preserving task performance. Our results show that an agent's executable environment itself can serve as persistent adaptive state. We provide additional discussion, including related work (App. A), in the Appendix.

AI USE STATEMENT

AI assistants were used to support manuscript editing and revision. The authors reviewed the resulting text and take full responsibility for the content, experimental results, and conclusions of this paper.

ETHICS STATEMENT

This work studies task-specific tool-space optimization for language-model agents using existing benchmarks and synthetic tasks. Our experiments evaluate task performance and tool-use efficiency; they do not establish the safety of autonomous execution. Likelihood-based tool scores measure support for a model's generated answer and do not certify its correctness or the safety of a tool. Practical execution therefore requires appropriate tool permissions and oversight.

REPRODUCIBILITY STATEMENT

The main text and appendices describe the task construction, fitting and evaluation protocols, candidate tool libraries, model configurations, scoring interventions, tool-budget selection, and baseline implementations. The appendices additionally report fixed-budget comparisons, scoring ablations, and settings for repeated updates and cross-model tool-space reuse. We plan to release the implementation, configuration files, and evaluation scripts, together with request identifiers and split definitions where permitted by the source datasets.

REFERENCES

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

Emre Can Acikgoz, Cheng Qian, Jonas Hübotter, Heng Ji, Dilek Hakkani-Tür, and Gokhan Tur. Tool-R0: Self-evolving LLM agents for tool-learning from zero data. *arXiv preprint arXiv:2602.21320*, 2026.

Anthropic. Tool search tool. Claude Platform Documentation, `https://platform.claude.com/docs/en/agents-and-tools/tool-use/tool-search-tool`, 2026. Accessed September 2026.

Tianle Cai, Xuezhi Wang, Tengyu Ma, et al. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*, 2023.

Benjamin Cohen-Wang, Harshay Shah, Kristian Georgiev, and Aleksander Madry. Contextcite: Attributing model generation to context. *arXiv preprint arXiv:2409.00729*, 2024.

Wenlong Deng, Yushu Li, Boying Gong, Yi Ren, Christos Thrampoulidis, and Xiaoxiao Li. On group relative policy optimization collapse in agent search: The lazy likelihood-displacement. *arXiv preprint arXiv:2512.04220*, 2025.

Haipeng Ding, Yuexiang Xie, Zhewei Wei, Yaliang Li, and Bolin Ding. From atomic actions to standard operating procedures: Iterative tool optimization for self-evolving LLM agents. *arXiv preprint arXiv:2607.07321*, 2026.

Yu Du, Fangyun Wei, and Hongyang Zhang. Anytool: Self-reflective, hierarchical agents for large-scale api calls. *arXiv preprint arXiv:2402.04253*, 2024.

Xiang Fei, Xiawu Zheng, and Hao Feng. Mcp-zero: Active tool discovery for autonomous llm agents. *arXiv preprint arXiv:2506.01056*, 2025.

Xinshun Feng, Xinhao Song, Lijun Li, Gongshen Liu, and Jing Shao. SEARL: Joint optimization of policy and tool graph memory for self-evolving agents. *arXiv preprint arXiv:2604.07791*, 2026.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025.

Amirata Ghorbani and James Zou. Data shapley: Equitable valuation of data for machine learning. *arXiv preprint arXiv:1904.02868*, 2019.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Xinyi Hong, Pinjun Dong, Xinyang Yu, et al. Tools are not islands: Set-level tool retrieval for llm agents via query-conditioned hyperedge prediction. *arXiv preprint arXiv:2607.25718*, 2026.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL `https://arxiv.org/abs/2310.06825`.

Mohammad Kachuee, Sarthak Ahuja, Vaibhav Kumar, Puyang Xu, and Xiaohu Liu. Improving tool retrieval by leveraging large language models for query generation. In *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*, pp. 29–38, 2025.

Hui Liu, Bin Zou, Kecheng Chen, Jie Liu, Wenya Wang, and Haoliang Li. Task-aware llm routing with multi-level task-profile-guided data synthesis for cold-start scenarios. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22047–22076, 2026a.

Jiayu Liu, Qihan Lin, Cheng Qian, et al. Planbench-xl: Evaluating long-horizon planning of llm tool-use agents in large-scale tool ecosystems. *arXiv preprint arXiv:2606.22388*, 2026b.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, et al. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023.

Bruce T. Lowerre. *The Harpy Speech Recognition System*. PhD thesis, Carnegie Mellon University, 1976.

Pan Lu, Bowen Chen, Sheng Liu, Rahul Thapa, Joseph Boen, and James Zou. Octotools: An agentic framework with extensible tools for complex reasoning. *arXiv preprint arXiv:2502.11271*, 2025.

Suhong Moon, Siddharth Jha, Lutfi Eren Erdogan, et al. Efficient and scalable estimation of tool representations in vector space. *arXiv preprint arXiv:2409.02141*, 2024.

OpenAI. Codex tool search: deferred tools behind BM25 retrieval. openai/codex repository, `codex-rs/core/src/tools/handlers/tool_search.rs` and `templates/search_tool/tool_description.md`, 2026. Accessed September 2026.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, 2025.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024.

Changle Qu, Sunhao Dai, Xiaochi Wei, et al. Towards completeness-oriented tool retrieval for large language models. *arXiv preprint arXiv:2405.16089*, 2024a. CIKM 2024.

Changle Qu, Sunhao Dai, Xiaochi Wei, et al. From exploration to mastery: Enabling llms to master tools via self-driven interactions. *arXiv preprint arXiv:2410.08197*, 2024b. ICLR 2025.

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

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL `https://qwen.ai/blog?id=qwen3.5`.

Zhe Ren et al. Self-improvements in modern agentic systems: A survey. *arXiv preprint arXiv:2607.13104*, 2026.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

Lloyd S. Shapley. A value for n-person games. In *Contributions to the Theory of Games II*. Princeton University Press, 1953.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI tasks with ChatGPT and its friends in Hugging Face. In *Advances in Neural Information Processing Systems*, 2023.

Zhengliang Shi, Yuhan Wang, Lingyong Yan, et al. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. *arXiv preprint arXiv:2503.01763*, 2025. ACL Findings 2025.

Mukund Sundararajan, Kedar Dhamdhere, and Ashish Agarwal. The shapley taylor interaction index. In *International conference on machine learning*, pp. 9259–9268. PMLR, 2020.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025.

Zhi Rui Tam, Chieh-Yen Lin, Yun-Nung Chen, Shao-Hua Sun, and Hung-yi Lee. Joint optimization of tool creation and use for large language model agents. *arXiv preprint arXiv:2608.24571*, 2026.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, et al. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

TypeSafe. TypeSafe system one API reference and model documentation. `https://docs.typesafe.ai`, 2026. Accessed 2026-09-21; model `jev-1.13.0`.

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, 2024a.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, et al. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Jize Wang, Zerun Ma, Yining Li, Songyang Zhang, Cailian Chen, Kai Chen, and Xinyi Le. GTA: A benchmark for general tool agents. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2024b.

Qin Wang, Olga Fink, Luc Van Gool, and Dengxin Dai. Continual test-time domain adaptation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Xiaohan Wang, Dian Li, Yilin Zhao, Sinbadliu, and Hui Wang. MetaTool: Facilitating large language models to master tools with meta-task augmentation. *arXiv preprint arXiv:2407.12871*, 2024c.

Zhiruo Wang, Daniel Fried, and Graham Neubig. Trove: Inducing verifiable and efficient toolboxes for solving programmatic tasks. *arXiv preprint arXiv:2401.12869*, 2024d.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-Pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024. URL https://arxiv.org/abs/2309.07597.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shuo Yang, Soyeon Caren Han, Xueqi Ma, Yan Li, Mohammad Reza Ghasemi Madani, and Eduard Hovy. EvoTool: Self-evolving tool-use policy optimization in LLM agents via blame-aware mutation and diversity-aware selection. *arXiv preprint arXiv:2603.04900*, 2026a.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026b.

Siyu Yuan, Kaitao Song, Jiangjie Chen, et al. Easytool: Enhancing llm-based agents with concise tool instruction. *arXiv preprint arXiv:2401.06201*, 2024. NAACL 2025.

Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, et al. Agentevolver: Towards efficient self-evolving agent system. *arXiv preprint arXiv:2511.10395*, 2025.

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)*, 2026a. arXiv:2510.04618.

Zheyuan Zhang, Kaiwen Shi, Zhengqing Yuan, Zehong Wang, Tianyi Ma, Keerthiram Murugesan, Vincent Galassi, Chuxu Zhang, and Yanfang Ye. Agentrouter: A knowledge-graph-guided llm router for collaborative multi-agent question answering. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 788–809, 2026b.

Jiuzhou Zhao, Chunrong Chen, Chenqi Qiao, Lebin Zheng, Minqi Han, Yanchi Liu, Yongzhou Xu, Xiaochuan Xu, and Min Zhang. Tcandon-router: Adaptive reasoning router for multi-agent collaboration. *arXiv preprint arXiv:2601.04544*, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Yuanhang Zheng, Peng Li, Wei Liu, et al. Toolrerank: Adaptive and hierarchy-aware reranking for tool retrieval. *arXiv preprint arXiv:2403.06551*, 2024.

Jingqi Zhou, Sheng Wang, Dezhao Deng, Junwen Lu, Junwei Su, Qintong Li, Jiahui Gao, Hao Wu, Jiyue Jiang, Lingpeng Kong, et al. Toolself: Unifying task execution and self-reconfiguration via tool-driven emergent adaptation. *arXiv preprint arXiv:2602.07883*, 2026.

Jiaru Zou, Ling Yang, Yunzhe Qi, Sirui Chen, Mengting Ai, Ke Shen, Jingrui He, and Mengdi Wang. AutoTool: Dynamic tool selection and integration for agentic reasoning. *arXiv preprint arXiv:2512.13278*, 2025.

TABLE OF CONTENTS

A Related Work **15**

B Implementation and Experimental Setup **16**

B.1 Benchmarks and Task Definitions 16

B.2 Computing Tool Scores 18

B.3 Constructing and Updating Tool Spaces 19

B.4 Scoring Details 20

C Additional Task-Performance and Transfer Results **20**

C.1 Cross-Model Reuse on BFCL 20

C.2 Tool Budgets and Documentation 20

C.3 Ranking Quality at a Fixed Budget 22

D Tool-Space Construction and Subsequent Updates **23**

D.1 Task Routing by Request Clustering 23

D.2 Visualization of Task Space 24

D.3 Sizes of Task Space 26

D.4 Constructing Spaces as New Task Types Arrive 26

D.5 Do Further Updates Improve an Existing Space? 27

D.6 Worked Examples of Tool-Space Optimization 28

E Analysis of Tool Scoring **32**

E.1 Top-K Selection as Surrogate Maximization 32

E.2 Scoring Targets and Attribution 33

E.3 Deletion Operation in the Leave-One-Out Score 35

E.4 From Likelihood Scores to Execution Utility 36

E.5 Scoring Models, Aggregation, and Incorrect Outputs 36

E.6 Sensitivity to the Number of Traces 37

F Computational Cost and Registry Scaling **37**

F.1 Scaling with Registry Size 39

A Related Work

Tool selection for LLM agents. Tool-augmented language models learn when and what to call from self-supervised signal (Schick et al., 2023), and scale to large API inventories with a per-request retriever trained on labeled query–API pairs (Patil et al., 2023; Qin et al., 2024); later retrievers target complete tool sets (Qu et al., 2024a; Hong et al., 2026), hierarchical libraries (Zheng et al., 2024; Du et al., 2024), usage-derived embeddings (Moon et al., 2024) and iterative requests (Fei et al., 2025), and a parallel line rewrites documentation once before execution (Yuan et al., 2024; Qu et al., 2024b). Frontier harnesses select in two stages, a coarse hand-picked split into a direct set and a deferred pool, then per-request BM25 or regex ranking over the deferred pool (OpenAI, 2026; Anthropic, 2026); replacing annotated sets with retrieved ones lowers agent pass rates, strong retrievers still miss target tools (Shi et al., 2025; Liu et al., 2026b), and lexical search misses tools whose names overlap. Selection in these systems is a serving-time decision for one request over a library kept fixed by hand; where a set is chosen offline, as in the task-specific toolset optimization of Lu et al. (2025), the search reads graded rollouts. AutoTool trains the policy to select tools from an evolving inventory during reasoning (Zou et al., 2025), and ToolSelf lets the agent reconfigure its own toolbox, strategy and context at run time through tool calls (Zhou et al., 2026); both learn a selection or configuration policy from trajectories, whereas `LOTS` keeps the model fixed and stores the decision as a ranking over the existing registry. MetaTool trains transferable tool-use ability into the model through meta-task augmentation (Wang et al., 2024c); the ranking `LOTS` stores moves between models with no training, which is what the cross-model reuse of §4.4 tests. `LOTS` decides after the tools have run, from the trace itself, and keeps the decision as state that later requests share; its comparison against self-report, documentation-based and gradient-based selection is in §4.2 and Appendix E.2. A coarse per-request retriever and a persistent task-specific configuration answer different questions, which candidates a request may see and which tools the task has earned, and could be composed (§5). The premise that a smaller menu can beat the full one is consistent with models using long contexts unevenly (Liu et al., 2023).

