Route, Don't Overload: Large Tool Spaces Are a Cost Problem, and the Payoff of Subagents Depends on the Model

Abdul Matin
Amazon
ablmatin@amazon.com

Fatemeh Taheri Dezaki
Amazon
fatimaat@amazon.com

Saman Saboksayr
Amazon
samiatto@amazon.com

Naveed Janvekar
Amazon
njjanvek@amazon.com

Abstract

LLM agents now draw on tool spaces of hundreds of MCP servers and thousands of APIs, and placing every tool schema in one agent's context scales poorly. Two fixes are common in practice but rarely compared under controlled conditions: retrieving a few tools into a single agent, or splitting the tools across an orchestrator and several subagents. We build a τ-bench testbed with open-weight models that holds the base model fixed, grows the tool space with injected distractors up to 400 tools, and compares six allocation policies. For a capable model a large tool space mainly costs tokens: giving one agent every tool leaves success within noise and raises raw token cost about $4\times$. Per-turn retrieval or prompt caching removes most of that extra cost. The policies differ in accuracy. Over thirty tasks per domain at 120 tools, splitting tools across subagents with per-subtask retrieval beats a single dynamic agent on both domains (airline .67 vs. .28). A control that keeps the plan but runs every subtask in one context is as accurate, so the gain comes from decomposition with per-subtask retrieval. The cost effect appears on all four models we test, but the best fix depends on the model: Qwen3-235B's subagents often never issue the final write, and splitting hurts it.

1 Introduction

Tool-augmented LLM agents increasingly operate over large tool spaces. The Model Context Protocol (MCP) alone has produced registries of thousands of servers, and long-context function-calling studies report accuracy dropping by as much as 85% as the number of exposed tools grows [1]. Certified tool-selection correctness collapses to roughly 20% once distractor tools are inserted into the pool [2]. The single-agent "put every tool in the prompt" design does not scale.

Two families of remedies have emerged. **Tool retrieval** embeds tool descriptions and injects only the top-k into a single agent's context, tripling selection accuracy while halving prompt tokens [3–5]. **Multi-agent orchestration** instead spawns specialized subagents that each carry a subset of the tools; production frameworks (LangGraph, CrewAI) and Anthropic's research system report that giving each agent a focused tool set improves success [6]. But multi-agent systems can also fail to beat single agents [7] and cost roughly $15\times$ the tokens of a single chat [6]; others argue for single-threaded agents on reliability grounds [8].

These remedies are seldom compared head-to-head, and the multi-agent literature treats the per-subagent tool set as a hand-made design choice rather than a variable to be studied. This leaves a

Managing Agents that Manage Agents: Workshop on Responsible Use of Meta-Agents (NeurIPS 2026)

basic question open: when a query needs tools from a large space, is it better to retrieve tools into one agent or to split them across subagents, and if split, should each subagent receive dynamically retrieved neighbor tools or a fixed cluster of domain tools?

We answer this empirically. We build a τ-bench testbed in which the tool-allocation policy is the only variable, sweep the tool space over $\{\text{native}, 30, 60, 120\}$ tools (and to 400 on retail) across the retail and airline domains, and measure reliability per token and dollar cost alongside accuracy. We find that a capable base model keeps its accuracy as the space grows, while the all-tools agent's token cost rises about fourfold. Re-retrieving the top-k tools each turn, or caching the schema prefix, removes most of that extra cost. Accuracy is where the policies differ. Partitioning tools across subagents, selected per subtask, beats a single dynamic agent at 120 tools on both domains, matches the all-tools agent on RETAIL at about a quarter of its tokens, and holds its accuracy on the harder domain, where the single dynamic agent collapses (§7, §7.2).

Contributions. We give controlled, end-to-end evidence on when existing techniques (retrieval, dynamic tool selection, orchestrator–worker decomposition) pay off. (1) We frame *agent–tool allocation* as one problem coupling which tools an agent sees with whether they sit in one context or are partitioned across agents, and define six policies spanning the single/multi-agent and dynamic/static axes (§3). (2) We build a controlled τ-bench [9] harness that fixes the base model and grows the tool space with injected distractors under a faithful orchestrator–worker implementation with one shared token budget (§4, §5). (3) Across two domains, four base models, tool spaces out to 400 tools, and a semantic-distractor regime, we show that large tool spaces mainly raise token cost for a strong model, that per-turn retrieval or prefix caching removes most of that cost, and that this cost effect holds on every model we tested while the best remedy for accuracy is model-dependent (§7). (4) Control experiments separate the plan from separate worker contexts, bound the decomposition, and build dependency clusters from held-out tasks (§7.2), and we test how far the best policy can be predicted from task structure (§6).

2 Related work

Tool retrieval over large tool spaces. Early systems pair a retriever with a tool-calling model to scale to thousands of APIs [10, 11] or encode each tool as a learned token [12]. MCP-era work retrieves tool documents by dense similarity [3, 4, 13, 14], while completeness- and dependency-aware methods retrieve a compatible *set* rather than individually similar tools [5, 15]; dynamic top-k [16] and cloud gateways sustain recall over thousands of tools [17, 18], and benchmarks find retrievers weak, success retrieval-bounded [19], and *semantic collisions* more harmful than random distractors [20]. All of this targets a *single* agent's context, or, in Kamiński et al. [20], per-subtask worker calls whose count *emerges* from decomposition rather than being a controlled variable. Growing the tool space also degrades the machinery meant to supervise these agents: safety critics calibrated against a fixed snapshot lose violation-detection ability as tools are added and application domains expand [21], the same scaling axis we sweep, measured on a different outcome.

Single- vs. multi-agent systems. Multi-agent frameworks [22–24] decompose tasks across roles, and Anthropic reports a 90.2% improvement over a single-agent baseline [6]. Other work finds the opposite: a failure taxonomy over seven frameworks finds multi-agent gains "often minimal," dominated by inter-agent misalignment [7], and practitioners argue for single-threaded agents [8]. Most directly, Kim et al. [25] fit a predictive model of *when* scaling to multi-agent helps across 260 configurations, finding coordination gains shrink as the base model strengthens and that multi-agent systems incur *overhead on tool-heavy tasks*, the regime we study; their model predicts the architecture *type*, not how the tool space is *partitioned* across agents, nor does it use the dependency structure among required tools.

