THE MISSING LINK: RELATIONAL TOOL RETRIEVAL FOR MORE CAPABLE AND CONSISTENT AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in tool-calling agents have focused on expanding capability through stronger foundation models and larger tool catalogs. Enterprise environments provide a natural setting for this challenge, with knowledge distributed across large, evolving benchmarks of tools and resources. Yet the utility of a tool ecosystem depends not only on the tools it contains, but also on the relationships connecting them. We present GEAR (Graph-Expanded Agent Retrieval), a retrieval framework that uses an ontology to link tools, concepts, and execution contexts. GEAR was designed to avoid both the fine-tuning required by COLT and the additional large language model (LLM) calls that MCP-Zero incurs at inference time. Our experiments on CAR-bench demonstrate that GEAR outperforms both baselines in capability while maintaining comparable consistency and consuming fewer tokens than MCP-Zero.

1 INTRODUCTION

Consider a simple request to an in-car agent: "Take me to a charging station on the way home." To fulfil it, the agent must call several tools in turn: it searches for a charging station, retrieves that station's location, and plans a route to it. A domain policy adds one constraint, namely that a place must be looked up before it can be set as a destination (Kirmayr et al., 2026). Issued three times, the same request can yield three different outcomes. Some requests fail on every attempt; others succeed once and then fail; in the worst case the agent reports, incorrectly, that the vehicle cannot navigate at all. Such unreliability is costly to users, in wasted time and effort, and to the providers that deploy these systems, in lost trust and revenue (Yao et al., 2025; Kirmayr et al., 2026).

The cause is structural rather than a failure of reasoning. An *agent* of this kind does not reason over all of its abilities at once. Each ability is exposed as a *tool*, a small external function the agent can call, and the complete set of tools, the *catalogue*, is typically larger than the model's *context window*. Only a short subset, the *shortlist*, can therefore be placed before the agent on any given turn (Repantis et al., 2026). When the shortlist omits the single tool a task requires, the task becomes unsolvable, however capable the underlying model. In the example above, the routing tool exists in the catalogue but is never surfaced.

This difficulty is not specific to the automotive setting; it arises whenever an agent must select from a large benchmark. In the ToolBench benchmark, for instance, booking a flight requires locating the correct tool among more than sixteen thousand candidates (Qin et al., 2024). A consistent finding across benchmarks is that the dominant obstacle is tool selection rather than tool use, and that success rates fall sharply as the catalogue grows (Li et al., 2023; Barres et al., 2025).

Selection is thus governed by whatever decides which tools the agent is shown, and this paper studies that decision along two axes. The first is *capability*: how many tasks a method allows the agent to complete. The second is *consistency*: whether those tasks are completed on every repeated attempt. Capability is measured by Pass@k, under which a task counts as solved if the agent succeeds on at least one of k attempts, and consistency by the stricter Pass^k, which requires success on all k attempts.

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.

Our approach follows from a single observation: many tasks depend on tools the user never names, and many tools carry policies the agent must respect (Kirmayr et al., 2026; Winston et al., 2026). GEAR (Graph-Expanded Agent Retrieval) addresses both by equipping retrieval with an *ontology*, a graph whose nodes are tools and whose labelled edges record how those tools relate (Gruber, 1993). Starting from the tools that ordinary similarity search already finds, GEAR traverses these edges to draw the related tools into the shortlist, even when their own descriptions match nothing that was said (Figure 1). GEAR is evaluated against two established methods, COLT (Qu et al., 2024) and MCP-Zero (Fei et al., 2025), and against an upper bound in which the agent is supplied the correct tools in advance, on two benchmarks, CAR-bench (Kirmayr et al., 2026) and ToolBench (Qin et al., 2024), with two language models, claude-haiku-4.5 (Anthropic, 2025) and gpt-5.4-nano (OpenAI, 2026).

The contributions of this paper are as follows.

1. **GEAR achieves the highest capability** on both language models, outperforming the other methods in a consistent order and, on one model, matching an agent supplied with the correct tools in advance (§5.1).
2. **GEAR is comparable in consistency to the alternatives**; moreover, every prompt-side addition we test degrades one metric rather than improving it (§5.3).
3. **GEAR's advantage widens with task difficulty**, as harder tasks depend more heavily on tools the request never names (§5.2).
4. **GEAR is the simplest to deploy**: it requires no training, no additional per-turn model call, and incurs a lower per-task cost than the method that issues tool requests at inference time (§5.5).

2 Related work

Recent surveys organise LLM tool learning around task planning, tool selection, tool calling, and response generation (Qu et al., 2025). Within this broader landscape, graph-augmented agents use relational structures to support planning, memory, tool management, and multi-agent coordination (Liu et al., 2025b). We focus specifically on graph-structured tool selection and study whether relational retrieval improves both capability and consistency across repeated trials. The remainder of this section positions GEAR among methods that determine which tools an agent sees.

Finding tools by similarity search. The most common approach is to score each tool by how much its description resembles the request, keeping the closest few. This can be done with *similarity search* (Robertson & Zaragoza, 2009; Karpukhin et al., 2020; Lewis et al., 2020). Some systems learn this matching from past examples, or even train the model to name the tool it wants directly (Qin et al., 2024; Patil et al., 2024; Schick et al., 2023; Wang et al., 2025). They all share one blind spot: resemblance surfaces only tools whose descriptions sound like the request, and misses tools that the task needs but nobody mentions.