Attribution and valuation. Context attribution asks which parts of a context a model's output relies on; ContextCite ablates context segments and fits a surrogate to the change in the output's likelihood (Cohen-Wang et al., 2024). `LOTS` uses the same kind of ablation signal on tool observations and differs in what it does with it: it aggregates the per-request attributions over the requests of a task and stores the result as a persistent tool space that later requests are served with. Asking what a component is worth by removing it has two standard answers: the Shapley value, which averages a component's marginal contribution over every coalition of the others (Shapley, 1953), and leave-one-out against the full set; data valuation uses both to price training examples (Ghorbani & Zou, 2019), choosing on grounds of cost. Here the choice is made on grounds of what the tasks require. On chained tasks a tool has no value apart from its partners, so a coalition-averaged value paired with an independent threshold reports a small marginal for a load-bearing tool, and a standalone score has nothing to rank; Appendix E.2 measures both with exact Shapley values and finds the highly aligned signal, with aggregation explaining the difference. `LOTS` removes one tool at a time from the full coalition, so each tool is valued with its partners present; and because it scores the likelihood of the agent's own fixed answer rather than its correctness, the resulting ranking requires no labels. Labels are used only to choose the scalar budget K. Gradient attribution over the tool observations is cheaper, one backward pass, and is compared there too; attention-based readings are not. Trajectory-grounded credit assignment also drives EvoTool, which localizes a failure to one module of a tool-use policy and mutates that module through natural-language critique (Yang et al., 2026a); `LOTS` assigns credit to tools instead, by an intervention on the likelihood of a fixed answer under a fixed model, and needs neither a failure signal nor a mutation step.

Self-evolving agents. The survey Gao et al. (2025) defines a self-evolving agent as one that modifies its parameters, context, toolset or architecture from its own trajectories or feedback, and it already covers tools, from tool creation and refinement to tool management and selection; Ren et al. (2026) organise the same field by model versus scaffold. Concrete systems differ in the object experience updates. Dynamic Cheatsheet (Suzgun et al., 2025) keeps a curated memory of strategies and snippets: each query is answered with the memory built from the queries before it, the memory is updated after the answer. ACE (Zhang et al., 2026a) keeps a context playbook of bullets that a generator, reflector and curator grow by incremental deltas; in its online setting each test sample is predicted under the current context and the context is then updated from that sample, with a batch size of one. SkillOpt trains a natural-language skill document as the external state of a frozen agent

and accepts an edit only when it improves a held-out score (Yang et al., 2026b). These systems store experience as text the model reads; LOTS stores it as the executable tool space the model is served. Voyager (Wang et al., 2023) keeps an ever-growing library of executable skills retrieved for later tasks; TroVE (Wang et al., 2024d) grows a toolbox of reusable functions while solving a stream of tasks and periodically trims it; LATM has a tool-maker model write tools that a tool-user model applies, with a cache that stores the functionality of a class of requests for reuse (Cai et al., 2023). EvoSOP synthesizes standard operating procedures from trajectories into higher-order tools and iterates their construction, merging, evaluation and pruning (Ding et al., 2026), and SMITH trains one policy to both create tools and use them (Tam et al., 2026); these methods grow the tool pool itself, whereas LOTS ranks and prunes a fixed registry and could rank the tools such methods create. SEARL jointly optimizes the policy and a tool-graph memory with trajectory- and step-level credit (Feng et al., 2026), and Tool-R0 co-evolves a task generator and a solver by self-play reinforcement learning from zero data (Acikgoz et al., 2026); both change the model's weights under a reward, whereas LOTS keeps p_θ fixed and needs no reward, so a space fitted by LOTS can also serve a policy trained this way. Adapting under a changing request distribution and the forgetting it induces are studied for model parameters in continual test-time adaptation (Wang et al., 2022), and continual-learning protocols for tool use (Wang et al., 2024a). However, none of these methods builds output-aware task-specific tool spaces that evolve from experience traces.

B Implementation and Experimental Setup

LOTS constructs a persistent tool space for each recurring task from the traces of requests served for that task. We first describe the benchmarks and fitting protocols, then give the scoring and update implementations used in the experiments. Throughout, p_θ is the frozen model, x is a request, y_x is its recorded output, and h_x is the context used to score that output. The global tool registry (space) is $\mathcal{S}$, and $E_t \subseteq \mathcal{S}$ denotes the tool subset maintained for task c_t. Figure 2 in the main text summarizes the procedure.

B.1 Benchmarks and Task Definitions

TGB. TGB is a synthetic, text-only benchmark for compositional tool use, with 4,000 requests across ten families. Each request is generated from a hidden scene (a table, sensor series, calendar, or small document corpus) that only the tools can access. The question requires the family's tool chain, and the gold answer is computed from the scene. Upstream tools return multiple records, such as six table rows, five readings, or two passages, so the downstream computation requires combining their outputs. The `compute_only` and `no_tool` families provide tool-optional and tool-free controls (Table 4). The registry contains seven chain tools and eight distractors: UnitConvert, TempConvert, CurrencyConvert, DurationCalc, Solver, GoogleSearch, Summarize, and Barcode. Distractors are executable; on an unsuitable input, each applies its own transformation and returns a result in its usual format. Larger registries add synthetic distractors with seeded names such as `TaxCalc` and `PayrollForecast`, each returning a plausible number derived from the scene. Accuracy measures whether the final answer matches the gold value.

Table 4: TGB families and their tool chains.

Family	Chain	Required computation
table_total	TableQuery → Calculator	sum of six quantity×price products
table_filter	TableQuery → Calculator	the same sum over a filtered subset
sensor_mean	SensorAPI → Calculator	mean of five readings
sensor_convert	SensorAPI → Calculator	mean, then a unit conversion
schedule_gap	CalendarAPI → Calculator	minutes between two clock times
doc_two_facts	DocRetrieve → Calculator	two retrieved prices combined
translate_fact	DocRetrieve → Translate	a figure written in cipher words
fx_settle	DocRetrieve + ExchangeRate → Calculator	price × quantity × rate
compute_only	Calculator (optional)	tool-optional control
no_tool	none	tool-free control

BFCL. The primary BFCL evaluation uses the 200 records of BFCL v4 `multi_turn_base` (Patil et al., 2025), grouped by primary API class: GorillaFileSystem, TradingBot, TravelAPI, and VehicleControlAPI, with 50 records per class. A record succeeds only when the required calls across all turns reach the reference end state. We therefore use documentation demotion rather than tool removal: all tools remain callable, including those needed only in a later turn. The cross-model and budget diagnostics use a separate 101-record tune set, with fitting and evaluation on the same records.

GTA-Atomic. GTA-Atomic, the atomic-task split of GTA (Wang et al., 2024b), contains 229 image-based requests served by a ReAct agent through the official harness and an executable tool server. Four annotated numeric task types provide exactly scoreable requests: `read_arith` (65), `web_fact` (35), `count_arith` (30), and `math_eq` (17). The main experiments (Table 3) use the three types `read_arith`, `count_arith` and `web_fact`; `math_eq` is not used. The Keep-all menu lists the 14 tools of the tool server, and the schema-token counts are taken over these 14; two of them, GoogleSearch and MathOCR, require external credentials and are marked unavailable, so a call to them fails. Fitting uses the seven candidates whose outputs can be cached for each request: OCR, ImageDescription, Calculator, CountGivenObject, RegionAttributeDescription, TextToBbox, and Solver. Accuracy is exact match between the final answer and the reference. Task assignments come from benchmark metadata: TGB families, BFCL API classes in the primary experiments, and annotated GTA-Atomic types.

Baselines. Unless otherwise stated, task-level baselines construct one tool space per task using its fitting requests and reuse that space during evaluation. The two groups in Tables 1–3 are separated by what selection consults: the search-based group ranks tools by matching requests to tool descriptions, whereas the LLM-based group runs the serving model during selection. In the second group, the LLM router and Jev select per request; TTO and Beam Search execute candidate spaces on the fitting requests, select once per task, and reuse the selected space for evaluation. Under pruning, the selected tools form the callable menu. Under documentation demotion, all tools remain callable, but only the selected tools retain their full documentation. Budget-matched baselines use the same task-specific budget K as `LOTS`. Gold answers or gold tool annotations are used only by the explicitly labeled references and accuracy-based search methods described below; evaluation labels are not used for selection.

- **Keep all.** Exposes the full benchmark-specific tool registry with complete documentation.
- **No tools.** Exposes an empty callable tool set.
- **Random.** Selects a uniformly sampled tool subset with the same size as the `LOTS` space. The subset is reused across evaluation requests of the same task. Experiments using a fixed budget or repeated random draws specify these settings separately.
- **Jev.** An external relevance model (TypeSafe System One, `jev-1.13.0`) reads each evaluation request and the tool descriptions and answers one yes/no question per tool with an independent probability; the K highest-scoring tools serve that request. It selects per request, as the LLM router does, and reads no trace. Jev is a commercial closed model accessed through an API; we pin the model version, cache every call by the hash of its input, and release the per-tool scores and the resulting spaces, so the evaluation can be rerun without API access. It receives the same request text and tool descriptions as the other description-based baselines.
- **Oracle.** Selects tools using gold tool annotations. This baseline serves as a labeled reference rather than a deployable selection method; its performance is not assumed to be an upper bound.
- **BM25.** Scores tools by lexical matching between requests and tool names, descriptions, and schemas, without observing tool outputs. Experiments using per-request retrieval are identified separately.
- **Dense.** Uses BGE embeddings to compare request text with tool descriptions. Tools are ranked by their embedding similarity to the task's fitting requests, and the top-K tools are retained. This baseline does not observe execution traces or tool outputs.
- **LLM router.** Prompts the serving model to select tools from their descriptions once per request, before observing tool outputs. The selected menu is used for that request. Unlike task-level baselines, the router repeats selection for each evaluation request.
- **Tool2Vec** (Moon et al., 2024). Our TGB implementation uses Qwen2.5-7B-Instruct to generate 20 queries per tool from its name and description, embeds them with e5-base-v2, and averages

their embeddings into a tool vector. A separate request-level variant with a fixed four-tool budget reaches 0.359 accuracy on Qwen2.5-7B.

- **Trace judge.** Gives an LLM judge the executed trace and generated answer and asks which tool outputs the answer depends on. The resulting judgments are aggregated over fitting requests to rank tools, and the highest-ranked tools are retained at the comparison budget. This baseline observes tool outputs, like `LOTS`, but attributes their contribution through explicit LLM judgments rather than likelihood differences. Gold answers are not provided to the judge.
- **TTO.** Searches for a task-specific tool subset using fitting-set accuracy as the selection objective. Candidate spaces are evaluated through additional execution on fitting requests, and the selected space is reused during evaluation. This is a label-using baseline: it requires reference answers to evaluate candidate spaces.
- **Beam Search.** Searches over tool subsets while retaining multiple candidate spaces at each search step. Candidates are ranked by fitting-set accuracy, and the selected space is reused during evaluation. Like TTO, this baseline uses fitting labels and additional execution to compare candidate spaces. The search starts from the empty space and stops when the best candidate of the next size does not strictly improve fitting accuracy. On TGB chain families every single tool scores zero on the fitting requests, because the answer needs two or three tools together, so no one-tool space improves on the empty space and the search stops there.

The labeled search baselines evaluate candidate menus on fitting accuracy. *TTO* adds the tool giving the largest improvement while accuracy strictly increases; *beam search* retains five candidate sets per expansion. These methods are marked †. The TGB inclusion comparator evaluates each one-tool addition to an empty menu and retains tools with a positive accuracy change. We use "label-free" only for operations with no correctness-label input: this includes own-output scoring.

B.2 Computing Tool Scores

For a recorded output y_x, `LOTS` compares its likelihood under the original scoring context h_x and a context h_x^{-s} in which tool s's evidence is removed:

$$I_x(s) = \mathcal{L}_\theta(y_x \mid h_x) - \mathcal{L}_\theta(y_x \mid h_x^{-s}), \qquad \mathcal{L}_\theta(y \mid h) = \frac{1}{|y|} \sum_{j=1}^{|y|} \log p_\theta(y_j \mid h, y_{<j}). \tag{7}$$

Both terms teacher-force the same output tokens; the model does not generate a replacement answer. The scored tokens are located within the full tokenized sequence, since separately tokenizing the answer can change its boundary. Prompts use the model's chat template with thinking disabled. Tool documentation remains unchanged during scoring. The intervention on the recorded evidence depends on the benchmark's execution format.

On **TGB**, the context lists one line per served tool in a fixed canonical order, formatted as `<Tool>: <output>` and truncated to 1,500 characters per line. Each tool runs once per request. The family's chain executes in dependency order, and other served tools run with generic arguments. Execution is deterministic given the request and menu. Removing s re-executes the chain without it: its line disappears, and a downstream tool that requires its output receives an unresolved argument and returns an error such as `Calculator: NameError`. Outputs unrelated to s remain unchanged. An upstream tool's score therefore includes the effects propagated through its downstream dependencies. The whole frozen answer is scored.

On **BFCL**, each record is scored turn by turn from its inference log. The scoring sequence contains the system prompt and query, with the original menu, followed by the turn's recorded steps. The last assistant message of the turn is the frozen target; it is never removed, including when it is itself a call to the candidate tool, so both contexts score the same string. An intervention removes, from the preceding steps, the assistant messages calling the candidate tool and their observations. If an assistant message calls several tools, the implementation removes that whole message and all observations of the step. Scores therefore reflect the removal of the recorded step in these cases. In the fitting traces of the five BFCL models, 24.9% of tool-calling assistant messages call two or more distinct tools (5.7% for Phi-4-mini to 30.9% for Qwen3-4B), and 44.1% of the (record, called tool) pairs involve at least one such message (12.3% to 54.0%), so up to that share of BFCL scores also carries the contribution of a co-called tool. Repeated calls to the candidate tool are

removed together. Turn-level differences are averaged within a record; a record with no scorable turn contributes nothing to the aggregate.

On **GTA-Atomic**, candidate outputs are computed independently, cached once per request, and truncated to 400 characters. An intervention deletes only the candidate's line. The four-type fits score the first number-like expression in the model's final full-menu message, or its last line if no number is present; only empty final messages are excluded. For `read_arith`, this yields 13 scored fitting requests for four models, 12 for two models, and 7 for phi-4. The primary 33/32 fit scores the same string. The answer boundary is located using the longest common token prefix of the context alone and the context followed by the answer. The separate 65-request diagnostic scores numeric, percentage, and currency expressions, excluding outputs without such expressions.