The gap. The two literatures remain separated: retrieval work [3, 16, 20] concentrates tools in one context and optimizes *which* tools to surface, while scaling work [25] decides *how many* agents to use but not how tools are divided among them. The closest joint work retrieves tools and agents over a knowledge graph [26], but does not compare partitioning against a single-agent full-toolset baseline nor contrast dynamic vs. clustered allocation. We are aware of no study that treats tool allocation across an orchestrator and subagents as a controlled variable, runs τ-bench in a multi-agent configuration, or ties the winning allocation to task structure and the *tool-dependency graph*.

3 Tool-allocation policies

Let $\mathcal{T}$ be the available tool space and q a user query handled under a policy document P. An allocation policy maps $(q, \mathcal{T})$ to one or more agents, each with a tool subset $\mathcal{T}_i \subseteq \mathcal{T}$. We study six policies, holding the base model M and the user simulator fixed. Four span the single/multi-agent and dynamic/static axes:

- **S-ALL**: one agent holding the full space, $\mathcal{T}_1 = \mathcal{T}$. This is the status-quo baseline.
- **S-RAG**: one agent receiving the top-k tools retrieved by dense similarity to q [3]. It retrieves once, from the user's opening message, before the task's details are known. It is a deliberately simple one-shot baseline.
- **M-NEIGHBOR**: an orchestrator splits q into subtasks, and each subagent receives the top-k *neighbor* tools retrieved for its subtask (dynamic allocation).
- **M-DOMAIN**: tools are clustered into domains ahead of time, and a router assigns each subtask to the subagent holding the matching cluster (static allocation).

Two more change one thing each, how tools are refreshed within one context and how clusters are formed:

- **S-DYN**: one agent that re-retrieves its top-k tools each turn, so it never carries the whole space. As the single-agent counterpart of M-NEIGHBOR's dynamic retrieval, it isolates "stop carrying unused schemas turn to turn" from "split the work across agents."
- **M-DEP**: subagents receive *dependency-aware* clusters, formed by co-invocation profiles from ground-truth trajectories (same cluster count as M-DOMAIN), so tools used together land in the same subagent; this isolates the *signal* (co-usage vs. text similarity) from granularity. It applies the dependency-/completeness-aware retrieval idea [5, 15] to *partitioning across subagents*. The clusters are built once from corpus-level co-invocation, not from the current task. That corpus includes the evaluated tasks; §7.2 also reports a variant built without them (Appendix A). In deployment the same co-invocation signal would come from logged tool calls instead of annotated trajectories.

4 Faithful orchestrator–worker implementation

A confounded multi-agent implementation would silently change the task interface (letting subagents talk to the user, or resetting the environment per subagent), so any measured difference would trace to the harness rather than the policy. We fix one faithful design and vary only tool allocation. All policies share the same τ-bench environment instance and user simulator. In the multi-agent policies an *orchestrator* is the sole party that reads and replies to the user, emitting the only `respond` actions, so the user-facing protocol is identical to the single-agent policies; subagents receive a natural-language subtask and return a summary via a `dispatch` meta-action that is not an environment step. Each subagent runs a bounded tool-calling loop against the *same* environment object using only its toolset $\mathcal{T}_i$, so tool calls mutate shared state as a single agent's would and the terminal database-state reward is computed identically; the multi-agent system gets no scoring advantage.

Every LLM call writes its prompt and completion tokens into one shared accumulator, so reliability per token charges the multi-agent policies for their full coordination overhead. Our serving setup exposes no prompt caching, so the main tables report *raw* tokens; §7.1 re-prices them with separate input and output prices and with a cached schema prefix. The base model, temperature, and per-agent step caps are fixed; apart from the orchestrator prompt that any multi-agent policy needs, only $\mathcal{T}_i$ and the number of agents change. All embedding-based policies draw from one shared dense retriever over the same index, so a dynamic-vs-static difference cannot be a retriever-quality artifact, and utility tools (`think`, `calculate`, `transfer_to_human_agents`) are force-included so retrieval cannot break the interface. Full identifiers, decoding settings, and clustering/partitioning procedures are in Appendix A.

5 Experimental design

Testbed. We use τ-bench [9], which pairs a tool-using agent with an LLM-simulated user under a policy document and scores the *final database state* against an annotated goal (deterministic; no dialogue judging). It provides two domains, RETAIL (16 native tools, 115 tasks) and AIRLINE (14 native tools, 50 tasks), and the pass^k metric.

Sweeping tool-space size. Native tool sets are small, so we enlarge the pool with *distractor* tools to sweep $|\mathcal{T}| \in \{\text{native}, 30, 60, 120\}$. The default regime draws distractors from other domains' tools plus renamed copies of those tools; a harder regime injects near-duplicate paraphrases of the native tools (confusable hard negatives) to stress selection under *semantic collisions* [20]. Distractors are never task-required, so the ground-truth tool set is known [2] and we measure both task success and *tool-selection precision/recall* against it.

Controls and statistics. The base model M and user-simulator model are fixed and reported with version. We decode at $T{=}0.6$ to supply the trial-to-trial variance pass^k requires, run four trials per task, and report pass^1 (success) and pass^4, where $\text{pass}^k = \mathbb{E}_{\text{task}}\left[\binom{c}{k}/\binom{n}{k}\right]$, n is trials run and c the number successful [9]. The unit of every significance test is the *task*, not the rollout. We average the four trials within each task and run an exact paired sign/permutation test over tasks, so trials add precision to each task's score but are never counted as independent samples. The main grid uses ten tasks per domain; the key comparison uses thirty (§7.2). Every policy uses the same prompts, with no per-policy tuning.