Graph-structured retrieval and ontologies. A second family addresses this blind spot by linking tools that belong together and following those links, so that a tool's companions enter the shortlist even when their descriptions match nothing in the request. Such links may be authored by hand (Lumer et al., 2025), mined from data (Gao et al., 2025), or learned so that a complete tool set is retrieved together, as in COLT (Qu et al., 2024). GEAR belongs to this family: its map is a *knowledge graph* (Gruber, 1993), an established formalism for recording the entities of a domain and the labelled relations between them, surveyed comprehensively by Hogan et al. (2021) and Ji et al. (2022). Ontologies of this kind already exist for the in-car domain on which our first benchmark is based (Feld & Müller, 2011). Authoring them by hand does not scale, so recent work drafts them with language models, while noting that automatically constructed graphs are often noisy or incomplete (Babaei Giglou et al., 2023; Feng et al., 2024; Mo et al., 2025); GEAR therefore freezes its ontology before use (§7). The benefit of querying such structure is well documented: retrieval over a constructed graph outperforms retrieval over unstructured text on questions that require several hops (Edge et al., 2024; Jiménez Gutiérrez et al., 2024).

Prompting and planning. A parallel line of work adds structure at inference time rather than in the retriever. Reasoning-and-acting prompts interleave the agent's thoughts with its tool calls (Yao et al.,

2023; Wei et al., 2022; Wang et al., 2023), plan-first methods such as ReWOO decouple planning from execution to avoid repeated tool observations (Xu et al., 2023; Liu et al., 2025a), and policy compliance can be written directly into the prompt or checked by an external solver (Winston et al., 2026). A related runtime strategy is active discovery: once tools are exposed in enormous pools, even listing them can overflow the agent's context window (Anthropic, 2024), so MCP-Zero has the agent write a short request for the ability it is missing and resolves it against the catalogue, rather than ranking a fixed list (Fei et al., 2025). This literature assumes that added structure helps; we instead test, on a small backbone, when prompt-side additions such as visibility labels and policy encoding cost rather than improve performance (§5.3).

Benchmarks and metrics. Finally, how success is judged has been shifting from average accuracy towards accuracy and consistency across repeated tries, because a method that only succeeds sometimes is not yet trustworthy. The Pass^k measure, the share of tasks completed successfully on all k tries, was introduced for this purpose and adopted by later benchmarks including CAR-bench (Yao et al., 2025; Barres et al., 2025; Kirmayr et al., 2026; Rabanser et al., 2026). Among the benchmarks we considered, τ-bench and τ^2-bench (Yao et al., 2025; Barres et al., 2025) and CAR-bench (Kirmayr et al., 2026) evaluate the complete agent over repeated trials, whereas ToolBench and API-Bank (Qin et al., 2024; Li et al., 2023) isolate tool-finding but report single-run success; we therefore adopt CAR-bench for end-to-end behaviour and ToolBench for retrieval at scale. On the retrieval-only side, tool-finding quality is measured by Recall@k, the fraction of a task's required tools that appear in the top k, and COMP@k, the fraction of tasks for which all required tools appear in the top k (Qu et al., 2024). GEAR is a tool-finding method held to this stricter standard, which the tool-finding literature has not yet embraced.

3 METHODS

Every method answers the same question: which tools should enter the agent's context this turn? All methods run under the identical setup of §4.1.

Baseline retrieval and GEAR. Let $\mathcal{T}$ denote the tool collection, d_i the text of tool i (its name, description, and parameters), and E the BGE-small encoder, which turns a piece of text into a vector of numbers (Xiao et al., 2024; Reimers & Gurevych, 2019). Each tool's vector $\mathbf{e}_i = E(d_i)$ is computed once, in advance. At each turn τ, the retriever turns the conversation c_τ into a vector $\mathbf{q}_\tau = E(c_\tau)$ and scores every tool by how similar its vector is to the conversation's, using cosine similarity:

$$s_i(\tau) = \frac{\mathbf{q}_\tau^\top \mathbf{e}_i}{\|\mathbf{q}_\tau\| \, \|\mathbf{e}_i\|}, \qquad S_\tau^{\text{base}} = \underset{i \in \mathcal{T}}{\arg\,\text{top-K}}\, s_i(\tau) \cup C_\tau,$$

where K is the number of top-scoring tools to keep, and C_τ is the set of tools already called in the conversation. *Baseline retrieval* serves this shortlist S_τ^{base} unchanged. *GEAR* (Graph-Expanded Agent Retrieval) needs one extra thing: a graph $G = (\mathcal{T}, \mathcal{E})$ in which every tool is a node and every edge links two related tools. GEAR expands the baseline shortlist using this graph. On CAR-bench, the edges $\mathcal{E}$ come from the tool ontology (Appendix A), $G = (\mathcal{T},\, \mathcal{E}_{\text{REQ}} \cup \mathcal{E}_{\text{INF}} \cup \mathcal{E}_{\text{ROUTE}} \cup \mathcal{E}_{\text{CONF}})$. GEAR then grows the shortlist by traversing these edges outward from the tools already found:

$$S_\tau^{\text{graph}} = S_\tau^{\text{base}} \cup \text{REQ}^*\big(S_\tau^{\text{base}}\big) \cup N_{\text{INF}}\big(S_\tau^{\text{base}}\big) \cup N_{\text{ROUTE}}\big(S_\tau^{\text{base}}\big),$$

In other words, REQ^* follows REQUIRES edges all the way (a tool that needs another, which needs another, and so on), N_{INF} adds tools one INFORMS edge away (a tool that informs another), N_{ROUTE} adds tools that are often used together, and a CONFLICT edge (two tools that cannot be used at the same time) . As a result, a tool's companions enter the shortlist even when their own descriptions match nothing in the conversation. This is the shortlist-coverage idea, and the results trace it through both benchmarks (§5).