The score measures dependence of the recorded output on the intervened evidence, rather than correctness. For fixed contexts, the unnormalized sequence log-ratio $J(y) = \log p_\theta(y \mid h) - \log p_\theta(y \mid h^{-s})$ obeys

$$\mathbb{E}_{y\sim p_\theta(\cdot|h)}[J(y)] = \mathrm{KL}\big(p_\theta(\cdot \mid h)\big\|p_\theta(\cdot \mid h^{-s})\big)\,. \tag{8}$$

This identity applies to model-sampled sequences and unnormalized log probabilities. `LOTS` evaluates a length-normalized difference on a greedily generated output; it is not an unbiased estimate of this KL divergence and need not be nonnegative. Appendix E.5 examines how incorrect outputs affect the resulting space.

B.3 Constructing and Updating Tool Spaces

From request scores to a task ranking. A request contributes a score $I_x(s)$ only for the tools that were exposed on it and could be scored. The task score $\bar{I}_t(s)$ is the mean of these scores over $Q_t(s)$, the task's requests that provide evidence for s. A request on which s was not exposed is left out of this mean; it is not counted as a zero. Tools are ranked by $\bar{I}_t(s)$, with ties broken by tool name so that the ranking is deterministic. The budget K is chosen once per task on its fitting requests: the top-K space is served for every candidate value, and the smallest value with the highest fitting accuracy is kept. On TGB the candidates are $K = 1, \ldots, 15$, so the full menu is one of them. Appendix C.2 lists the chosen budgets.

Spaces smaller than the budget. Only tools with evidence enter the ranking. When fewer than K tools have evidence, the space contains exactly those tools, $|E_t| = \min(K, |A_t|)$. The remaining slots stay empty: a tool that was never scored is never added to reach the budget. This is why the reported number of tools kept can be below K, most visibly in the first batches of a task served under a serving cap.

Updating a space across batches. Each task keeps a running score $\tilde{I}_t(s)$ for every tool that has been scored at least once. After a batch, the batch mean $\bar{I}_t^{(k)}(s)$ updates the running score by an exponential moving average,

$$\tilde{I}_t(s) \leftarrow \alpha\, \bar{I}_t^{(k)}(s) + (1-\alpha)\, \tilde{I}_t(s), \tag{9}$$

and the first batch that scores a tool initializes $\tilde{I}_t(s)$ to its batch mean. A tool that the batch did not expose keeps its running score unchanged; the score does not decay. The next space is the top-K of the running scores over all tools scored so far, so a tool that left the space can return without a change of rule. The first batch of a task is served under the registry, and later batches under the current space; the running scores of the tools in the space are updated from their traces. When the registry cannot be served in full (a serving cap C, Appendix D.5), the first batch is served under a random sample of C registry tools and each later request is explored with probability ε: it is served under the current space together with a random sample of other registry tools, up to C in total, which is the only way a tool outside the space obtains evidence. The task-arrival stream of Figure 3 uses batches of four requests, $\varepsilon = 0.25$, and $\alpha = 0.3$. The cold-start runs of Figure 4 use a cap of 16 tools, $\varepsilon = 0.5$, and $\alpha = 0.3$.

Stored state. The persistent state of a task is its running scores and the resulting list of tool names; no model parameter or prompt is stored. A deployment that has served t tasks therefore holds t such lists, and serving a request only reads the list of its task. Task assignments come from benchmark metadata in our experiments (Appendix B.1); without metadata, requests are clustered into tasks, and a request that matches no existing task opens a new space.

B.4 Scoring Details

Accuracy is the fraction of evaluation requests answered correctly, with BFCL scored over complete records. Tools kept counts callable tools under pruning and fully documented tools under demotion. Prompt tokens include the history and documentation processed at every agent turn; BFCL sums turns within each record. Schema tokens count only the tool-definition block. Selection cost and break-even counts are defined in Appendix F. The statistical unit is a request or record. Repeated greedy runs measure operational variability, not additional independent samples. Stream bands show ranges across the reported seeds. A 40-request evaluation set changes by 2.5 percentage points per answer; repeated serving of an unchanged space sometimes changes one answer.

C Additional Task-Performance and Transfer Results

C.1 Cross-Model Reuse on BFCL

Table 5 gives the complete transfer matrix behind Figure 6: ten execution models, eleven source models, $K = 16$ fully documented tools, three seeds.

Table 5: Cross-model configuration reuse on BFCL, $K = 16$. Rows are execution models; columns are source models. Underlining marks the own-model configuration and bold the best configuration in each row.

Deploy ↓	Full	2.5-3B	2.5-7B	2.5-14B	3-4B	3-8B	3.5-2B	3.5-4B	3.5-9B	Phi-m	g-12B	g-31B
Qwen2.5-3B	4.0	9.6	11.9	11.9	9.6	9.9	7.6	12.9	12.2	11.2	**14.8**	8.6
Qwen2.5-7B	16.2	**18.5**	15.5	**18.5**	16.5	17.5	**18.5**	17.8	14.8	13.9	17.8	18.1
Qwen2.5-14B	18.8	18.8	19.8	**20.5**	15.5	14.5	17.2	15.2	13.9	17.5	18.8	14.8
Qwen3-4B	22.1	28.1	27.1	25.7	23.1	25.1	21.4	21.8	25.4	25.7	**30.0**	23.1
Qwen3-8B	35.3	43.6	**44.2**	**44.2**	37.3	38.6	37.0	35.3	39.9	36.3	40.9	38.0
Qwen3.5-2B	5.0	0.0	1.0	1.3	0.0	0.0	**1.6**	0.0	0.0	1.3	0.0	1.0
Qwen3.5-4B	9.6	13.9	14.8	9.2	13.9	12.5	11.9	14.8	14.2	**16.2**	15.8	12.5
Qwen3.5-9B	23.4	30.4	33.3	31.0	32.7	33.7	31.4	31.0	**39.3**	28.1	35.6	32.3
Phi-4-mini	5.0	7.9	5.6	5.0	7.3	5.9	5.6	5.0	5.9	4.3	**8.6**	6.6
gemma-4-12B	30.0	**37.0**	36.6	31.7	27.4	29.7	32.3	30.0	31.7	30.7	25.4	30.0

Reuse under the primary-class protocol. Table 6 repeats the test under the primary-class protocol of Section 4.1: each source ranks a class's tools from its own 10 fitting records, each execution model documents the top $K = 16$, and accuracy is read on the 160 evaluation records over three seeds. Eight of the nine source–execution pairs improve on Keep all by 1.3 to 6.1 points and the ninth matches it.

Table 6: Cross-model reuse on BFCL with independent fitting and evaluation records: accuracy (%) on the 160 evaluation records at a fixed $K = 16$, mean over three seeds (Gemma4-12B as execution model: two seeds). Rows are execution models; columns are the source of the ranking.

execution model	from Qwen3-4B	from Qwen3.5-9B	from Gemma4-12B	Keep all
Qwen3-4B	**21.5**	18.1	19.4	18.1
Qwen3.5-9B	27.7	26.5	**29.4**	23.3
Gemma4-12B	26.9	**29.1**	27.5	25.0

C.2 Tool Budgets and Documentation

Table 7 lists the fitting accuracies that select the budget K on Qwen2.5-7B (selected columns of the $K = 1, \ldots, 15$ sweep). Compact chains prefer compact spaces (`fx_settle`: 0.72 at $K = 3$, 0.16 at $K = 15$), while both sensor families select large budgets.

Table 7: TGB budget selection on Qwen2.5-7B. Entries are accuracies on each family's 80 fitting requests. The sweep runs over $K = 1, \ldots, 15$ and the table shows selected columns with the chosen K; at most four retained tools are listed.

Family	$K=1$	$K=2$	$K=3$	$K=4$	$K=6$	$K=8$	$K=12$	$K=15$	Selected K	Tools at selected K
compute only	0.94	0.85	0.82	0.80	0.80	0.82	0.81	0.78	1	Calculator
doc two facts	0.01	0.99	0.99	0.97	0.97	0.96	0.96	0.99	2	DocRetrieve, Calculator
fx settle	0.01	0.00	0.72	0.53	0.23	0.15	0.10	0.16	3	DocRetrieve, ExchangeRate, Calculator
no tool	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	1	Calculator
schedule gap	0.00	0.23	0.28	0.45	0.64	0.69	0.56	0.59	9	CalendarAPI, Calculator, GoogleSearch, Barcode …
sensor convert	0.00	0.80	0.61	0.49	0.65	0.65	0.76	0.79	14	SensorAPI, Calculator, GoogleSearch, UnitConvert …
sensor mean	0.00	0.34	0.36	0.69	0.54	0.65	0.55	0.56	14	SensorAPI, Calculator, Barcode, UnitConvert …
table filter	0.00	0.88	0.75	0.80	0.79	0.76	0.84	0.84	2	TableQuery, Calculator
table total	0.00	0.85	0.74	0.59	0.69	0.69	0.60	0.60	2	TableQuery, Calculator
translate fact	0.00	0.89	0.90	0.88	0.82	0.82	0.88	0.88	3	DocRetrieve, Translate, UnitConvert

Figure 8 varies the documentation budget on the 101 BFCL tune records. The best budget depends on the source and execution model: Qwen3.5-9B reaches 39.3% at $K = 16$ and 26.1% at $K = 31$, Qwen2.5-7B 15.8% and 19.8%.

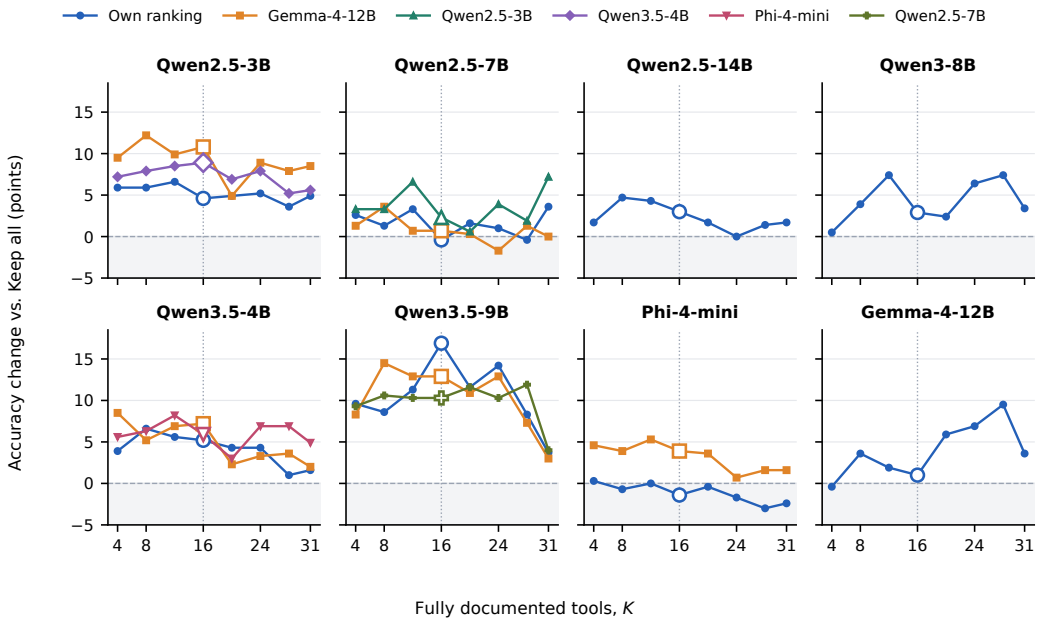

Figure 8: BFCL documentation-budget sensitivity on the 101 tune records: accuracy change over Keep all, in percentage points, as the number of fully documented tools K varies. Each panel is one deployed model; colors give the source of the ranking. Hollow markers mark $K = 16$; at $K = 31$ all documentation is kept and only the menu order changes. Results average three seeds, two for Qwen3-8B.

Table 8 compares pruning with documentation demotion on GTA-Atomic. On Qwen2.5-7B, demotion reaches 18.75% on the 32-request split against 12.50% for pruning; pruning cuts the schema from 2,147 to 186 tokens and raises Qwen2.5-7B's total tool calls from 117 to 173.

Table 8: GTA-Atomic pruning and documentation variants, with one serving session per model. Accuracy (%) is reported on the 32-request `read_arith` split and all 229 requests. Demotion keeps descriptions only for OCR and Calculator. Schema tokens count one tool-definition block; calls are totals over 229 requests.

Menu	Tools	Schema tokens	32 requests	All 229	Tool calls
Qwen2.5-7B					
Keep all	14	2,147	0.00	12.6	117
No descriptions	14	1,545	12.50	12.9	123
Demote (LOTS ranking)	14	1,587	**18.75**	15.7	143
LOTS pruning	2.0	186	12.50	**16.9**	173
LLM as router	1.2	128	0.00	11.3	117
Qwen2.5-14B					
Keep all	14	2,147	9.38	13.7	152
No descriptions	14	1,545	6.25	11.2	194
Demote (LOTS ranking)	14	1,587	6.25	14.0	166
LOTS pruning	2.0	186	9.38	14.9	163
LLM as router	1.6	159	3.12	12.7	118

C.3 Ranking Quality at a Fixed Budget

Every ranking is deployed at the same $K \in \{2, 4\}$ per family on the 320 evaluation requests, which separates the ranking from the label-based choice of the budget. LOTS uses the ranking behind the main table; Tool2Vec and Dense rank tools by similarity to the family's 80 fitting requests; Random averages five subsets per family.