Metrics. (1) Task success (DB-state match); (2) pass^k reliability; (3) tool-selection precision/recall vs. ground truth; (4) token cost (prompt+completion) and dollar cost per rollout; (5) pass^1 **per 1,000 tokens**. For RQ5 we derive three task-structure features from the annotated ground-truth trajectory: required-tool-set size, trajectory length, and tool-dependency depth (the longest chain of calls that consume prior tools' outputs).

6 Research questions

We ask five questions. **RQ1:** does a large tool space hurt a single agent's accuracy, as the small-model literature reports [1], or only its cost? **RQ2:** does allocation beat a single agent, and from what size? **RQ3:** does dynamic retrieval beat static and dependency-aware clustering at equal granularity? **RQ4:** is the cost gap intrinsic to single agents, or removable without orchestration? **RQ5:** can the best policy be predicted from task structure, complementing the architecture-type predictor of Kim et al. [25]?

7 Experimental results

We run all six policies over the full matrix: both τ-bench domains, all four tool-space sizes $|\mathcal{T}| \in \{\text{native}, 30, 60, 120\}$, ten tasks per domain, four trials per task at $T{=}0.6$ ($\approx$960 rollouts per domain), plus three robustness conditions: two further open base models (Kimi-K2.5, Qwen3-235B, on a three-policy subset), a weaker open base model (Ministral-8B agent, user simulator fixed on DeepSeek-V3.1), and a semantic-collision distractor regime. Table 1 reports the RETAIL main grid. At 120 tools M-NEIGHBOR matches S-ALL on success (.70 vs. .68, $p{=}1.0$) with .77 recall, while S-ALL inflates token cost $\sim 4\times$ native$\rightarrow$120 with no accuracy gain. M-DOMAIN and M-DEP top the reliability-per-token column (11.7 and 10.9 vs. M-NEIGHBOR's 6.0) because their failures are cheap. Their tool recall drops to .10 and .18, so they lose required tools and stop early.

RQ1: accuracy and cost of the all-tools agent. On RETAIL the accuracy of S-ALL is statistically flat across the sweep (.65 $\rightarrow$ .68 native$\rightarrow$120; paired test $p{=}1.0$) while its token cost rises 111k$\rightarrow$439k ($3.9\times$). On the harder AIRLINE domain accuracy is low throughout (.28 $\rightarrow$ .23) and cost still balloons ($115k \rightarrow 536k$). The extra cost comes from re-sending every tool schema on every call. Each S-ALL call carries about 25k schema tokens at 120 tools (2.9k at the native size), so at 120 tools its reliability per token is the worst of all six policies (1.5×10^{-3}), about $4\times$ below its own native value. At 120 tools M-NEIGHBOR and S-ALL are equally accurate (.70 vs. .68, $p{=}1.0$), and M-NEIGHBOR uses

Table 1: Main results on τ-bench RETAIL (base model and user simulator both DEEPSEEK-V3.1, fixed; ten tasks, four trials/task, T=0.6). Each cell shows *native / 120* tools, except "P/R@120", which is precision / recall, both at 120 tools. "raw tok" is mean total (prompt+completion) tokens per rollout across the whole system, with no prompt caching (§7.1 re-prices it). Precision and recall are measured against the ground-truth tool set. The last column is pass^1 per 1,000 tokens ($\times 10^{-3}$).

Policy	$\lvert\mathcal{T}\rvert$	success	pass^4	P/R@120	raw tok	$\frac{\mathrm{pass}^1}{\text{1k tok}}$
S-ALL	native/120	.65/.68	.20/.50	.82/.99	111k/439k	5.8/1.5
S-RAG	native/120	.35/.25	.10/.10	.26/.22	63k/43k	5.6/5.9
S-DYN	native/120	.83/.53	.60/.20	.63/.81	98k/93k	8.4/5.7
M-NEIGHBOR	native/120	.80/.70	.40/.30	.70/.77	136k/117k	5.9/6.0
M-DOMAIN	native/120	.70/.38	.40/.00	.24/.10	146k/32k	4.8/11.7
M-DEP	native/120	.80/.58	.50/.40	.38/.18	154k/53k	5.2/10.9

Table 2: Cross-model generalization on both domains (three-policy subset: S-ALL, S-DYN, M-NEIGHBOR; user simulator fixed on DeepSeek-V3.1 so only the agent model changes; ten tasks, four trials/task, T=0.6). Each cell is *native / 120* tools; **bold** marks the most accurate policy at 120 tools within each model and domain. Some native cells have three trials after dropping transient serving errors; paired tests are in the text.

		success (native/120)		raw tok (native/120)	
Model	Policy	retail	airline	retail	airline
DeepSeek-V3.1	S-ALL	.65/.68	.28/.23	111k/439k	115k/536k
	S-DYN	.83/.53	.43/.10	98k/93k	108k/132k
	M-NEIGHBOR	.80/**.70**	.28/**.58**	136k/117k	217k/194k
Kimi-K2.5	S-ALL	.76/.85	.38/.40	91k/314k	127k/404k
	S-DYN	.80/.65	.40/.20	85k/94k	118k/108k
	M-NEIGHBOR	.80/**.88**	.39/**.45**	118k/106k	176k/155k
Qwen3-235B	S-ALL	.77/.60	.13/.15	111k/466k	144k/574k
	S-DYN	.73/**.73**	.13/**.20**	90k/106k	140k/141k
	M-NEIGHBOR	.13/.25	.05/.13	116k/138k	240k/207k

about one-quarter of the raw tokens (117k vs. 439k). S-ALL is the more consistent of the two across trials (pass^4 .50 vs. .30: all four trials succeed on five tasks for S-ALL and three for M-NEIGHBOR).

RQ4: a dynamic single agent removes the cost gap but not the accuracy loss. Partly. S-DYN keeps RETAIL token cost flat ($98k \to 93k$), but its RETAIL success falls from .83 to .53 as the space grows. At 120 tools on RETAIL M-NEIGHBOR leads S-DYN on accuracy (.70 vs. .53), not significantly on the ten-task grid (p=0.28), and the two tie on reliability per token (6.0 vs. 5.7×10^{-3}, p=0.58). Over thirty tasks the accuracy gap is significant (.68 vs. .50, p=0.021; §7.2) at 1.3× S-DYN's tokens. The larger gap is on AIRLINE, where S-DYN collapses (.43→.10) while the multi-agent policies reach .50–.58 at 120 tools, a paired advantage of +.48 for M-NEIGHBOR (p=0.016), and +.39 over thirty tasks (p<0.001).