COLT: learned graph retrieval. Instead of edges written by hand, COLT learns them from which tools appear together in past tasks (Qu et al., 2024). First, an encoder is trained so that the score $s_i^{\text{COLT}}(q) = f_\theta(q)^\top g_\theta(d_i)$ ranks tools for a request q. Then a second stage re-orders the candidates over a graph that links requests, tools, and known-good tool sets, trained so that tools which together complete a correct tool set score highly together. We adapt COLT to CAR-bench under the identical setup: it trains on the 129 training instructions with duplicate tool sets removed, unseent testset tools

enter as nodes with no edges, and the harness serves precomputed top-12 rankings, with visibility and policy encoding switched off.

MCP-Zero: active discovery. MCP-Zero turns tool-finding around: instead of matching the user's words, the agent writes a short request for the capability it is missing, and a matcher finds the closest tools for that request (Fei et al., 2025). At each turn τ, the agent produces this request $r_\tau = \mathrm{LLM}(c_\tau)$ with a separate model call. A matcher E' (text-embedding-3-large) then scores every tool by $\cos(E'(r_\tau), E'(d_i))$, and the twelve closest form the shortlist, with visibility and policy encoding off; if a request fails, the method falls back to the previous turn's ranking. Baseline retrieval, GEAR, and COLT all start from what the user said (c_τ) and need no model call beyond the agent's own turn. MCP-Zero instead starts from what the agent says it needs (r_τ), so its tool-finding step is itself a second model call.

4 Experimental design

The two benchmarks described below supply every task used in this paper. §5 presents the specific setups we build on top of them, together with their results [1].

4.1 CAR-bench setup

CAR-bench sets up an in-car agent with 58 connected tools, 19 domain policies, and a simulated multi-turn user; a separate judge model then checks whether the policies were followed (Kirmayr et al., 2026). The main run uses gpt-5.4-nano as both the agent and the simulated user, with gpt-4.1 as judge. The repeat run uses claude-haiku-4.5, with claude-sonnet-4-6 as judge. On CAR-bench, GEAR's graph is the tool ontology (Appendix A). A task counts as solved only if it passes every check: the agent makes the correct final tool calls, the required look-ups, follows the policies, and ends the conversation correctly. Each task has a gold record listing its *gold actions*, the final calls that changes the state and solve the task. Some policies also require a look-up before a change is made; these mandatory look-ups are the *required state-reads*. This study uses the 50-task unseen base test split; every run records a manifest with models, seed, git commit, and the ontology. Every gpt-5.4-nano condition runs the same 50 tasks at three trials each, temperature 0 (a total of 1,800 conversations); the claude-haiku-4.5 replication reruns six core conditions with policy encoding off.

Each task runs $k = 3$ trials, giving two aggregates:

$$\mathrm{Pass@}k = \frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}} \mathbf{1}\left[\sum_{i=1}^{k} r_{t,i} \geq 1\right], \qquad \mathrm{Pass}^k = \frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}} \mathbf{1}\left[\prod_{i=1}^{k} r_{t,i} = 1\right],$$

where $r_{t,i} \in \{0, 1\}$ records whether task t succeeded (1) or failed (0) on trial i. We treat Pass^3 as the score a method must win to count as consistent. For $k < 3$, Tables 2 and 1 report the first k trials in order (the sequential-prefix convention), while the confidence intervals average over all $\binom{3}{k}$ subsets of trials (the unbiased estimator); at $k = 3$ the two give the same number.

4.2 ToolBench setup

ToolBench is the tool-finding benchmark COLT was built for, and it isolates one question: given a written request, can a method find the right tools in a large collection (Qin et al., 2024; Qu et al., 2024)? It has no simulated user and no policies. It comes in two sizes: a small collection of 1,419 tools and a large one of 11,474 (ToolBench's G3 and G2 splits). GEAR's graph cannot be written by hand at this scale, so on ToolBench it is built automatically from the data: for each split we count, over the training instructions, how often tools a and b appear in the same correct tool set, then divide by the largest count, $w_{ab} = c_{ab}/\max_{a'b'} c_{a'b'} \in [0, 1]$. The top-$k$ baseline-retrieval candidates act as seeds Σ_q, and every tool's score is nudged up by its strongest link to a seed,

$$s'_j = s_j + \lambda \cdot \max_{\sigma\in\Sigma_q,\, w_{\sigma j}\geq\theta} w_{\sigma j},$$

with seed count k, boost weight λ, and edge threshold θ tuned on a held-out slice of the training queries. Two scores summarise tool-finding quality: Recall@k, how many of a request's needed

[1] Code will be released upon publication.

