

Where the Tokens Go: A Measured Anatomy of LLM Agent Trajectory Costs

Tommy Tran

Independent Researcher

New York, New York, United States

tran.tommy@hotmail.com

Abstract

Tool-using agents repeatedly transmit prompts and conversation history, but aggregate token counts obscure which components drive cost. We introduce a five-component accounting convention for append-only agent loops, using provider usage records and fixed-prefix token-count probes without a local tokenizer. With prompt caching disabled, in 954 episodes (4891 requests) from one claude-haiku-4-5-20251001 model across three synthetic task families, the recurring fixed prefix accounts for 76.6% of baseline input tokens. Tool schemas contribute 69.1%, echoed assistant output 6.5%, and the tool-result residual, including framing, 16.9%. A descriptive fit to continuing episodes gives per-step input growth of about 363 tokens; longer-horizon cost ratios are extrapolations. Across the same 144 tasks, family-informed tool pruning reduces mean cost by 24.1% with a success difference of -1.4 percentage points (paired 95% CI [-3.5, 0.0]). A rolling history window saves 2.0% with identical observed outcomes. Truncating results at 600 characters reduces success from 98.6% to 47.2% while more than doubling mean cost alongside longer unsuccessful episodes. These findings apply to the tested model, mock tools, and short horizons. The accompanying package reproduces the analysis from frozen usage logs and prefix counts.

1 Introduction

Agentic use of large language models has a cost structure that differs sharply from single-shot inference. A tool-using agent operates as a loop: at every step the client resends the entire conversation so far (system prompt, tool schemas, task, all previous assistant turns, and all previous tool results) and pays for it again as input tokens. Practitioners know this in the abstract, and a growing literature measures *how much* agents cost (Erol et al., 2025; Liu et al., 2025; Wang et al., 2025), but surprisingly

little published work measures *where* the tokens actually go inside a trajectory: how much of the bill is the tool inventory, how much is retransmitted history, how much is the model’s own prior output echoed back to it, and how each component scales with trajectory length.

This gap matters because the cheap remedies that teams reach for first (dropping old history, pruning tool schemas, truncating tool results) act on *different* components of the bill. Without a measured decomposition, it is unclear which remedy targets a large component and which targets noise, and unclear what each remedy does to task success. Agent benchmarks such as AgentBench (Liu et al., 2024b), SWE-bench (Jimenez et al., 2024), WebArena (Zhou et al., 2024a), GAIA (Mialon et al., 2023), and τ -bench (Yao et al., 2024) report success and sometimes aggregate cost, but not the internal anatomy of spend; a recent survey of agent evaluation explicitly identifies cost-efficiency measurement as underdeveloped (Yehudai et al., 2026).

This paper contributes a small, careful measurement study aimed exactly at that gap:

1. Component accounting and a reproducible harness. We show that the per-request usage metadata, supplemented by fixed-prefix token-count probes, supports a five-component accounting of the input tokens of an append-only agent loop, via a simple accounting identity (Section 3), with no access to the provider’s tokenizer. Fixed-prefix counts reproduce the billed first-request input exactly on every baseline episode; continuation attribution is an accounting convention whose residual includes message framing.

2. A measured cost anatomy. Across three seeded task families (multi-hop document QA, structured data transformation, and filesystem-style search/read/write) executed by a claude-haiku-4-5-20251001 agent with deterministic mock tools and programmatic success checks, we quantify the anatomy of input spend and the growth law be-

hind it: per-request input rises by about 363 tokens per step, giving a descriptive quadratic cumulative-input fit (Section 5).

3. An honest evaluation of cheap interventions. We evaluate tool-schema pruning, rolling-window history, and marker-based tool-result truncation (at two budgets), alone and combined, on the cost/success frontier (Section 6). The results include a clear negative finding: aggressive truncation not only halves success but *raises* cost, because failed episodes run longer.

We deliberately keep the scope narrow. This is a single-model, single-provider measurement study over controlled synthetic environments; it does not claim scaling laws across models or realism across domains (Limitations). Its value is a clean, fully reproducible baseline anatomy, an intervention evaluation that reports failures as prominently as wins, and a harness that others can point at their own agents.

2 Related Work

Tool-using agents. ReAct-style interleaving of reasoning and actions (Yao et al., 2023) underlies most current tool agents, with tool invocation learned or elicited in various ways (Schick et al., 2023; Qin et al., 2024). Reflexion adds verbal self-correction across retries (Shinn et al., 2023), and MemGPT manages context as a memory hierarchy (Packer et al., 2023). All of these change the shape and length of trajectories, which is precisely the quantity our harness meters.