RQ2: where splitting overtakes the all-tools agent. On RETAIL, M-NEIGHBOR's success is statistically indistinguishable from S-ALL at every size (p>0.25), while the token comparison flips: M-NEIGHBOR costs 1.2× more at the native size (136k vs. 111k), about the same at 30 tools, and 2.0× and 3.7× less at 60 and 120 tools (p=0.002 each). On AIRLINE, M-NEIGHBOR is more accurate than S-ALL at every non-native size (.45–.58 vs. .13–.23), significantly at 30 and 60 tools (p=0.016, 0.008) and not at 120 (p=0.078), and 2.8× cheaper at 120 tools. The reliability-per-token ratio summarizes the same crossover ($+4.5 \times 10^{-3}$ at 120 tools on RETAIL).

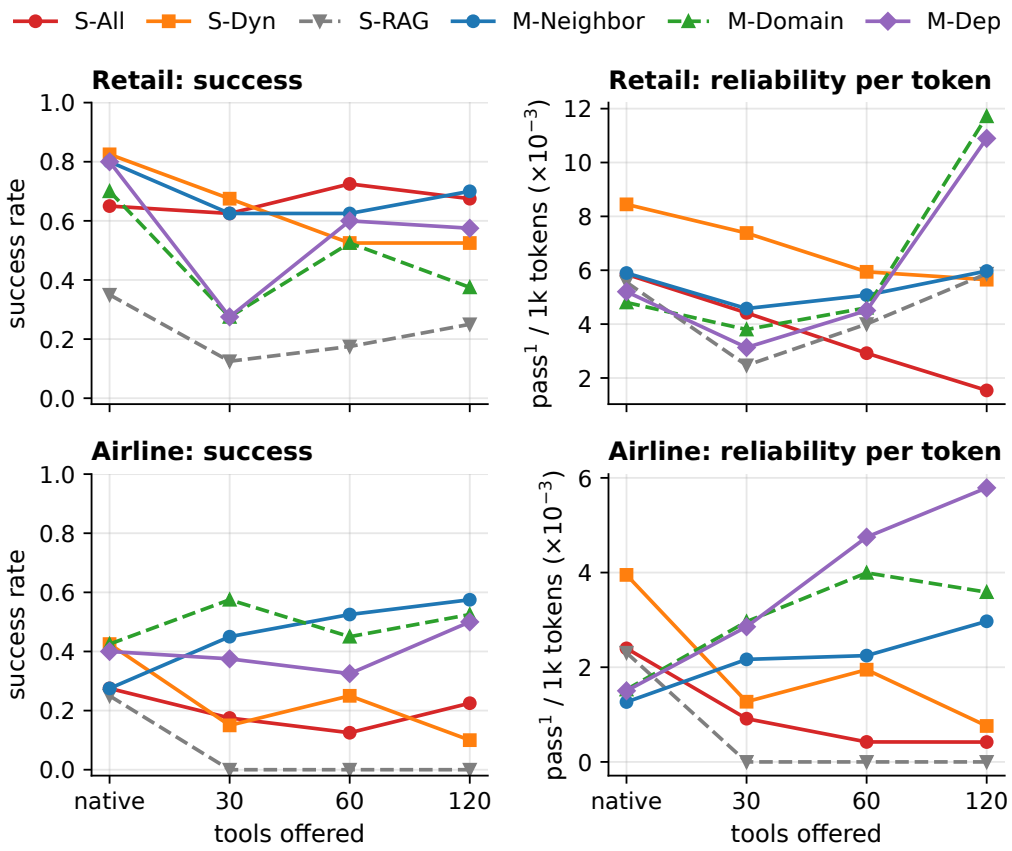


Figure 1: RETAIL (top) and AIRLINE (bottom); success (left) and pass^1 per 1,000 tokens (right) vs. tool-space size (native = 16 tools on RETAIL, 14 on AIRLINE), one line per policy. S-ALL (red) holds accuracy roughly flat while its reliability per token collapses; on RETAIL, M-DOMAIN/M-DEP score high on the ratio because their failed rollouts are cheap (low recall, Table 1).

RQ3 (DeepSeek-V3.1): dynamic per-subtask retrieval beats both static clusterings. (At the native size M-DOMAIN and M-DEP degenerate to one all-tools cluster, so their meaningful range begins above it.) At 120 tools on RETAIL, M-NEIGHBOR holds tool recall at .77 and success at .70, whereas the static clusterings shed required tools as distractors crowd in: recall drops to .18 (M-DEP) and .10 (M-DOMAIN), success .58 and .38. Dependency-aware clustering beats similarity clustering on RETAIL accuracy, but neither matches query-conditioned retrieval, and the ordering does *not* carry to AIRLINE (Table 5, Appendix B): there M-NEIGHBOR is still highest (.58) but the static clusterings reverse (M-DOMAIN .53 > M-DEP .50). AIRLINE tasks are structurally uniform (most use the same few tools), so the corpus co-invocation matrix is nearly all-to-all and M-DEP's fingerprints barely separate. The clustering policies ran on one model. The success ordering (query-conditioned retrieval above static clusters) also holds under semantic collisions (below).

Finally, S-RAG, a single top-k retrieval from one flat task query, is the weakest policy everywhere (success .35 → .25 on RETAIL, 0.0 at every non-native size on AIRLINE and under semantic collisions). M-NEIGHBOR and S-DYN use the same retriever with a fresh subtask- or per-turn query and do not share the failure, which points to the one-shot query as the cause.