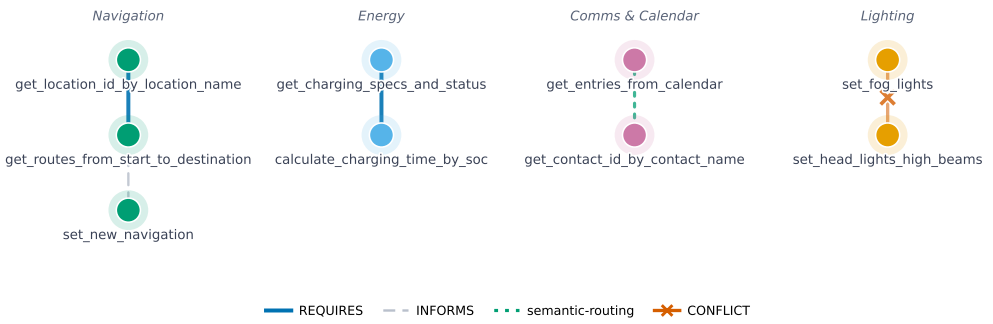

Figure 1: A small piece of GEAR's map of the in-car tools. Each dot is a tool, and each line says how two tools relate: which tool needs another, which should be checked before another, which are often used together, and which conflict. GEAR walks these lines to bring a tool's neighbours onto the shortlist, even when nobody asked for them by name.

tools appear in the top k, and COMP@k, how often all of them do (Qu et al., 2024). Formally, for query q with gold toolset G_q and top-k shortlist $S_k(q)$ over the query set $\mathcal{Q}$:

$$\text{Recall@}k = \frac{1}{|\mathcal{Q}|}\sum_{q\in\mathcal{Q}}\frac{|G_q \cap S_k(q)|}{|G_q|}, \qquad \text{COMP@}k = \frac{1}{|\mathcal{Q}|}\sum_{q\in\mathcal{Q}}\mathbf{1}\big[G_q \subseteq S_k(q)\big].$$

5 Results

We organise the evidence around the questions raised in §1. We first establish capability and consistency on CAR-bench (§5.1), then examine how the methods separate as tasks grow harder (§5.2), the effect of prompt-side additions in GEAR (§5.3), tool-finding quality at catalogue scale on ToolBench (§5.4), and finally the cost and latency each method incurs (§5.5). Table 1 collects the main CAR-bench results for both language models, and the subsections below read it one question at a time.

Table 1: Main results on CAR-bench for both language models, with tokens per trial. Pass@k requires success on at least one of k trials, Pass^k on all k (sequential-prefix convention for $k < 3$). Dashed lines separate baselines, GEAR conditions, and the golden tool set (upper bound with the correct tools supplied).

Condition	P@1	P@2	P@3	P^1	P^2	P^3	Tok/trial
gpt-5.4-nano							
baseline	0.34	0.36	0.36	0.34	0.30	0.28	101k
baseline + visibility	0.34	0.38	0.38	0.34	0.26	0.24	101k
COLT	0.52	0.60	0.68	0.52	0.42	0.32	92k
MCP-Zero	0.44	0.54	0.60	0.44	0.42	0.40	86k
GEAR	0.56	**0.68**	0.74	0.56	0.42	0.38	114k
GEAR + visibility	0.52	0.64	0.66	0.52	0.36	0.28	116k
GEAR + policy encoding	**0.60**	0.64	0.64	**0.60**	**0.48**	0.38	117k
GEAR + policy encoding + visibility	0.44	0.54	0.64	0.44	0.38	0.28	108k
Golden tool set	**0.60**	0.64	**0.76**	**0.60**	**0.48**	**0.44**	108k
claude-haiku-4.5 replication (encoding off)							
baseline	0.36	0.42	0.48	0.36	0.30	0.30	105k
COLT	0.54	0.60	0.64	0.54	0.44	0.36	**98k**
MCP-Zero	0.48	0.52	0.58	0.48	0.38	0.32	108k
GEAR	**0.64**	**0.72**	**0.78**	**0.64**	**0.48**	**0.40**	147k
Golden tool set	0.52	0.70	0.74	0.52	0.42	0.38	135k

5.1 Capability and Consistency on CAR-bench

Capability. On CAR-bench, GEAR attains the highest Pass@3 of the retrieval methods on both language models (Table 1). On gpt-5.4-nano, replacing similarity search with GEAR raises Pass@3 from 0.36 to 0.74; a McNemar test on the paired per-task outcomes gives $p = 0.0026$. The methods fall in a consistent order, GEAR 0.74, COLT 0.68, MCP-Zero 0.60, and similarity search 0.36, and the same order is observed on claude-haiku-4.5. This gain coincides with a reduction in false out-of-scope responses, from 68 conversations under similarity search to 6 under GEAR, consistent with GEAR placing required but unnamed tools on the shortlist.

GEAR's Pass@3 of 0.74 lies within 0.02 of the golden tool set (0.76), the condition in which the agent is supplied the correct tools in advance. On claude-haiku-4.5, GEAR reaches 0.78, at or above the golden tool set's 0.74. The capability still attributable to tool-finding on these tasks is therefore small.