Agent benchmarks. AgentBench (Liu et al., 2024b), SWE-bench (Jimenez et al., 2024), WebArena (Zhou et al., 2024a), GAIA (Mialon et al., 2023), τ -bench (Yao et al., 2024), and the Berkeley Function-Calling Leaderboard (Patil et al., 2025) evaluate task success of tool agents; several report token or dollar totals, but none decompose spend within trajectories. Yehudai et al. (2026) survey agent evaluation and call out cost-efficiency as an open measurement problem. CostBench (Liu et al., 2025) evaluates whether agents can *plan* cost-optimally in the task domain; we instead measure the meta-level cost of running the agent itself.

Inference efficiency. Surveys of efficient LLM inference and serving (Zhou et al., 2024b; Wan et al., 2024) cover model- and system-level techniques. Closest to our interventions are prompt compression methods such as LLMLingua and

LongLLMLingua (Jiang et al., 2023, 2024) and agent-specific context optimization (Kang et al., 2026); these are learned or optimized compressors, whereas we evaluate the zero-effort heuristics practitioners deploy first and quantify what they actually buy. FrugalGPT (Chen et al., 2023) and cost-of-pass analysis (Erol et al., 2025) study cost at the level of model choice and economics; prompt caching (Gim et al., 2024) attacks the retransmitted-prefix component we measure. Long-context degradation (Liu et al., 2024a) provides an accuracy-side motivation for history reduction that is independent of our cost-side motivation. Recent work on efficient agents (Wang et al., 2025) studies framework-level design choices on web-search agents; our contribution is complementary: a component-level accounting method plus a controlled anatomy across task families.

3 Measurement Methodology

Setting. An episode is a sequence of API requests $t = 0, 1, \dots, n - 1$ to a chat completion endpoint with native tool use. Request t carries the system prompt S , the tool schemas D , the task statement Q , and the transcript so far; the model returns assistant content (text and tool calls), the client executes the tools and appends their results, and the loop repeats. The provider bills the full input of every request, so a token that enters the transcript at step k is paid for again at every subsequent step. Each response carries authoritative usage metadata: billed input tokens I_t and output tokens O_t (plus cache read/write counters, which we log and verify to be zero, as caching is disabled).

Accounting identity. In an append-only loop, the input of request t is the input of request $t - 1$ plus what was appended in between: the echoed assistant output of step $t - 1$ and a residual R_{t-1} covering tool results and changes in message framing. Hence

$$I_t = I_0 + \sum_{s < t} O_s + \sum_{s < t} R_s, \quad (1)$$

Here echoed assistant output is assigned its reported output-token count; any difference in its input serialization is included in R_s . This convention does not independently measure the tokenization of tool-result text. In the equations, I_0 covers the fixed prefix (system prompt, tool schemas, task statement, message framing). Since I_t and O_t are reported by the provider, Eq. 1 lets us *derive* the

residual attributed to tool results R_s without tokenizing anything ourselves: $R_{t-1} = I_t - I_{t-1} - O_{t-1}$. Cumulative input spend then decomposes exactly into

$$\sum_t I_t = nI_0 + \sum_t A_t + \sum_t H_t, \quad (2)$$

with $A_t = \sum_{s < t} O_s$ the echoed assistant history and $H_t = \sum_{s < t} R_s$ the tool-result history of request t .

Splitting the fixed prefix. To split I_0 into components we use the provider’s free token-counting endpoint on the exact request shapes: system prompt alone, system plus tool schemas, and each task statement. The tool-schema component includes the provider-side tool-use preamble injected whenever tools are present, which is a real recurring cost of enabling tools: for our full inventory it is 1602 tokens per request, versus 110 for the system prompt.

Validation. Two checks assess the fixed-prefix counts and accounting consistency. First, summing the counted components (system, schemas, task, message framing) predicts the billed I_0 with 0.00% error on all 144 baseline episodes: the token-counting endpoint reproduces the billing meter exactly. Second, the derived marginal tool-result increment $I_t - I_{t-1} - O_{t-1}$ is nonnegative at every one of the 454 baseline continuation steps (0 violations), as the identity requires, so the decomposition sums to observed episode input exactly rather than approximately. Accounting closure follows from defining the residual by subtraction and is not independent validation of its semantic components. Porting the decomposition requires checking the provider’s input/output serialization and obtaining fixed-prefix counts. History-editing agents break Eq. 1 by design; for those, the same logs still yield per-step input curves, as we show for the rolling-window condition.