RQ5: the best policy is partly predictable from task structure. We correlate, per task, the single- vs. multi-agent gap (S-DYN minus M-NEIGHBOR at $|\mathcal{T}|$=120) with required-tool-set size, trajectory length, and tool-dependency depth (Spearman, permutation test). This analysis is exploratory (ten tasks, several features and outcomes, no multiple-comparison correction). On RETAIL the *success* gap is consistent with task structure: it correlates negatively with trajectory length ($\rho = -0.68$, p=0.033) and required-tool-set size ($\rho = -0.62$, p=0.059), so longer, tool-heavier tasks favor multi-agent

routing and shorter ones the single dynamic agent (the mean gap over all ten tasks is -0.18). Both features trend the same way on the reliability-per-token gap but non-significantly ($\rho = -0.52$ and -0.49, p=0.12 and 0.15). On AIRLINE the tasks are too structurally uniform for a correlation, but the mean success gap is strongly negative (-0.48), consistent with multi-agent routing dominating the harder domain.

Robustness: weaker base model and semantic distractors. With the weaker Ministral-8B agent (user simulator held fixed) the S-ALL cost balloon is unchanged ($134k \to 553k$, $4.1\times$, flat accuracy) and M-NEIGHBOR again leads at 120 tools (.73, default regime). Under semantic collisions [20] S-RAG falls to 0.0 while the query-conditioned S-DYN and M-NEIGHBOR stay in the .44–.63 range and the static clusterings shed required tools (M-DOMAIN .35, M-DEP .05 at 60 tools).

Cross-model generalization. We repeat a three-policy subset (S-ALL, S-DYN, M-NEIGHBOR at native and 120 tools, user simulator fixed on DeepSeek-V3.1) on Kimi-K2.5 and Qwen3-235B (Table 2). The *cost* result holds on every model we tested: all three inflate S-ALL tokens 3.2–4.7× native→120 in both domains with no reliable accuracy change (Kimi's retail accuracy rises .76 → .85 and Qwen's falls .77 → .60, neither significant, p=0.38, 0.22). The best *remedy* is model-dependent. For DeepSeek-V3.1 and Kimi-K2.5, M-NEIGHBOR is most accurate at 120 tools (.70, .88). Its advantage over S-DYN is significant for Kimi (+.23, p=0.031) and its edge over S-ALL within noise. Qwen3-235B diverges: its multi-agent path collapses (M-NEIGHBOR .13–.25, worst at both retail sizes despite ∼2× the calls of S-ALL), while S-DYN holds accuracy at near-constant cost (90k→106k) and beats M-NEIGHBOR (+.48, p=0.008). No call errored. The failure is in coordination: Qwen over-decomposes and omits the terminal state-mutating write in 25 of 30 retail failures, against 2 of 11 failures under S-DYN (Appendix C; a bounded-decomposition control is in §7.2). AIRLINE agrees: M-NEIGHBOR leads for DeepSeek-V3.1 (.58) and Kimi (.45) while their S-DYN degrades on the harder domain (Kimi +.25, p=0.078), and Qwen is uniformly weak (≤ .20).

Scaling to 400 tools: the benchmark does not stress the model at this size. We extend RETAIL to $|\mathcal{T}| \in \{120, 250, 400\}$ with diverse cross-domain distractors for DeepSeek-V3.1 (strong) and Ministral-8B (weak), ten tasks, three seeds (Table 6, Appendix B). Its distractor pool and seeds differ from the main grid, so its 120-tool column is a separate run.[1] S-ALL accuracy does not fall out to 400 tools (DeepSeek-V3.1 .70 → .80, Ministral-8B .57 → .50; neither change significant, p=0.38, 0.63). One reading is that the benchmark does not get harder for the model as tools are added. Another is that these distractors are easy negatives: no distractor lies within cosine distance 0.10 of a native tool, so this sweep grows the *number* of distractors, while confusable near-duplicates were tested only up to 120 tools (semantic regime). The raw cost balloon grows to 0.9–1.3M S-ALL tokens while retrieval policies stay near 100k, so at 400 tools the best reliability per token belongs to a retrieval policy for both models (S-DYN for DeepSeek-V3.1, M-NEIGHBOR for Ministral-8B), 7–18× that of S-ALL. For the weak model the split's payoff *grows* with scale: Ministral S-DYN degrades .53 → .27 (p=0.031) while M-NEIGHBOR leads it by +.50 at 400 (p=0.008).

7.1 Cost under real prices and prompt caching

Raw tokens overstate what a deployment pays when the provider caches a repeated prompt prefix, and they ignore that output tokens cost more than input tokens. Our serving setup exposed neither, so we re-price the logged prompt and completion tokens of every rollout (Table 3). Dollar cost uses each model's on-demand input and output prices on Amazon Bedrock (DeepSeek-V3.1: \$0.58 and \$1.68 per million tokens). For S-ALL we then cache its static prefix, the system prompt plus every tool schema (about 24k tokens at 120 tools and 57k at 400): the first call writes it at 1.25× the input price and every later call reads it at 0.1×. We give caching only to S-ALL, the most favorable case for the baseline: S-DYN changes its tool list every turn, and the tool list precedes the conversation in the prompt, so its prefix does not stay fixed.

Prices change no ranking (prompt tokens dominate every policy), but caching changes the size of the gap. At 120 tools a cached S-ALL costs \$0.065 per RETAIL rollout, 4× less than uncached (\$0.256) and close to S-DYN (\$0.056). On AIRLINE it is \$0.085 vs. \$0.079. At 400 tools caching

[1] DeepSeek-V3.1 M-NEIGHBOR scores .43 at 120 tools here and .70 in Table 1, a difference that is not significant when paired by task (p=0.12), with the same tool recall (.74 vs. .77). Ministral-8B M-NEIGHBOR scores .57 here and .73 in the default regime.

Table 3: Dollar cost per rollout (Bedrock on-demand input/output prices, DeepSeek-V3.1) re-priced from logged tokens. "cached" gives S-ALL a cached schema prefix (write $1.25\times$, read $0.1\times$); the other policies are left uncached. Success in parentheses.