Consistency. The large capability gains do not translate into proportional gains in consistency. Pass^3 stays within a narrow low band, roughly 0.24 to 0.44, across all conditions (Table 1), so no method makes a small agent behave the same way on every trial. Within that band GEAR is comparable to the strongest alternatives rather than dominant: it reaches Pass^3 of 0.38 on gpt-5.4-nano, within one task of MCP-Zero's 0.40, and leads on claude-haiku-4.5 at 0.40, against COLT 0.36 and MCP-Zero 0.32. Tellingly, even the golden tool set reaches only 0.44, so the consistency ceiling is itself low: supplying the correct tools raises how often a task is ever solved far more than how often it is solved every time. This gap between capability and consistency motivates the ablation in §5.3, which asks whether adding structure to the prompt can close it.

5.2 Effect of task difficulty

Grouping the 50 CAR-bench tasks by the number of gold actions they require localises GEAR's advantage to the harder tasks (Figure 2). On tasks with one or two gold actions, all methods perform comparably. GEAR's margin over COLT and MCP-Zero is largest through the middle of the range, where the average number of distinct tools per task rises from 1.4 to 6.2, and beyond roughly six gold actions all methods decline.

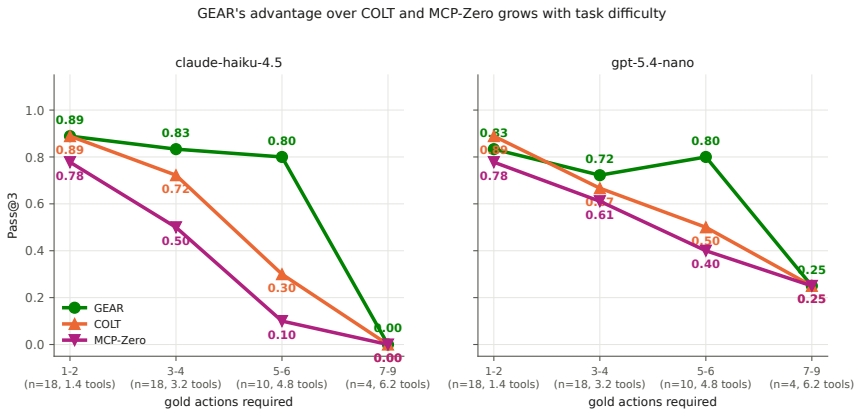

Figure 2: Pass@3 against gold actions per task, per method, for claude-haiku-4.5 (left) and gpt-5.4-nano (right). GEAR's margin over COLT and MCP-Zero is largest in the mid-range and narrows as all methods decline beyond roughly six gold actions.

5.3 Ablation of prompt-side additions

Having established that structure helps inside the retriever, we ask whether adding it to the prompt helps too. We isolate the two prompt-side additions on gpt-5.4-nano in a factorial ablation over three switches: graph-based retrieval, visibility of the ontology labels, and policy encoding in the tool descriptions (Table 2).

Table 2: Factorial ablation of the six switch conditions on CAR-bench's 50 unseen Base tasks (gpt-5.4-nano): the switch view of the corresponding rows in Table 1. Ticks mark the three switches each condition turns on: graph-based retrieval, showing the relationship labels, and putting the policies in the tool descriptions. For $k < 3$, Pass rates use the sequential-prefix convention.

				Pass@k			Pass^k		
Condition	Retri-eval	Visi-bility	Policy encoding	k=1	k=2	k=3	k=1	k=2	k=3
baseline	×	×	✓	0.34	0.36	0.36	0.34	0.30	0.28
baseline + visibility	×	✓	✓	0.34	0.38	0.38	0.34	0.26	0.24
GEAR	✓	×	×	0.56	**0.68**	**0.74**	0.56	0.42	**0.38**
GEAR + visibility	✓	✓	×	0.52	0.64	0.66	0.52	0.36	0.28
GEAR + policy encoding	✓	×	✓	**0.60**	0.64	0.64	**0.60**	**0.48**	**0.38**
GEAR + policy encoding + visibility	✓	✓	✓	0.44	0.54	0.64	0.44	0.38	0.28

Effect on capability. Neither addition improves Pass@3, and one clearly harms it. Encoding the policies into the tool descriptions lowers GEAR's Pass@3 from 0.74 to 0.64, and making the ontology labels visible lowers it to 0.66, with both together at 0.64; on the baseline, visibility moves Pass@3 only from 0.36 to 0.38. The best capability is reached by bare GEAR, which carries the emptiest prompt.

Effect on consistency. The two additions act differently on Pass^3. Making the labels visible lowers consistency wherever it is applied (GEAR 0.38 to 0.28; baseline 0.28 to 0.24), whereas policy encoding leaves it unchanged (0.38 for GEAR with and without it). No addition raises Pass^3 above bare GEAR's 0.38, and no condition escapes the low band of §5.1. Taken together, structure added to the prompt either costs a metric or does nothing, so it belongs in the retriever, not the prompt.

5.4 Recall and completeness on ToolBench

ToolBench isolates tool-finding at scale, over catalogues of 1,419 and 11,474 tools, with no simulated user and no policies. Two metrics summarise tool-finding quality: Recall@k, the fraction of a task's required tools that appear in the top k, and COMP@k, the fraction of tasks for which all required tools appear in the top k. GEAR's mined graph improves on similarity search in every cell of Table 3, raising both recall and completeness on the small and large catalogues, at zero API cost and with no training. COLT's trained ranker leads Recall@3 on the split it was retrained for (0.765 against GEAR's 0.708), but GEAR matches it on completeness by $k = 10$ (COMP@10 0.825 against 0.823). MCP-Zero is the strongest untrained method on Recall@3 with the off-the-shelf encoder, yet GEAR recovers the complete required tool set more often, and its margin over similarity search widens as the shortlist grows.