4 Experimental Setup

Model and loop. All episodes use claude-haiku-4-5-20251001 (temperature 0, maximum 1,024 output tokens per step, at most 12 steps, no prompt caching) through the provider’s native tool-use API. The agent must finish by calling a `submit_answer` tool; an episode with no submission is a failure. We chose a small, cheap model deliberately: the intervention comparison required many real episodes under a fixed budget;

the observed anatomy depends on both the loop and the generated trajectories (total spend for every experiment in this paper: \$13.49).

Task families. We generate three families of 48 tasks each from a single recorded seed, with deterministic mock tools and programmatic success checks (exact or numeric match against ground truth; write tasks additionally check final file contents):

- **DocQA:** multi-hop questions (2 to 3 hops) over a 48-document synthetic corpus of personnel profiles, company briefs, and project overviews, with the key facts embedded at random positions inside filler text; tools `search_docs` and `read_doc`.
- **DataTrans:** aggregation and filtering questions over three relational JSON tables (20 customers, 15 products, 140 orders); tools `list_tables` and `read_table` with optional equality filters.
- **FileSys:** search/read/write tasks over a 24-file mock filesystem (configs, source files, logs); tools `list_files`, `read_file`, `search_files`, `write_file`.

The corpus, tables, and file tree are checked into the artifact, so every tool call is exactly reproducible. The agent always carries the full 9-tool inventory (a realistic multi-domain assistant configuration) unless a condition prunes it.

Conditions. We run every task under six conditions: **Baseline** (full inventory, full history, full results); **Tool pruning** (only the 3 to 5 tools relevant to the task family); **Truncation 600 / 1500** (tool results capped at 600 or 1,500 characters with an explicit [TRUNCATED] marker); **Rolling window** (history limited to the task statement plus the last 3 exchange pairs, with a note marking elided steps); and **Combined** (pruning + 600-character truncation + window).

Protocol. One episode per task per condition, plus two additional baseline runs on 45 tasks to quantify run-to-run variance at temperature 0. In total: 954 episodes, 4891 API requests, 10.8M input and 0.54M output tokens. Every request’s full usage record (input, output, cache read/write) is logged per step to an append-only JSONL cache; the analysis pipeline is deterministic (fixed bootstrap seed, 2,000 resamples for 95% CIs) and runs

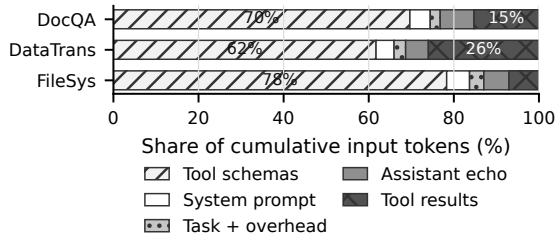


Figure 1: Attribution of cumulative input tokens in baseline episodes, by task family. The fixed prefix (tool schemas, system prompt, task statement) is retransmitted at every step and dominates spend at these trajectory lengths.

from the frozen log. Variance at temperature 0 is small but not nil: repeated baseline triples agree on success for 100.0% of tasks, with a mean per-task cost coefficient of variation of 0.1%.

5 Results: Cost Anatomy

The bill is mostly retransmission. Figure 1 shows where baseline input tokens go. Aggregated over families, 76.6% of all input tokens are the fixed prefix paid again at every step: 69.1% tool schemas and tool-use preamble, plus system prompt and task statement. The tool-result residual accounts for 16.9% and echoed assistant turns for 6.5%. The composition shifts with the family’s tool-result sizes: DataTrans episodes, whose `read_table` calls return large JSON payloads, spend 26.0% of input tokens on tool results, versus 7.1% for FileSys. Output tokens are only 4.4% of total tokens, but at a 5× per-token price they are 18.9% of baseline dollar cost.

A descriptive fit to per-step input. Mean input tokens per request rise near-linearly with step index in baseline episodes: a fit weighted by continuation count in the residuals to the per-step means gives a slope of 363 tokens per step from an intercept of 1711 ($R^2 = 0.930$ on step means; Figure 2, left). The fit uses step indices with at least ten continuing episodes. Each residual is multiplied by that step’s episode count before squaring; the reported R^2 instead weights squared deviations by the count. Later steps contain only episodes that have continued, so the task mix changes across means. Only indices 0 through 4 meet that threshold. Appendix D reports counts and pointwise uncertainty at every observed step. This is an in-sample description; the 6-step and 12-step costs extrapolate beyond the fitted indices and are not held-out predictions. Un-

der this growth law, cumulative input for a t -step episode is $I_0t + \beta t(t - 1)/2$: quadratic in t . The fitted model implies a 12-step to 6-step input ratio of $2.83\times$ and an 8-step to 4-step ratio of $2.64\times$ (Figure 2, right). Extra steps are not linearly priced, which is why interventions that shorten or bound history matter more the longer an agent runs, and why failure modes that add steps (retries, flailing) are doubly expensive.