Policy	RETAIL@120	AIRLINE@120	RETAIL@400 (diverse)
S-ALL, uncached	\$0.256 (.68)	\$0.314 (.23)	\$0.533 (.80)
S-ALL, cached	\$0.065 (.68)	\$0.085 (.23)	\$0.096 (.80)
S-DYN	\$0.056 (.53)	\$0.079 (.10)	\$0.055 (.60)
M-NEIGHBOR	\$0.071 (.70)	\$0.117 (.58)	\$0.063 (.60)

still leaves S-ALL $1.7\times$ more expensive than S-DYN (\$0.096 vs. \$0.055; $2.0\times$ for Ministral-8B), because even a discounted read of a 57k-token prefix on every call adds up. So the gap comes from re-sending schemas, and either caching or per-turn retrieval removes most of it. Caching does not change accuracy, so the accuracy comparison below is unaffected.

7.2 Control experiments

More tasks. The comparison our main claims rest on, S-DYN vs. M-NEIGHBOR at 120 tools, uses thirty tasks per domain (tasks 0–29). On RETAIL success is .68 for M-NEIGHBOR vs. .50 for S-DYN ($+.18$, $p{=}0.021$), and on AIRLINE .67 vs. .28 ($+.39$, $p{<}0.001$). The 20 tasks outside the main grid show the same retail gap ($+.19$). The split costs more tokens on both domains (114k vs. 86k and 156k vs. 103k).

Plan or separate workers? M-NEIGHBOR bundles three things: an explicit plan, a retrieval query per subtask, and a fresh context per worker. M-SHARED keeps the first two and removes the third: the orchestrator plans and dispatches as before, and each subtask gets the same retrieved tools, but one worker executes every subtask in a single persistent context. Its accuracy matches M-NEIGHBOR on both domains (retail .80 vs. .70, $p{=}0.56$; airline .48 vs. .58, $p{=}0.61$), and like it, it beats S-DYN on AIRLINE ($p{=}0.016$). It costs 1.6–$1.7\times$ M-NEIGHBOR's tokens, because the shared context carries every earlier subtask's tool traffic. The accuracy gain comes from explicit decomposition with a retrieval query per subtask, and separate worker contexts mainly keep the cost down.

Bounded decomposition. M-BOUND is M-NEIGHBOR with the orchestrator told it may use at most three subagents, each with a complete multi-step subtask including the final write, and a hard cap at three. For Qwen3-235B the cap works as intended: subagents per rollout fall from 7.4 to 3.0 and tokens from 138k to 97k on RETAIL at 120 tools. Accuracy does not recover (.18 vs. .25, $p{=}0.53$), it stays far below S-DYN (.73, $p{=}0.004$), and the terminal write is still missing in 29 of 33 failures. Over-decomposition is a symptom. Qwen fails at turning subagent results into the final write. For DeepSeek-V3.1 the cap hurts (retail .40 vs. .70, $p{=}0.065$; airline .28 vs. .58, $p{=}0.031$).

Held-out dependency clusters. M-DEPHO builds M-DEP's co-invocation corpus with the ten evaluated tasks removed. On RETAIL the held-out partition is identical to the full-corpus one at 60 and 120 tools (and agrees on 94% of tool pairs at 30), and success at 120 tools is unchanged (.58 vs. .58). On AIRLINE, where the held-out corpus loses a quarter of its trajectories, held-out success is no lower at any size (.68 vs. .50 at 120). The 120-tool airline partition is nearly unchanged (98% pair agreement), so we read that increase as run-to-run variance. Leakage does not explain M-DEP's results.

8 Conclusion and limitations

For a capable model, a large tool space mainly raises cost. Giving one agent every schema leaves success statistically flat and inflates raw token cost about fourfold (RQ1). Per-turn retrieval removes that cost within one context, and caching the schema prefix removes most of it, leaving a $1.7\times$ gap to S-DYN at 400 tools.

The cost effect is the same on every model we tested, but the best way to recover accuracy differs. A strong coordinator (DeepSeek-V3.1, Kimi-K2.5) gains accuracy by splitting tools per subtask, most clearly on the harder airline domain (for DeepSeek-V3.1 on both domains over thirty tasks). The gain

Table 4: Control experiments at 120 tools (DeepSeek-V3.1 unless noted; four trials per task). Success and raw tokens per rollout; "sub." is mean subagents per rollout on RETAIL. Paired task-level tests are given in the text.

		RETAIL		AIRLINE	
Policy	sub.	success	tok	success	tok
Thirty tasks per domain					
S-DYN	–	.50	86k	.28	103k
M-NEIGHBOR	–	.68	114k	.67	156k
Tasks 0–9					
M-NEIGHBOR	5.6	.70	117k	.58	194k
M-SHARED	6.5	.80	196k	.48	309k
M-BOUND	2.9	.40	85k	.28	99k
M-DEP	3.1	.58	53k	.50	86k
M-DEPHO	2.8	.58	50k	.68	70k
Qwen3-235B, tasks 0–9, RETAIL *only*					
S-DYN	–	.73	106k		
M-NEIGHBOR	7.4	.25	138k		
M-BOUND	3.0	.18	97k		

comes from the plan and its per-subtask retrieval, and separate worker contexts keep the cost down. A weaker coordinator (Qwen3-235B) does better as a single dynamic agent. Its multi-agent path drops the terminal state-mutating write, and capping its decomposition does not fix that. In practice, a large tool list should not sit uncached in one context, and subagents are worth their overhead on hard, multi-step tasks when the base model coordinates them well.

Limitations and future work. *(i) Coverage.* One benchmark, two domains, and open-weight models only; most cells use ten tasks (thirty for the key comparison); the full six-policy set ran on DeepSeek-V3.1 alone, so RQ3 is a one-model finding, and transfer to proprietary models is untested. *(ii) Realism.* Synthetic and cross-domain distractors lack the overlapping functionality, ambiguous descriptions, failure modes, and evolving schemas of production registries [17]; the 400-tool plateau may mean the benchmark does not stress the model at this size, or that the distractors are easy negatives, since confusable distractors were tested only to 120 tools. Harder registries, such as organic catalogs like MCP-Flow [18] or injected tool failures, would test this directly. *(iii) Baselines.* We do not test cross-encoder rerankers, learned routers, dependency-aware retrieval for a single agent [5], or a tuned top-k (we fix k=12), and M-NEIGHBOR itself could cache its subagent prefixes. Allocation is also only the prompt-side cost lever: task-specific fine-tuning of a smaller model reduces serving cost on a separate axis [27] and composes with it. *(iv) Cost.* Caching is modeled from logged tokens, not measured, and neither tokens nor dollars capture latency, rate limits, or the wall-clock cost of sequential orchestration. *(v) Simulator.* The LLM user could move the crossover. Prompt optimization [28] and dual-control (τ^2-bench [29]) are further future work.