Table 3: Tool-finding quality on ToolBench, shown as Recall@3 / COMP@10 in each cell, with each method's total API-token cost.

Collection	Encoder	baseline	+ GEAR	COLT trained ranker	MCP-Zero
small (1,419 tools)	off-the-shelf	0.311 / 0.211	0.338 / 0.252	–	**0.398 / 0.266**
small (1,419 tools)	trained	0.686 / 0.798	0.708 / **0.825**	**0.765** / 0.823	–
large (11,474 tools)	off-the-shelf	0.235 / 0.096	0.243 / 0.134	–	**0.313 / 0.144**
large (11,474 tools)	trained	0.641 / 0.762	**0.657 / 0.791**	–	–
API tokens, total (small / large)		0 / 0	0 / 0	0 / –	2.70M / 10.40M

5.5 Cost and latency

CAR-bench. Token consumption per trial is reported in Table 1: on gpt-5.4-nano it ranges from 86k for MCP-Zero and 101k for the baseline to 114k–117k for the GEAR conditions, and reaches 147k for GEAR on claude-haiku-4.5. Per task, COLT is the cheapest ($0.012 on gpt-5.4-nano) because its rankings are precomputed, whereas MCP-Zero is the most expensive ($0.025 on gpt-5.4-nano and

$0.252 on claude-haiku-4.5) because its retrieval step is a second model call; GEAR uses the most tokens per trial yet costs less per task than MCP-Zero, and requires neither training nor a per-turn model call. Authoring the ontology is a one-off cost, reconstructed at roughly $2.10 for a single large-model session. Latency per conversation turn on gpt-5.4-nano is about 2.8 s for similarity search, 4.1 s for MCP-Zero, 4.8 s for GEAR, and 9.2 s for COLT's ranking server, and between 5.3 and 6.6 s on claude-haiku-4.5; GEAR's capability gain therefore costs roughly two additional seconds per turn over the baseline.

ToolBench. Retrieval runs offline, with no agent loop, so cost and latency are dominated by the encoder rather than the method. Similarity search, GEAR, and COLT's ranker all use local encoders at zero API cost, and GEAR's mined-graph re-ranking is a local operation that adds negligible token consumption or latency (Table 3). MCP-Zero is the exception: generating a capability request per query costs 2.70M API tokens on the small catalogue and 10.40M on the large one (roughly $0.55 and $2.05), a bill that grows with catalogue size while every other method's remains zero.

6 Discussion

The strongest result across our experiments belongs to the graph. On CAR-bench, GEAR solves more tasks than any other setup on both language models (Pass@3 0.74 and 0.78; Table 1). On ToolBench, GEAR's automatically built graph beats standard search in every cell, with no training and no API cost (Table 3). One idea explains both results. The graph retrieves all related tools, above all the information-gathering steps (the required state-reads), into the shortlist. Similarity search misses them, the gold records leave them out, COLT's training never shows them, and MCP-Zero cannot see them.

What to put in the prompt (Table 1). Both prompt additions make the agent worse, visibility on consistency and policy encoding on capability; the best condition carries the emptiest prompt. The graph belongs in the retriever, not the prompt. **Which method to run (Table 1).** GEAR beats COLT by 6 to 14 points of Pass@3, and MCP-Zero by 14 to 20 points. On Pass^3, GEAR leads clearly on claude-haiku-4.5 (0.40 against 0.36 and 0.32) and ties MCP-Zero within one task on gpt-5.4-nano (0.38 against 0.40); matching that tie costs MCP-Zero seven tasks of Pass@3 on the same model.

What each method costs. COLT is cheapest ($0.012 per task on gpt-5.4-nano) because its rankings precompute, but it needs two training stages GEAR never does and still trails GEAR on both metrics. MCP-Zero costs the most ($0.025, $0.252) because its retrieval step is a second LLM call; GEAR uses the most tokens per trial yet still costs less per task, and although MCP-Zero's own paper reports up to 98% token reduction on APIBank (Fei et al., 2025), under our shared-task protocol it is the costliest method and the only one priced above zero on ToolBench.

The choice at catalogue scale (Table 3). Counting co-occurring tools is free and helps at nearly every cutoff. COLT's trained ranker beats GEAR on Recall@3 in the split it was retrained for, and even there GEAR matches it on complete-toolset hits; on the large catalogue, retraining was out of scope, so COLT's cells are empty while GEAR's mined graph needs none. MCP-Zero's per-query request reaches the best untrained quality, at a bill that grows fourfold with scale and is never zero.

Which method to deploy follows from the setting: for a small, closed catalogue with written policies, author the ontology; with training data and a stable catalogue, COLT saves a little money at the cost of training and still trails GEAR; for an open catalogue with no training data, MCP-Zero avoids wrong impossibility claims at the highest cost and lowest capability.

7 Limitations

The in-car evidence rests on a test set of tasks, run on two small language models, and the ontology was authored against this same setting, so it may transfer less well to unfamiliar ones. The ontology was drafted by a language model with no human correction, so any errors in it propagate to every GEAR condition. The judge that scores the conversations is itself a model, and it is somewhat over-strict, wrongly failing about one in seven conversations whose actions exactly match the gold record. The scores carry wide confidence intervals, as small studies do, though the main capability result holds under any reasonable reading of them. Most importantly, no method we tried, GEAR included, reliably improved consistency under Pass^3: this is not a shortcoming of the graph but

a sign that reliability for a small agent lies somewhere other than in which tools it is shown, and identifying that lever is the problem we leave open for the field.