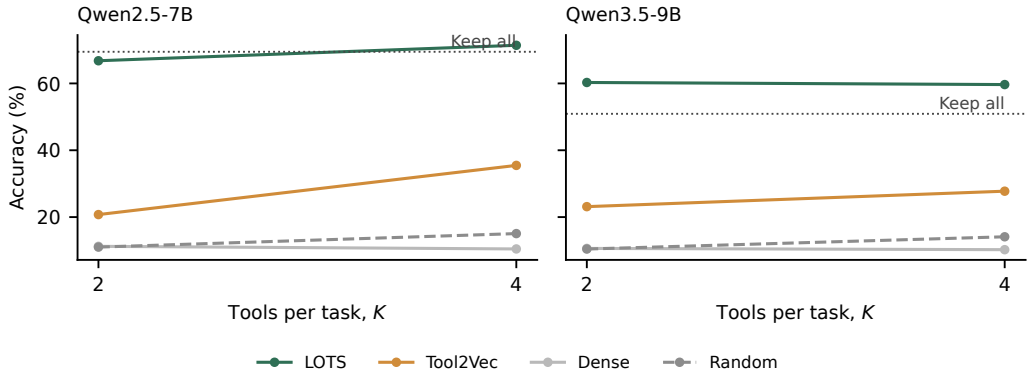

Figure 9: TGB accuracy against the budget K, every ranking deployed at the same K per family. Macro average over the ten families; Random is the mean over five subsets per family; the dotted line is Keep all.

Table 9: Paired accuracy difference, LOTS minus the baseline, at each fixed budget, with a 95% interval from a family-stratified paired bootstrap over evaluation requests (2,000 resamples). Positive values favor LOTS.

Baseline	$K = 2$	$K = 4$
Qwen2.5-7B		
Tool2Vec	+46.0 [+44.9, +47.1]	+35.9 [+34.6, +37.3]
Dense	+55.5 [+54.3, +56.8]	+60.9 [+59.5, +62.4]
Random	+55.7 [+54.5, +57.0]	+56.3 [+55.0, +57.7]
Qwen3.5-9B		
Tool2Vec	+37.2 [+36.2, +38.2]	+31.9 [+30.6, +33.2]
Dense	+49.7 [+48.5, +50.8]	+49.4 [+48.1, +50.7]
Random	+49.9 [+48.7, +51.0]	+45.6 [+44.2, +46.9]

At $K = 2$ and $K = 4$, LOTS is 32 to 61 points above Tool2Vec, Dense and Random on both executors, and every interval excludes zero. The fitted budgets of the main table lie in this range: seven of the ten Qwen2.5-7B families have $K \leq 4$. At a fixed budget no fitting label enters space construction.

D TOOL-SPACE CONSTRUCTION AND SUBSEQUENT UPDATES

These experiments follow tool spaces from their initial construction to later updates. (0) We first replace the metadata task assignment with request clustering (§D.1). (1) We then examine whether LOTS can construct and reuse a separate space for each task type as new types arrive. (2) We then keep the task distribution fixed and test whether updating an existing space improves on freezing it after the first batch. (3) Finally, we restrict the tools visible from the outset and test whether exploration can discover missing tools.

D.1 TASK ROUTING BY REQUEST CLUSTERING

The main experiments assign each request to a task with benchmark metadata: TGB families, BFCL API classes and annotated GTA-Atomic types. In deployment the assignment can come from metadata, from clustering request embeddings, or from a task router (Zhang et al., 2026b; Liu et al., 2026a; Zhao et al., 2026); a request that matches no existing task opens a new task space. This section replaces the metadata on TGB with unsupervised clustering of request embeddings and measures the cost of the replacement.

Protocol. Each request is embedded with a frozen sentence encoder, E5-base-v2 with the `query:` prefix or BGE-base-en-v1.5. The first fifth of TGB (800 requests) fits the router and the tool spaces; the remaining 3,200 requests evaluate them. A cluster replaces a family everywhere the main protocol uses one: contributions are aggregated within a cluster, the cluster's ranking is cut at a budget K selected on its own fitting requests, and each evaluation request is served under the space of the cluster it routes to. We compare the family router (R0) with k-means on cosine distance at a fixed k (R1), k-means at the k that maximises the silhouette score over $k \in [2, 20]$ (R2), and online threshold clustering with running-mean centroids (R3), against two controls: a single cluster that pools every request (C1) and a random partition matched to the cluster-size distribution of R1 at k=10 (C2). Family names do not occur in the request text, which we verify before clustering. Clustering quality is reported as purity, normalised mutual information and adjusted Rand index against the ten family labels, which the router never reads.

Clustering recovers the task partition. At k=10, k-means recovers the family partition exactly under both encoders, and accuracy equals that of the family router to four decimal places on both models (Table 10). The silhouette criterion selects k=10 under BGE without being told the number of families, and k=9 under E5, which merges two families at a cost below half a point. Metadata task labels are a convenience in this setting, and an unsupervised router supplies the same partition. TGB requests are generated from family-specific templates, so this setting is favourable to clustering; the result shows that LOTS is insensitive to replacing metadata with an unsupervised partition, not that real request streams separate as cleanly.

The gain comes from sharing evidence within a task. Both controls lose 8 points on Qwen2.5-7B and 8 to 9 points on Qwen3.5-9B. C2 keeps the cluster sizes and the fitted budgets of a correct partition and only reassigns which requests share a space, and it performs at the level of a single pooled cluster. The benefit therefore comes from accumulating contributions over requests that use the same tools, with the smaller menu alone accounting for none of it.

Over-splitting a task is cheap; merging tasks is costly. k=20 costs 0.03 points on Qwen2.5-7B, and the threshold router at τ=0.90 maintains 54 clusters, more than five per family, while staying within a point of the family router. Merging costs more as the budget tightens: at k=5, where purity falls to 0.500, accuracy drops by 1.3 points at K and by 19.6 points at a fixed K=2, since a space of two tools cannot serve two merged families at once. A practical router should therefore lean towards splitting. Thresholds do not transfer across encoders, because cosine similarities are not calibrated between them: under E5 the threshold router collapses to fewer than two clusters at τ=0.80 and loses 5.7 points on Qwen2.5-7B and 7.1 on Qwen3.5-9B, while BGE keeps separate clusters at the same threshold.

Table 10: **Routing by request clustering on TGB.** Clustering quality is measured on the 3,200 evaluation requests against the ten family labels; accuracy (%) is the macro average over families under the per-cluster fitted budget K, mean $\pm$ standard deviation over three clustering seeds. Deterministic routers have one seed.

Router	Clusters	Purity	NMI	ARI	Qwen2.5-7B	Qwen3.5-9B
R0: family label	10	1.000	1.000	1.000	82.84	65.53
R1: k-means, k=10, E5	10	1.000	1.000	1.000	82.84 ± 0.00	65.53 ± 0.00
R1: k-means, k=10, BGE	10	1.000	1.000	1.000	82.84 ± 0.00	65.53 ± 0.00
R2: silhouette k, E5 (k=9)	9	0.900	0.969	0.898	82.38	65.19
R2: silhouette k, BGE (k=10)	10	1.000	1.000	1.000	82.84	65.53
R1: k-means, k=5, E5	5	0.500	0.758	0.428	81.53 ± 0.71	62.92 ± 0.78
R1: k-means, k=20, E5	20	1.000	0.881	0.711	82.81 ± 0.12	64.92 ± 0.16
R3: threshold τ=0.90, BGE	54.3	1.000	0.869	0.817	82.00 ± 0.27	65.31 ± 0.24
R3: threshold τ=0.95, BGE	384	1.000	0.624	0.325	78.70 ± 0.06	60.72 ± 0.07
C1: one cluster	1	0.100	0.000	0.000	74.69	57.72
C2: random partition	10	0.128	0.006	0.000	74.72	56.62

D.2 Visualization of Task Space

Figures 10–12 show one fitted space per benchmark, the evolution of one space, and the agent's first tool call. In Figure 10, the `LOTS` space for TGB `fx_settle` is exactly the chain (13.1% to 63.1%); on BFCL TradingBot it documents in full the twelve functions the agent calls most (40.8% to 47.5%); on GTA-Atomic `web_fact` with Qwen3.5-9B it keeps six of eight tools (52.9% to 70.6%, Random 52.9%). Figure 11 follows the cold-start run for `fx_settle`: DocRetrieve enters after the second batch, ExchangeRate and Calculator after the fourth, and accuracy rises from 0% to 67.5% against 22.5% for the full registry. Figure 12 links the first tool call to the outcome on GTA-Atomic: under the `LOTS` space the agent opens 12 of 17 requests with ImageDescription, 9 of them correctly, with 2.9 calls per request against 3.9 under the full menu.

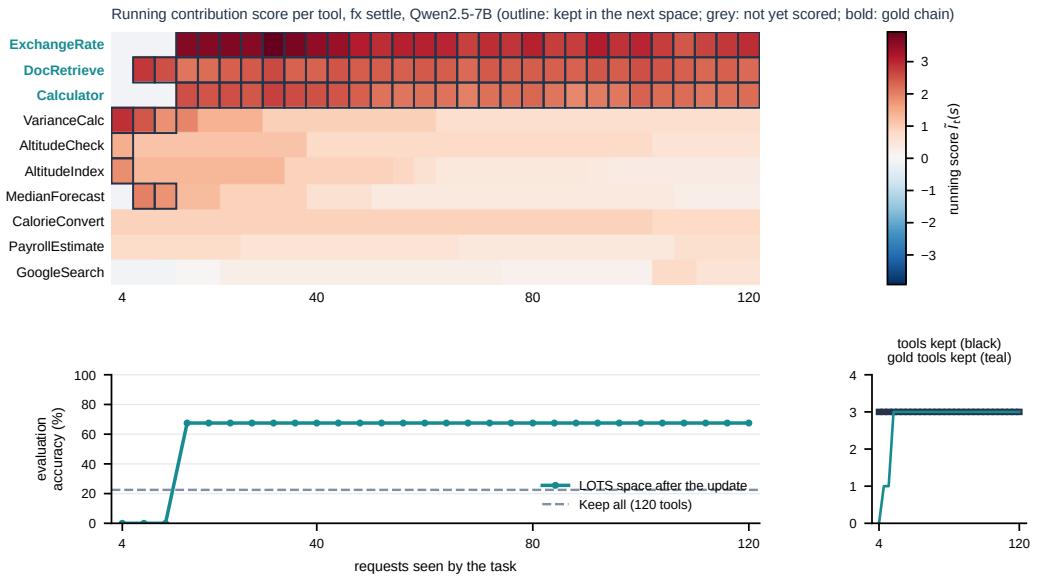

Figure 11: Evolution of one persistent space: TGB `fx_settle`, Qwen2.5-7B, 120-tool registry with a cap of 16 tools per request, batches of four requests, $\varepsilon = 0.5$, $\alpha = 0.3$, $K = 3$. Top: running score of the ten highest-scoring tools after each batch; outlines mark the tools kept in the next space; bold labels mark the task's chain. Bottom: accuracy of the updated space on the evaluation requests, and the number of tools and chain tools it keeps.

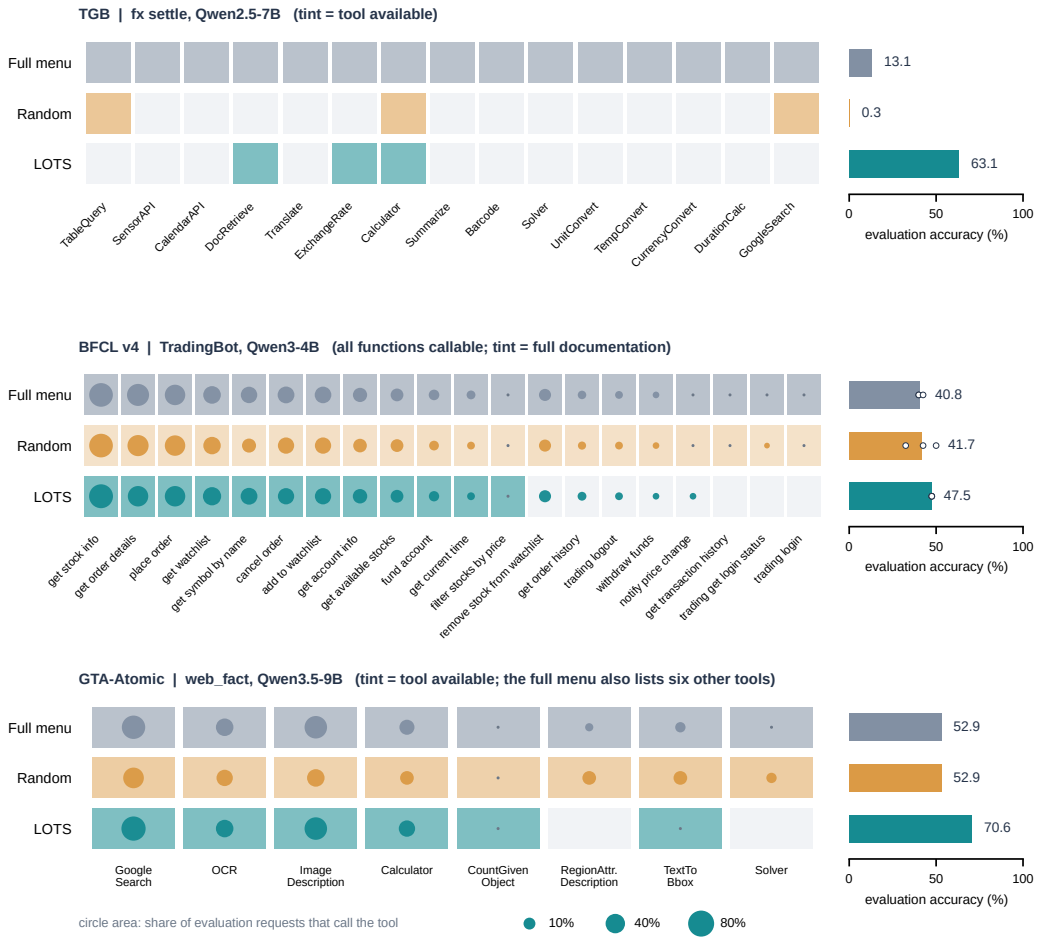

Figure 10: Tool space, tool use, and performance for one task per benchmark. Tint marks the tools in the space; on BFCL, where every function stays callable, it marks full documentation, and a paler tint for Random gives the share of records in which the tool is documented. Circle area is the share of evaluation requests that call the tool. TGB executes every served tool once per request, so its panel shows the configuration only. Bars give evaluation accuracy; white dots are the three seeds on BFCL.