Responsible-use statement

This is a controlled efficiency study of how tools are allocated across an orchestrator and its subagents; its most direct effect is on the compute cost of tool-using agents. Cutting the repeated schema tokens we measure lowers the energy cost of deployed agents, and the study uses only open-weight models and a public benchmark. Cheaper orchestration also carries the usual dual-use risk of agent research: tool-calling agents take actions with real side effects, and lower cost can speed deployment before reliability is understood. That risk compounds because the guardrails meant to catch it degrade on the same axis: critics calibrated on a smaller tool environment silently lose detection ability as it grows [21]. Our faithful orchestrator–worker design keeps a single user-facing party and logs every subagent tool call into one trace, so decomposition does not dissolve the audit trail; we score against a deterministic database state rather than a dialogue judge; and we caution that reliability per token must never be optimized alone, since it rewards agents that exit early. Query-conditioned retrieval is also a security surface (adversarial or near-duplicate tool descriptions can mislead selection), so

retrieval indices over third-party registries should be curated, and irreversible writes should keep a human approval step.

References

[1] Kiran Kate, Tejaswini Pedapati, Kinjal Basu, Yara Rizk, Vijil Chenthamarakshan, Subhajit Chaudhury, Mayank Agarwal, and Ibrahim Abdelaziz. LongFuncEval: Measuring the effectiveness of long context models for function calling. *arXiv preprint arXiv:2505.10570*, 2025.

[2] Seongjin Yeon, Isha Chaudhary, and Gagandeep Singh. Quantitative certification of agentic tool selection. *arXiv preprint arXiv:2510.03992*, 2025.

[3] Tiantian Gan and Qiyao Sun. RAG-MCP: Mitigating prompt bloat in LLM tool selection via retrieval-augmented generation. *arXiv preprint arXiv:2505.03275*, 2025.

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

[5] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Towards completeness-oriented tool retrieval for large language models. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, 2024.

[6] Anthropic. How we built our multi-agent research system. `https://www.anthropic.com/engineering/multi-agent-research-system`, 2025.

[7] Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

[8] Walden Yan. Don't build multi-agents. `https://cognition.ai/blog/dont-build-multi-agents`, 2025.

[9] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

[10] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, et al. ToolLLM: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

[11] 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.

[12] Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. ToolkenGPT: Augmenting frozen language models with massive tools via tool embeddings. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[13] Elias Lumer, Vamse Kumar Subbiah, James A. Burke, Pradeep Honaganahalli Basavaraju, and Austin Huber. Toolshed: Scale tool-equipped agents with advanced rag-tool fusion and tool knowledge bases. *arXiv preprint arXiv:2410.14594*, 2024.

[14] Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, et al. Re-Invoke: Tool invocation rewriting for zero-shot tool retrieval. In *Findings of the Association for Computational Linguistics: EMNLP*, 2024.

[15] Bhrij Patel, Davide Belli, Amir Jalalirad, Maximilian Arnold, Aleksandr Ermolov, and Bence Major. Dynamic tool dependency retrieval for lightweight function calling. *arXiv preprint arXiv:2512.17052*, 2025.

[16] Zhengzong Chen, Lei Tang, Lijun Liu, et al. MagicSelector: Joint optimization for agent tool selection via counterfactual decomposition and progressive reranking. *arXiv preprint arXiv:2607.17751*, 2026.

[17] Mingxin Li, Enge Song, Yueshang Zuo, Xiaodong Liu, Rong Wen, Qiang Fu, Gianni Antichi, Jian He, Shunmin Zhu, et al. Scalable LLM agent tool access in the cloud. *arXiv preprint arXiv:2607.15593*, 2026.

[18] Wenhao Wang, Peizhi Niu, Zhao Xu, Zhaoyu Chen, Jian Du, Yaxin Du, Xianghe Pang, Keduan Huang, Yanfeng Wang, Qiang Yan, and Siheng Chen. MCP-Flow: Facilitating LLM agents to master real-world, diverse and scaling MCP tools. *arXiv preprint arXiv:2510.24284*, 2026. ACL 2026 Main.

[19] Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. *arXiv preprint arXiv:2503.01763*, 2025.

[20] Jerzy Kamiński, Ilya Galyukshev, Sergey Chuprin, Artem Kuznetsov, and Anna Kalyuzhnaya. Too many tools, too much confusion? navigating agentic tool selection at scale. *Algorithms*, 19(6):447, 2026. doi: 10.3390/a19060447.

[21] Fatemeh Taheri Dezaki, Lakshman Kolasani, Ankita Bhaumik, Siddartha Sathyamoorthy Rao, and Ayush Jain. When critics go stale: Benchmarking safety guardrail degradation under environment drift. In *Proceedings of the Second Workshop for Research on Agent Language Models (REALM)*, pages 922–929. Association for Computational Linguistics, 2026.

[22] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

[23] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, et al. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024.

[24] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[25] Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, Mark Malhotra, Paul Pu Liang, Hae Won Park, Yuzhe Yang, Xuhai Xu, Yilun Du, Shwetak Patel, Tim Althoff, Daniel McDuff, and Xin Liu. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025.

[26] Faheem Nizar, Elias Lumer, Anmol Gulati, Pradeep Honaganahalli Basavaraju, and Vamse Kumar Subbiah. Agent-as-a-graph: Knowledge graph-based tool and agent retrieval for LLM multi-agent systems. *arXiv preprint arXiv:2511.18194*, 2025.