8 CONCLUSION

This paper tested shortlist filling and prompt additions, the two standard levers for a small tool-calling agent, under a strict consistency metric, across two benchmarks, three methods, and two language models. GEAR, retrieval expanded over a graph of tools and their relationships, delivered the best capability on both language models with a generated ontology, and carried the gain to full catalogue scale with a mined graph, for the reason set out in §6. Every prompt addition lowered a metric: structure helps this class of agent inside the retriever, not the prompt. Nothing we tried moved the consistency metric reliably; finding an intervention that does is the problem we pose to the field.

AI USE STATEMENT

A large language model drafted the in-car tool map, its labelled links, and its rule descriptions from CAR-bench's own tool documents, and a separate agent produced the reconstructed one-off cost figure for drawing the map. Language models also helped with editing this manuscript, with writing the analysis and figure code, and with finding related work. We reviewed all of this and take responsibility for the final content, including the text, claims, code, and results.

ETHICS STATEMENT

No real people take part in these experiments: the person in every in-car conversation is played by a language model, and the benchmark contains no personal data (Kirmayr et al., 2026). The setting carries a real safety stake, since an in-car agent that succeeds only on a retry could misdirect a driver, which is exactly why the every-time measure at the centre of this paper matters, catching such failures before deployment.

REPRODUCIBILITY STATEMENT

Every number here comes from the official CAR-bench evaluator on its standard test split, run three times per task at the steadiest model setting. The tool map is frozen and fingerprinted, and each run records which models were used, the random seed, the exact code version, and the map's fingerprint, so the results can be regenerated from the committed files and scripts.

REFERENCES

Anthropic. Introducing the Model Context Protocol. `https://www.anthropic.com/news/model-context-protocol`, 2024. Announced 2024-11-25.

Anthropic. Claude Haiku 4.5 system card. `https://www.anthropic.com/claude-haiku-4-5-system-card`, 2025. Released 2025-10-15.

Hamed Babaei Giglou, Jennifer D'Souza, and Sören Auer. LLMs4OL: Large language models for ontology learning. In *The Semantic Web – ISWC 2023*, 2023.

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

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

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

Michael Feld and Christian Müller. The automotive ontology: Managing knowledge inside the vehicle and sharing it between cars. In *Proceedings of the 3rd International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI)*, pp. 79–86. ACM, 2011. doi: 10.1145/2381416.2381429.

Xiaohan Feng, Xixin Wu, and Helen Meng. Ontology-grounded automatic knowledge graph construction by LLM under Wikidata schema. In *Workshop on Human-Interpretable AI (HI-AI) at KDD*, volume 3841 of *CEUR Workshop Proceedings*, pp. 117–135, 2024.

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

Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutiérrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *ACM Computing Surveys*, 54(4):71:1–71:37, 2021.

Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 33(2):494–514, 2022.

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

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020. doi: 10.18653/v1/2020.emnlp-main.550.

Johannes Kirmayr, Lukas Stappen, and Elisabeth Andre. Car-bench: Evaluating the consistency and limit-awareness of llm agents under real-world uncertainty. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 40599–40618. Association for Computational Linguistics, July 2026. doi: 10.18653/v1/2026.acl-long.1886. URL https://aclanthology.org/2026.acl-long.1886/.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, 2020.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Yanming Liu, Xinyue Peng, Jiannan Cao, Yuwei Zhang, Xuhong Zhang, Sheng Cheng, Xun Wang, Jianwei Yin, and Tianyu Du. Tool-planner: Task planning with clusters across multiple tools. In *International Conference on Learning Representations*, 2025a.

Yixin Liu, Guibin Zhang, Kun Wang, Shiyuan Li, and Shirui Pan. Graph-augmented large language model agents: Current progress and future prospects. *IEEE Intelligent Systems*, 41:45–55, 2025b. doi: 10.1109/MIS.2025.3642667.

Elias Lumer, Pradeep Honaganahalli Basavaraju, Myles Mason, James A. Burke, and Vamse Kumar Subbiah. Graph RAG-tool fusion. *arXiv preprint arXiv:2502.07223*, 2025.

Belinda Mo, Kyssen Yu, Joshua Kazdan, Joan Cabezas, Proud Mpala, Lisa Yu, Chris Cundy, Charilaos Kanatsoulis, and Sanmi Koyejo. KGGen: Extracting knowledge graphs from plain text with language models. In *Advances in Neural Information Processing Systems*, 2025.

OpenAI. Introducing GPT-5.4 mini and nano. https://openai.com/index/introducing-gpt-5-4-mini-and-nano/, 2026. Released 2026-03-17.

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

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

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 33rd ACM International Conference on Information and Knowledge Management (CIKM)*, 2024.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343, 2025. doi: 10.1007/s11704-024-40678-2.

Stephan Rabanser, Sayash Kapoor, Peter Kirgis, Kangheng Liu, Saiteja Utpala, and Arvind Narayanan. Towards a science of AI agent reliability. In *International Conference on Machine Learning*, 2026.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019. doi: 10.18653/v1/D19-1410.