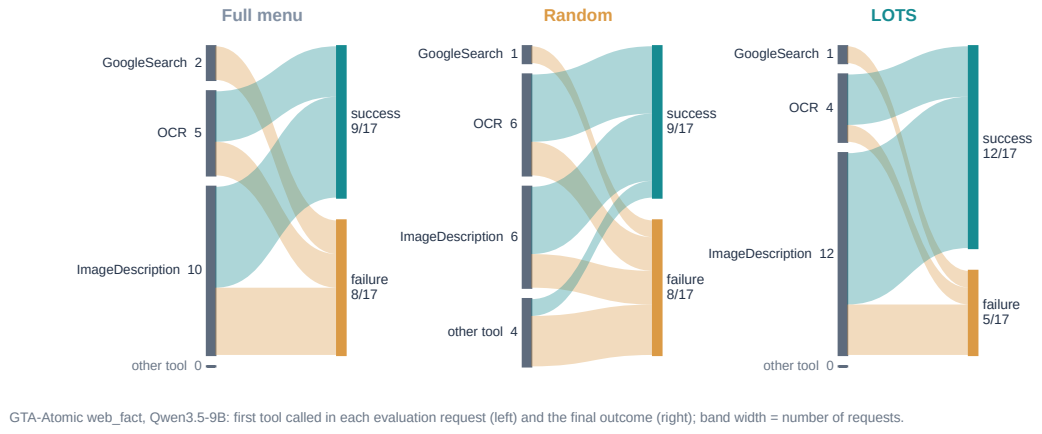

Figure 12: First tool call and final outcome on GTA-Atomic `web_fact` with the search tool enabled. Each band links the first tool the agent calls on an evaluation request to whether the request is answered correctly; band width is the number of requests. The figure describes association and omits the later calls of a request.

D.3 Sizes of Task Space

Table 11 reports the mean number of tools each method leaves to the agent: callable tools on TGB, fully documented tools on BFCL.

Table 11: Mean number of tools in the deployed space. TGB counts callable tools out of a 15-tool registry; BFCL counts fully documented tools out of 79 to 105 candidates per API class. Methods marked * use the budget K of LOTS by construction; methods marked † read correctness labels on the fitting requests; methods marked ‡ select per request, and their count is averaged over the evaluation requests.

TGB	Qwen2.5-7B	Qwen2.5-14B	Llama3.1-8B	Mistral-7B	Qwen3.5-9B	Qwen3-8B	phi-4	G4-12B
Keep all	15	15	15	15	15	15	15	15
BM25*	5.1	3.6	6.8	3.9	5.0	2.8	3.9	4.4
Dense*	5.1	3.6	6.8	3.9	5.0	2.8	3.9	4.4
Tool2Vec*	5.1	3.6	6.8	3.9	5.0	2.8	3.9	4.4
LLM as router‡	1.6	2.7	2.5	2.6	1.8	2.2	2.0	2.1
TTO†	12.0	15.0	3.0	1.0	2.0	4.0	12.0	15.0
LOTS	5.1	3.6	6.8	3.9	5.0	2.8	3.9	4.4
BFCL	Qwen3-4B	Qwen3.5-4B	Qwen3.5-9B	Gemma4-12B	Phi-4-mini			
Keep all	all	all	all	all	all			
BM25*	19.8	14.8	24.2	13.8	8.0			
LLM as router‡	4.7	4.7	5.0	5.5	6.6			
TTO†	4.3	5.0	9.0	3.5	0.8			
Tool2Vec	16	16	16	16	16			
LOTS	19.8	14.8	24.2	13.8	8.0			

LOTS keeps 2.8 to 6.8 of the 15 TGB tools (mean 4.4) and documents 8.0 to 24.2 tools per BFCL class; the router keeps 1.6 to 2.7 on TGB and 4.7 to 6.6 on BFCL. TTO ranges from one tool on Mistral-7B to the full registry on Qwen2.5-14B and Gemma4-12B, and Beam Search on Gemma4-12B deploys the empty space for eight of ten families.

D.4 Constructing Spaces as New Task Types Arrive

Figure 3 uses 120 tools, the 15 authored tools and 105 distractors. Eight families arrive at intervals of 15 steps (which eight depends on the model; for Qwen2.5-7B they are the seven of Figure 3 and `schedule_gap`); each batch has four requests, a family's first batch uses the full registry, and later batches use its current space with $\varepsilon = 0.25$, EMA weight $\alpha = 0.3$ and top-K retention with the budget from its 120-tool fit. Table 12 extends the main-text example to seven models and three seeds: LOTS finishes above Keep all in all 21 runs, with serving prompt tokens about one third of Keep all (scoring cost excluded).

Table 12: Task-arrival streams on seven models and three seeds. Accuracy is credited over arrived families at the end of the stream. Each entry is LOTS / Keep all. Prompt tokens are averaged over served requests, including exploration and excluding update scoring.

Model	Families	Seed 0	Seed 1	Seed 2	Prompt tokens
Qwen2.5-7B	8	0.82 / 0.52	0.82 / 0.43	0.79 / 0.46	641 / 1832
Qwen2.5-14B	8	0.87 / 0.76	0.86 / 0.80	0.85 / 0.82	623 / 1820
Qwen3-8B	8	0.42 / 0.21	0.53 / 0.23	0.37 / 0.20	629 / 1834
Qwen3.5-9B	8	0.62 / 0.30	0.59 / 0.32	0.61 / 0.29	659 / 1860
phi-4	8	0.77 / 0.48	0.77 / 0.53	0.74 / 0.49	531 / 1537
Mistral-7B	8	0.62 / 0.20	0.50 / 0.30	0.64 / 0.30	706 / 2026
G4-12B	8	0.84 / 0.77	0.87 / 0.70	0.87 / 0.73	637 / 1768

Figure 13 shows how request-level evidence forms a ranking and then a space on BFCL and GTA-Atomic. Figure 14 shows the TGB spaces under the top-K protocol: all contain their chain, with budgets from one tool for the two controls to fourteen for the sensor families.

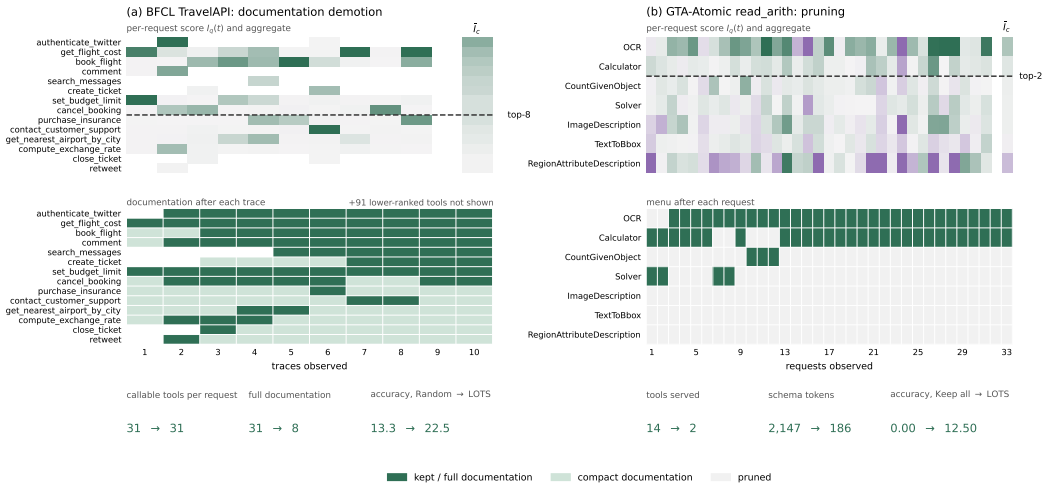

Figure 13: From request scores to a task ranking and tool space, Qwen2.5-7B. (a) BFCL `TravelAPI`, using documentation demotion. (b) GTA-Atomic `read_arith`, using pruning, fitted on 33 requests; the accuracies are those of Table 8 on the remaining 32.

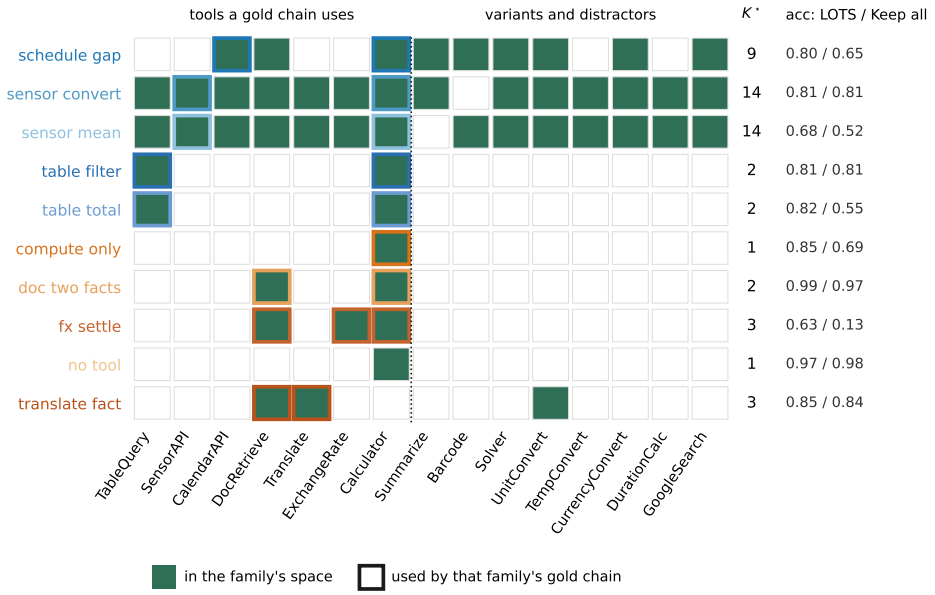

Figure 14: Task-specific tool spaces on TGB, Qwen2.5-7B, under the main protocol: each row is fitted on a family's first 80 requests and keeps its top-K tools. Outlined cells mark the family's gold chain; the right margin reports K and accuracy against Keep all on the remaining 320 requests.

D.5 Do Further Updates Improve an Existing Space?

We fix the task and request distribution and compare an updated space with one frozen after its first batch. In Figure 15, each TGB family runs eight batches of 40 requests with Qwen2.5-7B and the 15-tool registry; three arms share the request order and evaluation set: Keep all, a space frozen after

the first update, and the LOTS update applied after every batch (support-frequency retention, EMA $\alpha = 0.2$, threshold 5%). The first update carries most of the gain: the updated spaces reach 0.752 after the first batch and 0.757 after the eighth, against 0.750 for the frozen arm and 0.705 for Keep all, and prompt lengths stay at 56–205 tokens per request against 200–345 under the full menu. Two of the ten spaces change after the first update; the others confirm their configuration.

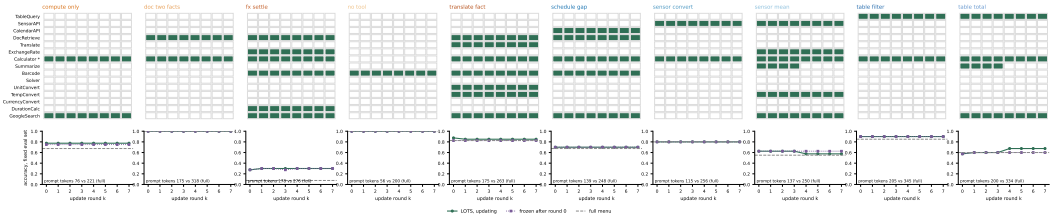

Figure 15: Stationary-task update trajectories, Qwen2.5-7B: 15-tool registry, eight batches of 40 requests. Top: tools retained by the updating arm after each round; outlines mark entries. Bottom: accuracy on each family's fixed evaluation set for the updating, frozen and Keep-all configurations.

Update rules under a serving cap. When the registry cannot be served in full, the first batch sees only a random sample of it and later requests are explored with probability ε (Appendix B.3). Table 13 compares three update rules in this setting under the same request order, exploration probability and seeds: the LOTS update, a size-matched random draw from the tools with evidence so far, and the union of all such tools. Each rule is a separate run, so the evidence differs once the spaces differ. LOTS reaches 0.27 with four tools that hold 41% of the required tools, against 0.01 for the random space and 0.02 for the union; the union's Gold of 0.29 follows from the cap, since its space already holds about 53 tools after the first round, so exploration adds nothing further, whereas the four-tool LOTS space keeps discovering tools.

Table 13: Update rules under a serving cap, Qwen2.5-7B: eight families, 120-tool registry, 30 rounds of four requests, $\varepsilon = 0.5$, three seeds; each rule is a separate run with the same request order, exploration probability and seeds. An exploring request receives the current space plus at most $16 - |S|$ further tools, and a space larger than 16 tools is served in full. Accuracy is read on each family's fixed evaluation set under the current space, averaged over the rounds and at the final round; size is the final space; gold is the share of the family's required tools in the final space.

Update rule	Runs	Mean acc.	Final acc.	Size	Gold
LOTS update (top-K)	24	0.162	0.265	4.0	0.41
Random, same K	24	0.010	0.006	4.0	0.00
Union of exposed tools	24	0.023	0.023	52.7	0.29

D.6 Worked Examples of Tool-Space Optimization

This section follows one task through the procedure of §3 with the recorded data of one run. The examples explain the mechanism; the evidence for accuracy is in §4.2 and §4.3. Light red marks evidence removed by an intervention and light orange marks content that changes because of it.