[27] Lakshman Kolasani and Fatemeh Taheri Dezaki. Progressive fine-tuning for cost-effective structured attribute generation in e-commerce. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 585–591. Association for Computational Linguistics, 2026.

[28] Sandip Ghoshal, Anshul Mittal, Jyotika Singh, Miguel Ballesteros, Weiyi Sun, Fang Tu, Shailender Singh, Yassine Benajiba, Fahad Shah, Sujeeth Bharadwaj, Sujith Ravi, and Dan Roth. JTPRO: A joint tool-prompt reflective optimization framework for language agents. *arXiv preprint arXiv:2604.19821*, 2026. ACL 2026.

[29] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

A Reproducibility details

All runs share one harness and configuration; only the policy under test changes.

Models and decoding. Agent, orchestrator, and subagents use DeepSeek-V3.1, and so does the user simulator. The weaker-model and cross-model runs swap only the agent model and keep the user simulator on DeepSeek-V3.1. Retrieval uses all-MiniLM-L6-v2 run locally with cosine similarity. Every call uses $T{=}0.6$; the single agent and the orchestrator are capped at 30 steps per rollout and each subagent at 12; the retrieval budget is $k{=}12$ for S-RAG, S-DYN, and each M-NEIGHBOR subagent, above the largest ground-truth tool set (six tools).

Tool spaces. The default regime adds real tools from other τ-bench domains, then renamed copies of them (suffix _vn, description prefixed "(variant n)"), which embed close to other-domain tools, not native ones. The semantic-collision regime adds paraphrased near-duplicates of the native tools. The diverse regime (Table 6) adds real cross-domain tools, then τ^2-bench telecom tools, then well-separated synthetic tools. Distractors are seeded identically (seed$=$0) for all policies and trials. The utility tools `think`, `calculate`, and `transfer_to_human_agents` are always included.

Clustering policies. M-DOMAIN splits the offered tools into $k_c = \max(1, \mathrm{round}(|\mathcal{T}|/16))$ clusters by k-means on description embeddings, so the native size is a single cluster. M-DEP uses the same k_c but clusters tools by co-invocation: each tool's L2-normalized row of the co-invocation matrix over all ground-truth trajectories that call at least one tool (113 of 115 RETAIL, 43 of 50 AIRLINE; the rest only reply to the user) is clustered by cosine k-means, and distractors join the group of their embedding-nearest corpus tool. M-DEPHO drops the ten evaluated tasks from that corpus (103 and 33 trajectories). Both route a subtask to the nearest group centroid.

Control policies. M-SHARED keeps M-NEIGHBOR's orchestrator and per-subtask retrieval but runs every subtask in one persistent worker conversation. M-BOUND tells the orchestrator to use at most three subagents and rejects further dispatches.

Tokens and cost. Raw tokens sum prompt and completion tokens over every call by every agent in a rollout. Dollar cost uses Bedrock on-demand prices per million input/output tokens (DeepSeek-V3.1 \$0.58/\$1.68, Kimi-K2.5 \$0.60/\$3.00, Qwen3-235B \$0.22/\$0.88, Ministral-8B \$0.15/\$0.15). For cached S-ALL with n calls and static prefix P (system prompt plus tool schemas), billed input is prompt $- 0.9(n-1)P + 0.25P$.

Statistics. Comparisons are paired by task: trials are averaged per task and a two-sided sign-flip permutation test is applied to the per-task differences (exact up to 20 non-zero differences, else 2×10^5 random flips). Effects are mean per-task differences; there is no multiple-comparison correction. RQ5 uses Spearman correlation with a 10^4-shuffle permutation test.

B Additional tables

Table 5: Main results on τ-bench AIRLINE, the counterpart of Table 1 (DeepSeek-V3.1 agent and user simulator; ten tasks, four trials/task, T=0.6). Each cell shows *native / 120* tools; native = 14 tools.

Policy	success	pass^4	raw tok	$\frac{\mathrm{pass}^1}{\text{1k tok}}$
S-ALL	.28/.23	.10/.10	115k/536k	2.4/0.4
S-RAG	.25/.00	.00/.00	108k/39k	2.3/0.0
S-DYN	.43/.10	.10/.00	108k/132k	4.0/0.8
M-NEIGHBOR	.28/.58	.10/.10	217k/194k	1.3/3.0
M-DOMAIN	.43/.53	.00/.20	278k/146k	1.5/3.6
M-DEP	.40/.50	.20/.10	266k/86k	1.5/5.8

Table 6: Scaling RETAIL to 400 tools (diverse distractors; ten tasks, three seeds, T=0.6). Each cell is success / raw tok / pass^1 per 1,000 tokens ($\times 10^{-3}$). Six of the 90 Ministral M-NEIGHBOR rollouts hit transient serving errors and were dropped.

Model	Policy	120	250	400
DeepSeek-V3.1	S-ALL	.70/315k/2.2	.77/594k/1.3	.80/915k/0.9
	S-DYN	.57/98k/5.8	.47/91k/5.1	.60/92k/6.5
	M-NEIGHBOR	.43/112k/3.9	.70/98k/7.1	.60/104k/5.8
Ministral-8B	S-ALL	.57/371k/1.5	.53/811k/0.7	.50/1.27M/0.4
	S-DYN	.53/129k/4.1	.43/130k/3.3	.27/130k/2.0
	M-NEIGHBOR	.57/111k/5.1	.68/102k/6.7	.77/108k/7.1

C Why Qwen3-235B collapses under M-NEIGHBOR

We inspected all 40 Qwen M-NEIGHBOR RETAIL rollouts at 120 tools (30 failures). None logged an error. As orchestrator, Qwen spawned 7.4 subagents on average (up to 20) for tasks needing

about four tools, and failed rollouts used 34 LLM calls for only about 9 environment steps. The subagents ran the lookups, but the final state-changing action the reward checks (for example `exchange_delivered_order_items`) was missing in 25 of the 30 failures, against 2 of 11 failures for the same model under S-DYN. Capping the orchestrator at three subagents (§7.2) leaves this unchanged (29 of 33 failures).