Vyzantinos Repantis, Ameya Gawde, Harshvardhan Singh, and Joey Blackwell II. How many tools should an llm agent see? a chance-corrected answer. *arXiv preprint arXiv:2605.24660*, 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. In *Advances in Neural Information Processing Systems*, 2023.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. Toolgen: Unified tool retrieval and calling via generation. In *International Conference on Learning Representations*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.

Cailin Winston, Claris Winston, and René Just. Solver-aided verification of policy compliance in tool-augmented LLM agents. *arXiv preprint arXiv:2603.20449*, 2026.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2024. doi: 10.1145/3626772.3657878.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. ReWOO: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, 2025.

A THE TOOL ONTOLOGY IN FULL

Figure A1 draws the frozen tool ontology as a diagram, with 53 of its tools laid out so that related tools are pulled closer together. The ontology holds eight domains, 58 tools, and 81 labelled edges at the tool level (15 REQUIRES, 47 INFORMS, 15 semantic-routing, and 4 CONFLICT edges), plus 8 domain-level edges and 14 per-tool policy encodings. The complete, fingerprinted ontology, with every tool description, edge label, and parameter constraint, ships as a released artifact whose SHA-256 fingerprint appears in every run manifest. Figure 1 in the main text shows a small excerpt for orientation; the full dependency structure is drawn below.

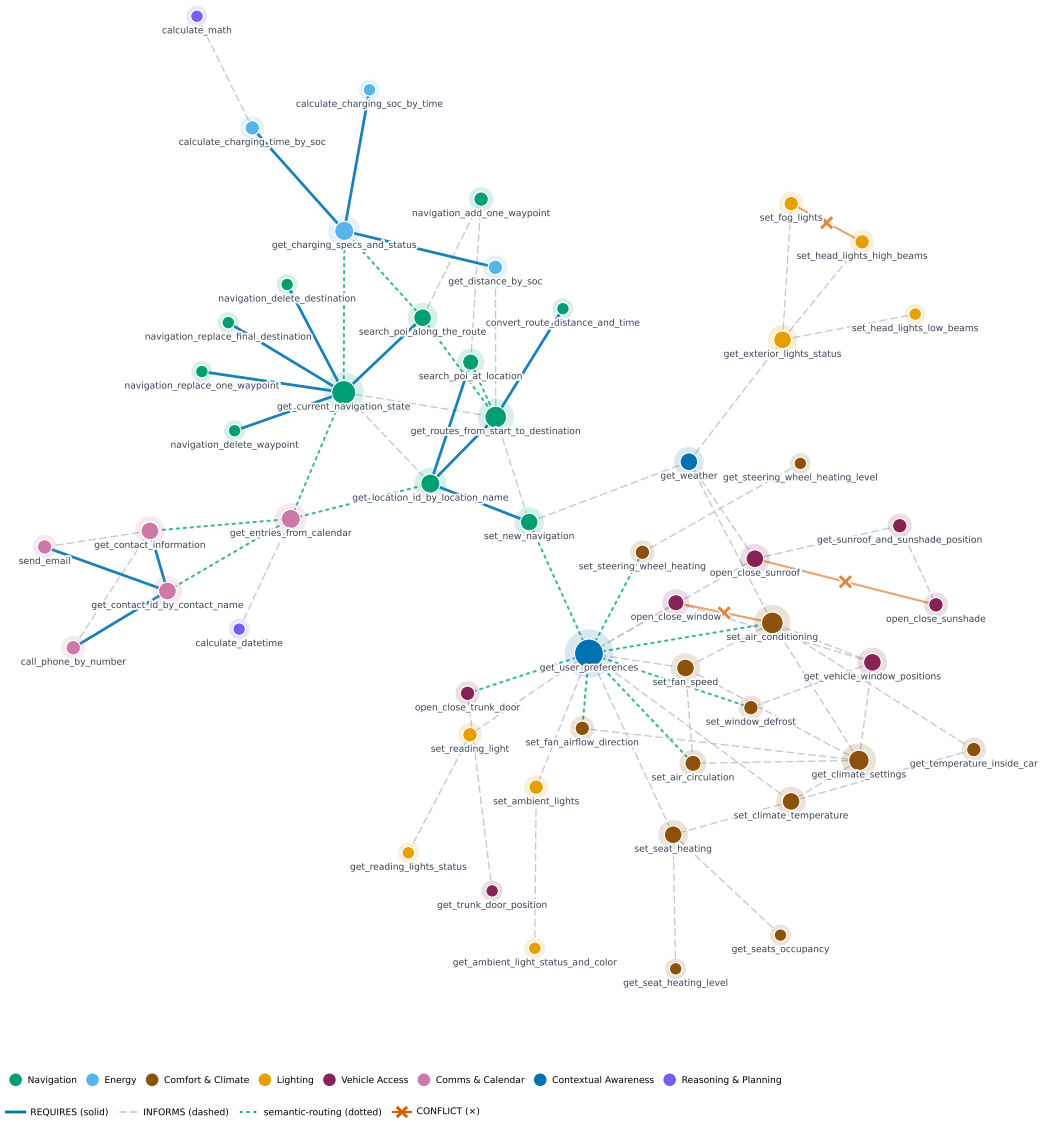

Figure A1: The tool ontology's dependency structure over 53 tools; node colour gives the domain, node size the degree, and line style the edge type.