TGB: one task from request scores to a reused space. The task is fx_settle inside the task-arrival stream of Figure 3 (Qwen2.5-7B, seed 0, 120-tool registry), and the request is the first request of the task's first batch; no other selection was made. Requests, answers, scores, stored scores, spaces, and accuracies are copied from the run log; tool outputs are regenerated by the benchmark's deterministic tool code, which the log does not store.

1. Task and request

Task type fx_settle: a price and a quantity retrieved by DocRetrieve, a rate from ExchangeRate, combined by Calculator.
Model / run Qwen2.5-7B-Instruct; task-arrival stream, seed 0; batches of 4 requests, exploration ε=0.25,

EMA weight α=0.3.
Request `fx_settle_0125`, update round 0 (the task's first batch).
Space before the update the whole registry, 120 tools (15 authored, 105 synthetic distractors).
Budget K=3, chosen beforehand on the task's fitting requests under the 120-tool registry by accuracy (top-K accuracy K=1: 0.01, K=2: 0.00, K=3: 0.72, K=4: 0.51, K=6: 0.36).

User request

```
We are settling 12 units of Vexforn-X4 in the settlement currency, and 960.00 has already
  been prepaid.  By how much does the settlement value exceed the amount already prepaid?
```

2. Tool outputs and realized answer

On TGB the benchmark executes every served tool once per request and lists the outputs in the prompt; these are not calls chosen by the model, and the prompt contains no tool documentation.

Outputs of the task's tool chain

```
Calculator:  971.86
DocRetrieve:  DocRetrieve:
[1] Vexforn-X4 -- Vexforn-X4 is supplied in single units at a listed price of 63.52 per
  unit.
[2] Mirdan-X4 -- general background, no figures given.
ExchangeRate:  ExchangeRate:  1 unit = 1.275 settlement units
```

Outputs of four of the 117 other served tools (113 further lines omitted)

```
GoogleSearch:  No results found.
CurrencyConvert:  CurrencyConvert:  no monetary amount found.
Summarize:  Summarize:  the image contains no readable figures.
TorqueIndex:  TorqueIndex:  79.50
```

Realized answer y

```
11.86
```

Evaluation note, not part of any scoring input: the reference value is 11.86, so this answer is counted correct.

3. Leave-one-out intervention on `DocRetrieve` (excerpt)

Both columns end in the same target string, which is teacher-forced; only the tool outputs differ. On TGB removing a tool re-executes the chain without it, so the Calculator line changes as well; the other 116 output lines are identical in the two contexts.

Context h_x

```
Answer the question with a short final
  answer only.

Question:  We are settling 12 units of
  Vexforn-X4 in the settlement currency, and
  960.00 has already been prepaid.  By how
  much does the settlement value exceed the
  amount already prepaid?

Tool output:
GoogleSearch:  No results found.
Calculator:  971.86
DocRetrieve:  DocRetrieve:
[1] Vexforn-X4 -- Vexforn-X4 is supplied
  in single units at a listed price of
  63.52 per unit.
[2] Mirdan-X4 -- general background, no
  figures given.
ExchangeRate:  ExchangeRate:  1 unit =
  1.275 settlement units

Final answer:
 11.86  ← scored target y
```

Context h_x^{-s}, s = **DocRetrieve**

```
Answer the question with a short final
  answer only.

Question:  We are settling 12 units of
  Vexforn-X4 in the settlement currency, and
  960.00 has already been prepaid.  By how
  much does the settlement value exceed the
  amount already prepaid?

Tool output:
GoogleSearch:  No results found.
Calculator:  NameError:  name 'P' is not
  defined
ExchangeRate:  ExchangeRate:  1 unit =
  1.275 settlement units

Final answer:
 11.86  ← same target y
```

4. Likelihood contributions for this request

$\mathcal{L}_\theta$ is the mean log-likelihood of the tokens of the realized answer (here `' '`, `'1'`, `'1'`, `'.'`, `'8'`, `'6'`), located inside the full prompt; $I_x(s) = \mathcal{L}_\theta(y \mid h_x) - \mathcal{L}_\theta(y \mid h_x^{-s})$ is computed from unrounded values.

Tool s	$\mathcal{L}_\theta(y \mid h_x)$	$\mathcal{L}_\theta(y \mid h_x^{-s})$	$I_x(s)$	$I_x(s)$ in the run log
DocRetrieve	-0.1316	-2.6867	+2.555	+2.640
ExchangeRate	-0.1316	-2.7974	+2.666	+2.654
Calculator	-0.1316	-2.6290	+2.497	+2.554
Summarize	-0.1316	-1.2136	+1.082	+1.050
CurrencyConvert	-0.1316	-0.8356	+0.704	+0.723
GoogleSearch	-0.1316	-0.3350	+0.203	+0.200
TorqueIndex	-0.1316	-1.2569	+1.125	+1.028

The run log stores only $I_x(s)$. The two likelihood terms were recomputed for this appendix with the same prompt, answer-token positions, and normalization (HF transformers, CPU, bfloat16); the original run used vLLM on GPU, and the last column gives its value. The three chain tools carry the score. Distractor scores are small and can be positive: the score measures how much the evidence supports the answer the model gave, whether or not that answer is correct.

5. Aggregate over the batch and update the space

The task's first batch has 4 requests, all served under the registry, so every tool has 4 scored requests; the table shows the whole batch for seven of the 120 tools. Two of the four realized answers are wrong, and their chain tools still score high.

Request	`0125`	`0385`	`0034`	`0175`	Scored	Batch mean	Rank
Realized answer	`11.86`	`49.26`	`22.52`	`280.21 - 280.00 = 0.21`	requests	$\bar{I}_t(s)$	of 120
Counted correct	yes	no	no	yes			
DocRetrieve	+2.640	+3.278	+2.175	+0.746	4	+2.210	3
ExchangeRate	+2.654	+2.775	+3.136	+0.788	4	+2.338	1
Calculator	+2.554	+2.280	+3.576	+0.807	4	+2.304	2
Summarize	+1.050	-0.097	+0.041	-0.010	4	+0.246	7
TorqueIndex	+1.028	-0.118	+0.036	-0.002	4	+0.236	11
CurrencyConvert	+0.723	-0.061	+0.167	+0.054	4	+0.221	23
GoogleSearch	+0.200	-0.099	+0.046	-0.014	4	+0.033	120

Update top-K=3 of the ranking: `Calculator, DocRetrieve, ExchangeRate`. **Space** 120 tools $\rightarrow$ 3 tools.

Next batch Served under the three-tool space (no request was explored). The stored score is the batch mean after the first batch and an exponential moving average afterwards, $0.3\times$ new batch mean $+\,0.7\times$ stored; a tool that the batch did not expose keeps its stored score.

Tool	Stored after batch 1	Batch-2 mean	Stored after batch 2
ExchangeRate	2.3385	3.2876	2.6232
Calculator	2.3041	2.4274	2.3411
DocRetrieve	2.2097	2.2685	2.2273
MarginCalc (rank 4, not exposed)	0.2655	—	0.2655

6. Reuse on a subsequent request

Request `fx_settle_0351`, update round 1: served after the space was built, under that space (Calculator, DocRetrieve, ExchangeRate).

```
We are settling 14 units of Kalond-R3 in the settlement currency, and 540.00 has already
  been prepaid.  By how much does the settlement value exceed the amount already prepaid?
```

Tool outputs in the prompt (complete)

```
Calculator:  556.57
DocRetrieve:  DocRetrieve:
[1] Kalond-R3 -- Kalond-R3 is supplied in single units at a listed price of 36.24 per
  unit.
[2] Kalond-Q3 -- general background, no figures given.
ExchangeRate:  ExchangeRate:  1 unit = 1.097 settlement units
```

Realized answer

```
16.57
```

Evaluation note: the reference value is 16.57, so the answer is counted correct. This request was not also served under Keep all. On the task's 40 fixed evaluation requests the three-tool space scores 0.675 and Keep all 0.225; the second batch is answered 3 of 4 correctly with 143 prompt tokens per request against 1830 under the registry.

BFCL. The second example uses the per-class protocol of Table 2: Qwen3-4B-Instruct, API class `TradingBot`, fitted on the class's 10 fitting records; the record is the class's first fitting record and the turn its first turn with a tool call before the final message.

B1. Recorded turn, scoring target, and intervention (excerpt)

Record `multi_turn_base_100`, turn 0. The scored target is the last assistant message of the turn; it is never removed. Removing `get_stock_info` deletes the assistant message that calls it and the observation that call produced; the system prompt with the tool menu is identical in both contexts and is omitted here.

Context h_x (end of the turn)

```
user: I'm contemplating enhancing my
  investment portfolio with some tech
  industry assets, and I've got my eye on
  Nvidia. I'm keen to know its current
  stock price, and would appreciate if you
  could source this information for me.
assistant:
  [get_stock_info(symbol="NVDA")]
tool: {"price": 220.34,
  "percent_change": 0.34, "volume":
  1.234, "MA(5)": 220.45, "MA(20)":
  220.67}
assistant: The current stock price of
  Nvidia (NVDA) is $220.34, with a 0.34%
  increase in price today. The trading
  volume is 1.234 million shares, and its
  5-day Moving Average (MA(5)) is $220.45,
  while the 20-day Moving Average (MA(20))
  is $220.67.
```

Context h_x^{-s}, $s =$ get_stock_info

```
user: I'm contemplating enhancing my
  investment portfolio with some tech
  industry assets, and I've got my eye on
  Nvidia. I'm keen to know its current
  stock price, and would appreciate if you
  could source this information for me.
assistant: The current stock price of
  Nvidia (NVDA) is $220.34, with a 0.34%
  increase in price today. The trading
  volume is 1.234 million shares, and its
  5-day Moving Average (MA(5)) is $220.45,
  while the 20-day Moving Average (MA(20))
  is $220.67.
```

Logged score of `get_stock_info` on this record (mean over the record's scored turns): +0.147. The log keeps this record-level difference only; the two likelihood terms and the per-turn values were not stored.

B2. Full and demoted documentation of one tool

`get_transaction_history` is ranked 27 of 85 for this class, below K=20, so it is served with demoted documentation (function `_compress` of the implementation). The name, the parameter names, types and defaults, and the required list are kept; the class introduction and the per-parameter prose are removed; the first sentence of the tool's own description is kept. The `response` field is unchanged and omitted here.

Full documentation (265 tokens)

```
"name": "get_transaction_history"
"description": "This tool belongs to the
  trading system, which allows users to
  trade stocks, manage their account, and
  view stock information.
Tool description: Get the transaction
  history within a specified date range."
"start_date": {"type": "string",
  "default": "None"}
"description": "Start date for the
  history (format: 'YYYY-MM-DD')."
"end_date": {"type": "string", "default":
  "None"}
"description": "End date for the history
  (format: 'YYYY-MM-DD'). "
"required": []
```

Demoted documentation (202 tokens)

```
"name": "get_transaction_history"
"description": "Get the transaction
  history within a specified date range."
"start_date": {"type": "string",
  "default": "None"}
"end_date": {"type": "string", "default":
  "None"}
"required": []
```

Token counts are of this tool's JSON definition under the Qwen3-4B tokenizer; they are not the cost of a request.

B3. Resulting task configuration

Full documentation the top K=20 of the class's 85 ranked tools: 12 tools of the trading API (`get_order_details`, `get_watchlist`, `get_stock_info`, `get_symbol_by_name`, `place_order`, `add_to_watchlist`, `get_account_info`, `fund_account`, `cancel_order`, `get_available_stocks`, `get_current_time`, `filter_stocks_by_price`) and 8 tools of other APIs that appear in the class's records.

Demoted documentation every other tool in a record's menu, including 8 tools of the trading API (`get_transaction_history`, `get_order_history`, `trading_login`, `notify_price_change`, `trading_get_login_status`, `remove_stock_from_watchlist`, `withdraw_funds`, `trading_logout`).

Effective tools remain callable.

On this class's 40 evaluation records (three seeds): LOTS 47.5/47.5/47.5 (mean 47.5); random ranking at the same budget 50.0/32.5/42.5 (mean 41.7); Keep all, mean of three seeds, 40.8. K was chosen on the 10 fitting records.

E Analysis of Tool Scoring

The following diagnostics examine the evidence used by the score, the attribution rule, and the stability of the aggregate ranking. They use different scopes to isolate these choices: request-level selection, pooled configurations across TGB families, and the task-specific spaces of the main experiments. Request-level thresholding retains tools above a tuned threshold $\tau^\star$; support-frequency retention uses the rule defined in Appendix D.5, with an explicitly reported exception for Llama-3.1-8B. Results from different scopes are not treated as matched comparisons.

E.1 Top-K Selection as Surrogate Maximization

`LOTS` measures each tool's contribution with the other tools present, then selects a task-specific space from the resulting scores. This section restates Theorem 1 of Section 3.2 with its setting and proof: the selection exactly optimizes an additive approximation to the fixed-output likelihood objective. Bounded tool interactions then give an explicit bound on the approximation error and the resulting selection gap. Consider one task and a fixed collection of requests Q served under a common tool menu S, with $m = |S|$. For each request x, keep its recorded output y_x fixed. Let $h_x(A)$ denote the scoring context obtained by retaining the evidence associated with $A \subseteq S$, using a fixed intervention rule and leaving tool documentation unchanged. Define

$$F(A) = \frac{1}{|Q|} \sum_{x \in Q} \mathcal{L}_\theta(y_x \mid h_x(A)). \tag{10}$$

Thus, F measures support for the recorded outputs, rather than the accuracy of newly generated answers. The task-averaged `LOTS` score is

$$I(s) = F(S) - F(S \setminus \{s\}). \tag{11}$$

These scores define the additive approximation

$$\widehat{F}(A) = F(S) - \sum_{s \in S \setminus A} I(s). \tag{12}$$

It agrees exactly with F at the full menu and at every single-tool deletion. Since $\widehat{F}(A) = F(S) - \sum_{s \in S} I(s) + \sum_{s \in A} I(s)$, retaining the K highest-scoring tools exactly maximizes $\widehat{F}(A)$ subject to $|A| = K$. The approximation arises only when several tools are removed together: their joint effect need not equal the sum of their individual leave-one-out effects.

