Resource-Bounded Discovery of Reusable Actions

Roshan Klein-Seetharaman*

Submission to Learning in an Agentic World, COLT 2026 Workshop.

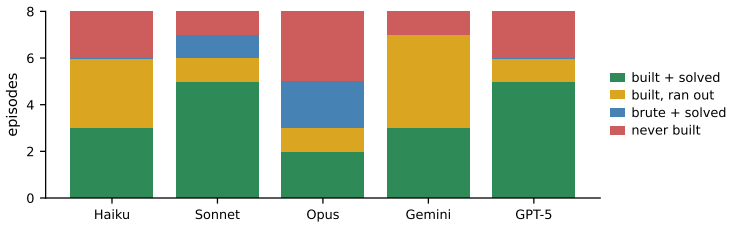
Overview. LLM-based agents are able to build, use, and chain together *given* tools. This capability is well benchmarked [4, 5, 6], and is a foundation for modern agentic systems. However, existing tool-use benchmarks are typically built around environments the models are heavily pretrained in, such as code-writing tool-creation [1, 2] or skill acquisition in semantically rich game worlds [3], and thus have strong priors about the tools they can build and their affordances. We introduce a relabeled interaction benchmark for *resource-bounded discovery of reusable actions* and use it to argue that three capabilities usually bundled under "tool use" are in fact separable: *recognizing* that a tool is possible, *constructing* it, and *exploiting* it efficiently under an action budget. The underlying environment and tool, although given obfuscated names, are predefined; the controlled difficulty is discovering when and how to construct it. A five-model pilot across inference providers shows that frontier models routinely construct the tool yet fail to use it efficiently, and that this inefficiency does not disappear with larger model size.

Environment family. Each environment $\mathcal{E}_n$ has n locked doors; the success event is opening all n within an action budget B. The action set is $\{\,\texttt{examine}(x),\ \texttt{combine}(x,y),\ \texttt{use}(k,d)\,\}$, and the agent sees only a textual observation $o(s)$ of the latent state s. Two routes solve the family. *Grind:* `examine`(door) returns the key to a *uniformly random* door, with replacement, so keys repeat—a coupon-collector process with expected cost $\mathbb{E}[\textsc{grind}] = nH_n + n$ (≈ 30 at n=8). *Construction:* `examine` also emits an incidental *part*; one hidden pair of part-types (T=3 types, giving $\binom{T}{2} = 3$ candidate pairs, exactly one correct) `combine`s into a machine m with $m(d) = \mathrm{key}(d)$ deterministically, giving an optimal reuse cost $\approx 2n + 1$ (17 at n=8). Construction lowers expected total cost once $n \geq n^* = 3$.

Relabeling and budget. Each episode, the tool-relevant tokens (part-types, recipe, machine) are relabeled by a draw σ from a random token pool, while door and key indices stay transparent so the grind route is always legible. σ is designed to suppress *task-specific retrieval priors*. We impose a hard budget $B = 36 \approx 1.2\,\mathbb{E}[\textsc{grind}]$ (generous: a competent builder needs only $\approx$ 18–21). The budget is load-bearing because build-rate alone is a weak signal: the recipe can only be found by trying combinations, and given enough actions even a scattershot search eventually builds the tool.

Pilot. We run five models (Claude Haiku 4.5, Sonnet 4.6, Opus 4.8, GPT-5, Gemini 2.5 Pro) on $K = 8$ *paired* episodes each (identical per-episode relabelings across models); all traces are replay-verified against the deterministic environment. We report *solve rate* (success within B); *wasted-action fraction* (actions that make no progress—null `combine`s, re-opening a door, machining a door whose key is already held); and *conditional deployment rate* (given the tool is built, whether it is used rather than ignored).

model	solve	waste	build
Haiku 4.5	0.38	43%	0.75
Sonnet 4.6	0.75	22%	0.75
Opus 4.8	0.50	24%	0.38
Gemini 2.5 Pro	0.38	37%	0.88
GPT-5	0.62	29%	0.75



Per-model results (left: solve rate, wasted-action fraction, build rate) and the outcome breakdown over the 8 *budgeted episodes (right). The "built, ran out" band—tool built but budget exhausted—is the inefficiency signature; "brute, solved" (solving by grinding, never building the tool) appears only for Opus and Sonnet.*

Models do not fail by *ignoring* the tool: whenever the machine is built it is then used (conditional deployment rate = 1.0). They fail by using it *inefficiently*. Both while searching for the recipe and after building it, 22–43% of actions make no progress. Further, these results offer preliminary evidence that model scale does not directly predict budgeted success on this class of tasks.

Limitations and next steps. One substrate, one budget value, small $K = 8$, and imperfectly controlled tokenization (mitigated by relabeling and averaging). Natural extensions—sweeping B, n, and recipe-search size, adding reps for confidence intervals, and varying reasoning effort—would test how general the construction–exploitation gap is. More broadly, relabeled combinatorial tasks of this kind offer a testbed for resource-bounded tool discovery in agents.

*Sea12 Technologies; Yale University. `roshan.klein-seetharaman@yale.edu`. Code and data: `https://github.com/rks277/reusable-action-discovery`

References

[1] C. Qian, C. Han, Y. R. Fung, Y. Qin, Z. Liu, H. Ji. CREATOR: Tool Creation for Disentangling Abstract and Concrete Reasoning of LLMs. *Findings of EMNLP*, 2023.

[2] T. Cai, X. Wang, T. Ma, X. Chen, D. Zhou. Large Language Models as Tool Makers. *ICLR*, 2024.

[3] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, A. Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models. *TMLR*, 2024.

[4] T. Schick, J. Dwivedi-Yu, R. Dessì, et al. Toolformer: Language Models Can Teach Themselves to Use Tools. *NeurIPS*, 2023.

[5] Y. Qin, S. Liang, Y. Ye, et al. ToolLLM: Facilitating LLMs to Master 16000+ Real-world APIs. *ICLR*, 2024.

[6] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao. ReAct: Synergizing Reasoning and Acting in Language Models. *ICLR*, 2023.