Tool interactions. Following the discrete-difference formulation of set-function interactions (Sundararajan et al., 2020), define

$$\Delta_{s,t} F(B) = F(B \cup \{s,t\}) - F(B \cup \{s\}) - F(B \cup \{t\}) + F(B), \tag{13}$$

for distinct $s, t \notin B$. This quantity measures how the presence of one tool changes the marginal contribution of the other. Positive values indicate complementarity in the likelihood objective; negative values indicate substitutability. The following result bounds the error of the LOTS approximation using these interactions.

Theorem 1 (selection under bounded interactions)**.** Fix a budget K and let $r = m - K$. Suppose that, for some $\beta \geq 0$,

$$|\Delta_{s,t} F(B)| \leq \beta \tag{14}$$

for every $B \subseteq S$ with $K \leq |B| \leq m - 2$ and every distinct $s, t \in S \setminus B$. Then every set A of size K satisfies

$$|F(A) - \widehat{F}(A)| \leq \binom{r}{2}\beta. \tag{15}$$

Let A_{LOTS} contain the K highest-scoring tools, and let $A^\star \in \arg\max_{|A|=K} F(A)$. Then

$$0 \leq F(A^\star) - F(A_{\mathrm{LOTS}}) \leq r(r-1)\beta. \tag{16}$$

Proof. For any A with $|A| = K$, order the removed tools as $S \setminus A = \{d_1, \ldots, d_r\}$. Set $S_0 = S$ and $S_i = S_{i-1} \setminus \{d_i\}$. The likelihood loss at deletion i is

$$\ell_i = F(S_{i-1}) - F(S_i).$$

Its difference from the full-context score $I(d_i)$ comes from the preceding $i - 1$ deletions. Each preceding deletion changes this marginal by a second-order difference of the form equation 13. All corresponding contexts have sizes between K and $m - 2$, so the assumption gives

$$|\ell_i - I(d_i)| \leq (i-1)\beta.$$

Telescoping the deletion losses yields

$$F(A) = F(S) - \sum_{i=1}^{r} \ell_i.$$

Consequently,

$$|F(A) - \widehat{F}(A)| \leq \sum_{i=1}^{r} |\ell_i - I(d_i)| \leq \sum_{i=1}^{r} (i-1)\beta = \binom{r}{2}\beta.$$

Write $\delta = \binom{r}{2}\beta$. Because A_{LOTS} maximizes $\widehat{F}$ among sets of size K,

$$F(A^\star) \leq \widehat{F}(A^\star) + \delta \leq \widehat{F}(A_{\mathrm{LOTS}}) + \delta \leq F(A_{\mathrm{LOTS}}) + 2\delta,$$

which proves the selection bound. For $r = 0$ or $r = 1$, the approximation is exact. □

The bound requires that a tool's marginal contribution not change sharply as other tools are removed; each interaction in equation 13 costs four fixed-output evaluations, so sampled pairs can test it. The analysis covers one common-menu batch and the fixed-output objective, not running scores across menus or execution accuracy.

E.2 Scoring Targets and Attribution

Table 14 tests the scoring target on Qwen2.5-7B with request-level selection on TGB: scoring the same own output under every intervention reaches 0.724, against 0.648 when each candidate context generates its own answer and 0.796 with gold answers, which use labels; removing the tuned threshold grows the retained set from 3.7 to 10.4 tools and lowers accuracy to 0.698.

Table 14: Request-level scoring variants on TGB, Qwen2.5-7B. Gold-answer scoring uses labels; per-candidate generation scores a different string under each intervention. These configurations differ in retained size.

Variant	Tools kept	Accuracy
Fixed own output, LOO, $\tau^\star$	3.7	0.724
Fixed own output, LOO, $\tau = 0$	10.4	0.698
Per-candidate generation (L_{self})	—	0.648
Gold-answer scoring	12.2	0.796

Figure 16 illustrates the attribution on one request. On GTA-Atomic, scoring the numeric answer expressions selects Calculator+OCR in all five scoring runs (16.92%), whereas scoring the whole output selects OCR+RegionAttributeDescription (7.69%); 9 to 10 of the 65 outputs per run contain no numeric expression and are excluded. TGB scores its short answer and BFCL the final assistant message.

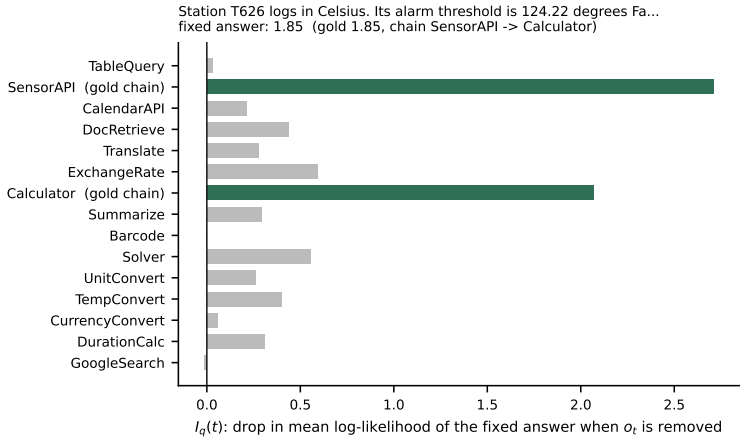

Figure 16: Tool scores for one TGB `sensor_convert` request, Qwen2.5-7B. The full-menu answer is fixed while each tool is removed using the TGB intervention, including downstream propagation where applicable. The two gold-chain tools carry the score.

Attribution in compositional tasks. In Table 15, a document-only selector reaches 0.418 on Qwen2.5-7B and per-request BM25 at four tools 0.193 (0.182 on Qwen3.5-9B); this request-level variant is separate from the task-level, budget-matched BM25 of Table 1. Exact Shapley values over the seven chain tools correlate between likelihood and correctness valuations (Pearson 0.95, Spearman 0.96, one sign disagreement), yet retaining the positive-valued tools reaches 0.370 and 0.360 against 0.795 for the seven-tool union: averaging over coalitions penalizes a tool whose partners are often absent.

Table 15: TGB coalition diagnostics over the seven chain tools. Positive Shapley values define the retained sets; the best subset is selected in hindsight. The final two rows are a separate single-tool diagnostic. Request-level `LOTS` uses a different scope and is reported in Table 14.

Diagnostic	Value	Accuracy
$\text{Shapley}^{\text{acc}}$ vs $\text{Shapley}^{\text{lik}}$, Pearson / Spearman	$+0.95$ / $+0.96$	
Sign disagreements	1 of 7 tools	
Keep $\{\phi_i^{\text{lik}} > 0\}$	4 tools	0.370
Keep $\{\phi_i^{\text{acc}} > 0\}$	5 tools	0.360
Union of all chain tools	7 tools	0.795
Best subset in hindsight		0.870
Single-tool score, AUROC vs one-tool success	0.789	
Best single tool deployed alone		0.135

Enumerating GTA-Atomic tool combinations. Table 16 evaluates all 15 pairs of six local tools on the 65 `read_arith` requests with Qwen2.5-7B. `LOTS` improves over Keep all by 7.69 points (95% interval $[-1.54, +16.92]$, $p = 0.18$) and over No tools by 10.77 ($[+3.08, +20.00]$, $p = 0.039$). In an independently built stack, Calculator+OCR reaches 21.54% against 7.69% for Keep all, 9.23% for size-matched Random and 23.08% for the gold oracle, and pooling the router's per-request choices over the task also yields Calculator+OCR.

Table 16: GTA-Atomic combination diagnostic, Qwen2.5-7B, using the same 65 `read_arith` requests for scoring and evaluation. Accuracy averages replicate executions. Δ is the paired difference of `LOTS` answer-expression scoring against the row, with a 95% bootstrap interval in percentage points.

Space	Tools	Replicates	Accuracy (%)	Δ (pp)
No tools	none	5	6.15	$+10.77$ $[+3.08, +20.00]$
Keep all (local tools)	all six	5	9.23	$+7.69$ $[-1.54, +16.92]$
No-API oracle	per-request gold	5	15.38	$+1.54$ $[0.00, +4.62]$
Calculator only	Calculator	3	4.62	
OCR only	OCR	3	15.38	
`LOTS`, answer expressions	Calculator + OCR	5	**16.92**	
`LOTS`, whole output	OCR + RegionAttributeDescription	3	7.69	
`LOTS` top-4, answer expressions	four tools	3	15.38	
Best of 15 pairs (hindsight)	OCR + Solver	3	23.08	
Rank of the `LOTS` pair	2 of 15			

Likelihood scoring versus an LLM trace judge. The judge replaces the likelihood score: the serving model answers each fitting request under the full menu, is asked which tools' outputs support its frozen answer (Zheng et al., 2023), tools are ranked by selection frequency, and the top-K form the space. In Table 17, the judge improves over Keep all on four of five models, and likelihood scoring is higher on all five, by 0.012 on Qwen2.5-14B to 0.206 on Qwen2.5-7B.

Table 17: Trace-judge ablation on TGB: the space is built from LLM judgments of the frozen traces instead of likelihood scores, with the same task-level aggregation and budget K.

	Qwen2.5-7B	Qwen2.5-14B	Mistral-7B	Qwen3.5-9B	Qwen3-8B
Keep all	0.696	0.880	0.464	0.509	0.565
Trace judge	0.623	0.906	0.558	0.613	0.689
`LOTS`	**0.829**	**0.918**	**0.658**	**0.652**	**0.815**

E.3 Deletion Operation in the Leave-One-Out Score

On TGB, removing s re-executes the chain, so a downstream tool that needs s's output returns an error. We compare this *re-execute* deletion with *evidence only*, which deletes just the recorded output line of s, on the same 80 fitting requests per family, the same traces and the same frozen answers (Qwen2.5-7B).

Table 18: Two deletion operations on Qwen2.5-7B: chain identification on the fitting requests and evaluation accuracy of the spaces they select at fixed budgets. The mean rank is over the chain tools of the nine tool-using families (1 is best).

	Chain identification			Evaluation accuracy (%)		
Deletion	Mean rank, upstream	Mean rank, downstream	Chain in top-$\lvert G\rvert$	$K=2$	$K=4$	$K=8$
Re-execute	1.11	2.00	9 / 9	66.8	71.4	72.3
Evidence only	1.56	1.56	9 / 9	66.6	71.4	72.3

Both deletions place the complete chain of every tool-using family within its top $|G|$ positions, their pooled scores correlate at 0.91, the top-4 sets agree in all ten families and the top-2 sets in nine, and the deployed accuracy is identical at $K = 4$ and $K = 8$ and differs by 0.2 points at $K = 2$. Evidence-only deletion halves the upstream scores (DocRetrieve in `fx_settle` 1.21 to 0.87, SensorAPI in `sensor_mean` 1.86 to 0.55), because the recorded downstream output still carries the upstream information; the mean upstream rank moves from 1.11 to 1.56 and Calculator moves above ExchangeRate in `fx_settle`. Re-execution credits an upstream tool with its effect through its dependents, the quantity a deployment removes when it drops the tool.

Table 19: **Likelihood ranking and execution utility on TGB.** Accuracy is averaged across ten task families, using 320 evaluation requests per family. The three configurations are those of Table 1: Top-K and Random use identical family-specific budgets, and Random uses a single fixed-seed draw. Bold denotes the best configuration for each model.

Model	Top-K	Random	Keep all
Qwen2.5-7B	**0.829**	0.234	0.696
Qwen2.5-14B	**0.918**	0.193	0.880
Llama3.1-8B	**0.848**	0.413	0.777
Mistral-7B	**0.658**	0.115	0.464
Qwen3.5-9B	**0.652**	0.275	0.509
Qwen3-8B	**0.815**	0.101	0.565
phi-4	**0.921**	0.243	0.797
Gemma4-12B	**0.914**	0.200	0.867
Mean	**0.819**	0.222	0.694

E.4 From Likelihood Scores to Execution Utility

For each of eight models and ten TGB families, the scores cached from the first 80 requests select the K highest-scoring tools (Top-K) and one random subset of the same size, evaluated on the remaining 320 requests. Excluding `no_tool`, seven models separate the reference-chain tools from the rest with an AUROC of 1.000 on all nine families and Gemma4-12B with 0.992, so 71 of 72 model–family pairs are separated perfectly. In Table 19, Top-K beats Random and Keep all on every model, averaging 0.819 against 0.222 and 0.694: the ranking, not the size of the space, carries the gain. Appendix C.3 deploys the same rankings at fixed budgets.

E.5 Scoring Models, Aggregation, and Incorrect Outputs

Rescoring the same instruct-model traces and frozen outputs with the base model, then serving with the instruct model, selects the same tools for Qwen2.5-14B, Mistral-7B and Qwen3-4B and gives higher accuracy for the other three models (Figure 17). The instruct scorer is closer to the gold chain on five of six models, and base scores are smaller on five, so fewer tools pass the fixed threshold of the support-frequency rule used here; at a fixed K, top-K selection is invariant to a uniform rescaling of the scores.

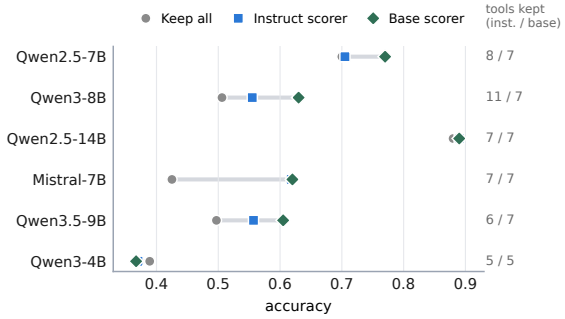

Figure 17: Scoring-model diagnostic on TGB: accuracy under Keep all and under the spaces fitted with an instruct-model or a base-model scorer, from frozen instruct-model outputs and a fixed support-frequency threshold. The instruct model serves every configuration; the right margin gives the number of tools each scorer keeps.

Incorrect outputs and aggregation. Figure 18 uses pooled configurations fitted from 3,000 traces per model and evaluated on 400 separate requests, with correct-only and wrong-only subsets and synthetic corruption of a random fraction of the scored answers ($1.17g + 3$ for numeric gold g, three draws). On Qwen2.5-7B the correct-only and wrong-only rankings correlate at Spearman 0.911; support-frequency retention keeps its eight-tool space through 50% corruption, whereas mean-score retention falls to 0.620 at 50% and 0.102 at 75%; Llama-3.1-8B declines from 0.765 to 0.690 at 50%

corruption. Incorrect outputs retain useful evidence, with robustness depending on the model and the retention rule; these pooled diagnostics do not cover the per-task top-K protocol.

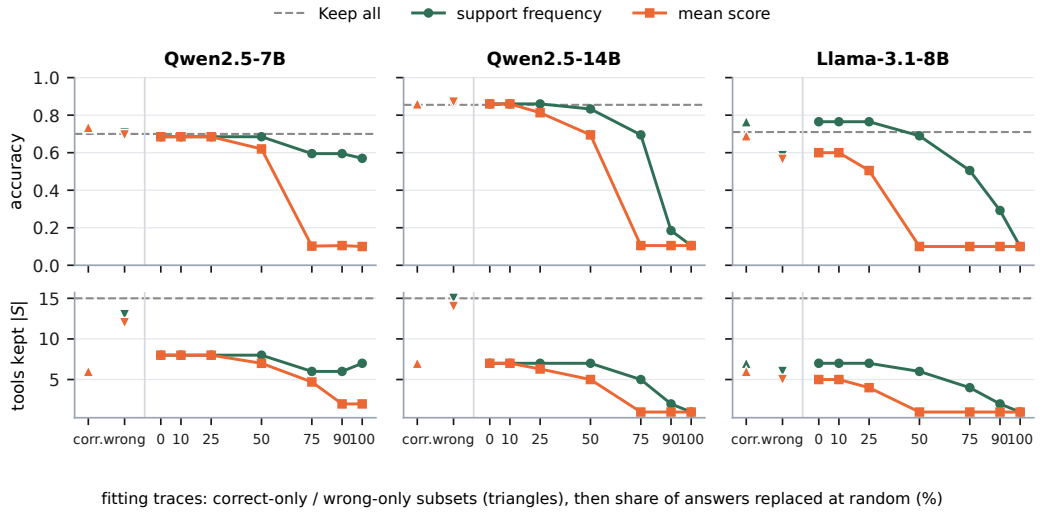

Figure 18: Incorrect-output diagnostic: 3,000 fitting traces per model and 400 separate evaluation requests. Triangles give label-filtered subsets (correct only, wrong only); lines replace a growing share of answers at random (three draws). Top: accuracy; bottom: tools kept. Support-frequency retention uses a 3% threshold for Llama-3.1-8B and 5% otherwise; mean-score retention uses $\tau = 0.1$.

E.6 Sensitivity to the Number of Traces

Figure 19 subsamples the 3,000-trace Qwen2.5-7B pool thirty times at each size: with 100 traces the ranking reaches a Spearman correlation of 0.957 and a Jaccard overlap of 0.95 with the full-pool fit, with 400 traces 0.986 and 0.97. The main experiments fit from 80 requests per family.

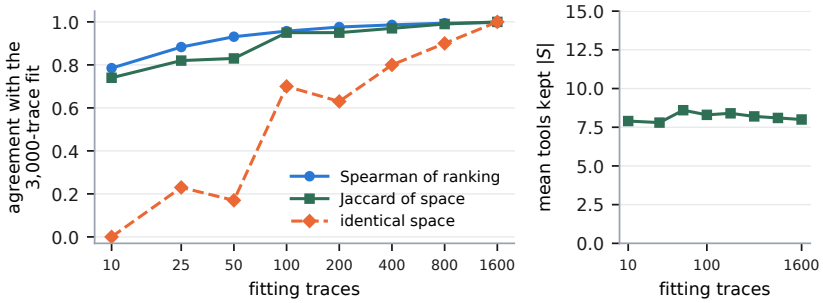

Figure 19: Trace-count sensitivity, TGB and Qwen2.5-7B. Thirty subsamples per size are compared with the configuration fitted from all 3,000 traces: rank correlation, Jaccard overlap of the space, and the fraction of subsamples with an identical space (left); mean space size (right).

F Computational Cost and Registry Scaling

A trace with m interventions needs at most $m+1$ teacher-forced sequences, one with the full context and one per intervention; no replacement answer is generated. Table 20 reports selection and serving input tokens: the GTA-Atomic fit scores 33 requests over seven candidates (264 sequences) at fixed $K = 2$, and the BFCL rows use the 101-record tune protocol at fixed $K = 16$. A router pays its selection call on every request; a frozen space pays once. The full-menu runs that produced the

fitting traces are excluded from the selection column; charging them adds about 98k prompt tokens on GTA-Atomic and 29M on BFCL. Serving tokens count every agent turn. On GTA-Atomic the 14-tool schema has 2,147 tokens and the OCR+Calculator schema 186, and Qwen2.5-7B processes 2,956 against 1,572 prompt tokens per request; the size-matched random menu uses 671 tokens but also makes fewer calls. On TGB, serving under the fitted spaces uses 143 to 181 prompt tokens per request against 251 to 319 for the full fifteen-tool menu, 33 to 46% fewer.

Table 20: Input-token costs of selecting and serving tool spaces. LOTS selection counts likelihood scoring. Router selection is paid per request; other selection costs are per fit. A lower serving cost comes with lower accuracy for Router and Random: both are below LOTS on every model of Tables 1–3. Inference counts all serving prompts for one request or BFCL record. Pays back divides reported LOTS selection tokens by serving tokens saved against Keep all. – denotes an inapplicable or unrun configuration. The Gemma4-12B TTO and Beam Search cells count the local search, 13 and 27 candidate spaces evaluated on the 33 fitting requests.

	Selection				Inference						LOTS
Model	Router	TTO†	Beam†	LOTS	Keep all	Random	Router	TTO†	Beam†	LOTS	Pays back
BFCL: fit on 101 tune records; inference sums all turns; payback is in records											
Q3-4B	1,513	1.72B	–	60.41M	289k	127k	31k	225k	–	174k	526
Q3.5-4B	1,530	1.66B	–	59.88M	243k	146k	47k	202k	–	190k	1,130
Q3.5-9B	1,530	1.24B	–	61.49M	352k	161k	46k	219k	–	238k	539
Q3-8B	1,513	1.37B	–	55.61M	207k	121k	43k	164k	–	162k	1,244
G4-12B	1,569	0.95B	–	63.39M	198k	95k	33k	125k	–	130k	931
Phi-4-mini	1,490	0.61B	–	69.27M	147k	67k	27k	89k	–	88k	1,179
GTA-Atomic read_arith, 33/32 diagnostic split: fit on 33 requests; inference and payback units are requests											
Q2.5-3B	385	0.71M	1.95M	101k	9,697	3,804	3,018	3,525	3,525	4,935	22
Q2.5-7B	385	0.27M	0.63M	101k	2,956	671	550	988	1,095	1,572	77
Q3-8B	385	0.42M	1.29M	101k	9,727	3,968	1,744	3,555	2,115	1,594	12
Q2.5-14B	385	0.24M	0.59M	101k	3,382	542	679	716	634	867	40
Q3.5-9B	401	0.64M	1.71M	102k	12,617	3,726	2,296	5,550	5,550	4,139	12
phi-4	381	0.88M	2.38M	86k	15,174	5,835	884	6,169	6,169	5,774	9
G4-12B	385	2.40M	5.25M	102k	6,531	808	1,304	385	1,392	1,275	19

For a frozen configuration with fitting cost C_{fit} and per-request serving costs $c_{\text{full}} > c_{\text{space}}$, the break-even count is

$$N_{\text{break}} = \frac{C_{\text{fit}}}{c_{\text{full}} - c_{\text{space}}}. \tag{17}$$

Figure 20 plots this account for GTA-Atomic; token break-even differs from latency break-even, since teacher-forced scoring, cached prefill and decoding cost different amounts per token. A deployment that keeps updating also pays for later scoring rounds, so over R rounds the total is

$$C_{\text{total}} = \sum_{k=0}^{R-1} \left(C_{\text{serve},k} + C_{\text{score},k} + C_{\text{extra trace},k} \right), \tag{18}$$

and Table 12 reports the serving component only.

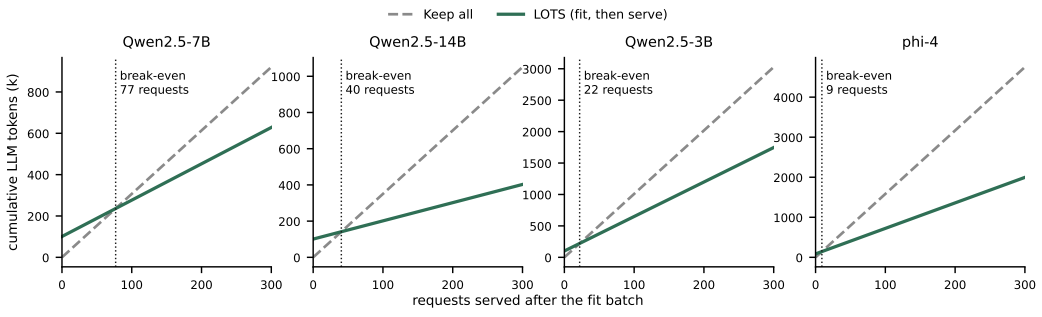

Figure 20: GTA-Atomic read_arith token amortization. LOTS starts with likelihood-scoring input tokens for the 33 fitting requests and accumulates serving input tokens; Keep all starts at zero and accumulates its own serving cost. The plotted account excludes any additional cost of collecting fitting traces.

F.1 Scaling with Registry Size

Figure 21 appends synthetic distractors to the 15-tool registry; two rounds each serve 400 requests from all ten families, the second under one pooled configuration fitted from the first. Full-menu accuracy falls from 0.677 at 15 tools to 0.508 at 120 on Qwen2.5-7B, where the fitted 43-tool configuration reaches 0.598; Qwen3.5-9B's fitted configurations keep about seven tools and average 0.607, 0.612, 0.594, and 0.538 over four seeds at 30, 60, 120, and 240 tools, while its full menu falls from 0.446 to 0.345; Figure 1(c) plots these means. Scoring time grows from 176 seconds at 15 tools to 8,158 seconds at 120 on one L40S, and to 3,437 seconds under the fitted configuration.

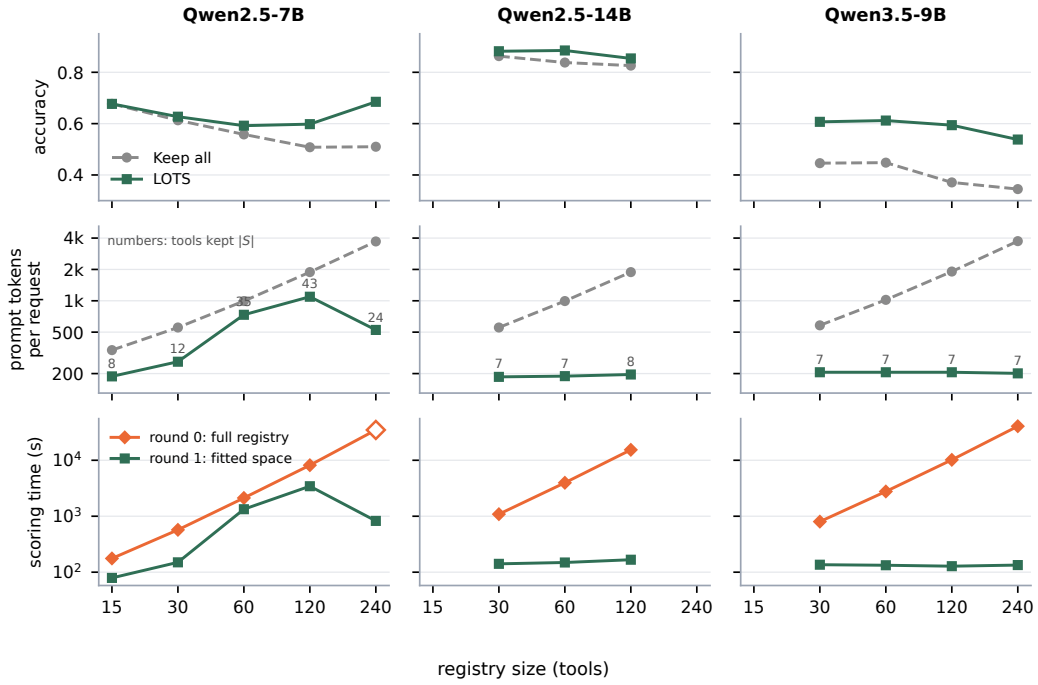

Figure 21: TGB registry-size diagnostic with one pooled configuration and two rounds of 400 requests. Top: accuracy under Keep all and under the fitted space. Middle: serving prompt tokens per request, with the number of retained tools above each point. Bottom: scoring time on one L40S for round 0 (full registry) and round 1 (fitted space); this is scoring time, not generation latency. The hollow marker uses a different batch cap. Qwen2.5-7B and Qwen2.5-14B average three seeds at 30 to 120 tools and Qwen3.5-9B averages four seeds for accuracy and tokens; the other points, and all timings, use one seed.