6 Interventions on the Cost/Success Frontier

Table 1 and Figure 3 summarize the intervention grid. For paired comparisons we additionally re-sample task IDs within each family, retaining both conditions for each sampled task (10,000 bootstrap resamples, seed 12345). Pruning’s success difference is -1.4 percentage points (95% CI [-3.5, 0.0]); its mean cost saving is 24.1% [21.9, 26.4]. These intervals describe task variation within the three fixed synthetic environments, not transfer to new environments or models.

Tool-schema pruning reduces observed mean cost. Restricting the tool list to the task family’s tools cuts mean episode cost by 24.1% (1.19¢ to 0.90¢) with observed success of 97.2% versus 98.6%. Overlap of the marginal CIs does not establish equivalence. Pruning lowers cost per solved task from 1.20¢ to 0.93¢. This reduction targets the largest measured component: schemas are the single largest component (69.1% of input tokens), and pruning removes roughly 40% of the schema component (from 1602 to an average of 945 tokens per request, the residual being the per-request tool-use preamble that any nonempty tool list incurs). Pruning presupposes knowing which tools the task needs. Here the family is known and no routing cost is included; this is an oracle-family condition.

The rolling window saves little at these horizons. Aggregate observed success is unchanged (98.6%) and savings are only 2.0%. The reason is visible in Table 1: baseline episodes average 4.2 steps, and a window of 3 exchange pairs only begins discarding history at step 5. Windowing acts on the tail of long episodes, and at these horizons the tail is thin. Longer horizons could change both the savings and the information lost, and were not tested here. All paired success outcomes agree in this sample, so a task bootstrap gives a degenerate zero-width interval; that does not establish popula-

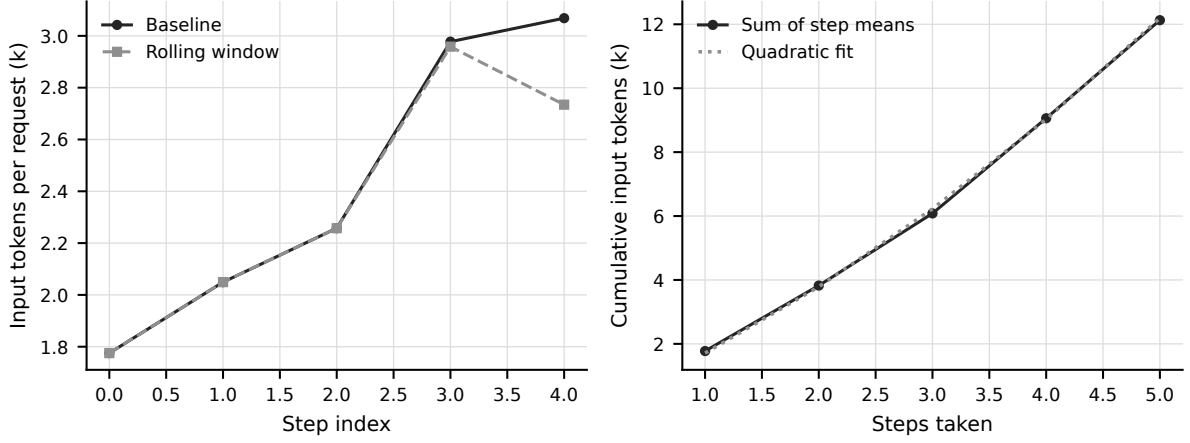


Figure 2: Left: mean input tokens per request by step index, baseline versus rolling window. Right: cumulative sums of the step-specific means versus the quadratic fit. These sums combine changing sets of continuing episodes; they are not mean cumulative costs for a fixed episode cohort.

Condition	DocQA		DataTrans		FileSys		All: cost/solved
	Succ. %	Cost ¢	Succ. %	Cost ¢	Succ. %	Cost ¢	¢
Baseline	100.0	1.40	95.8	1.26	100.0	0.89	1.20
Tool pruning	100.0	1.04	91.7	0.93	100.0	0.73	0.93
Truncation (1500)	100.0	1.40	89.6	1.35	95.8	1.08	1.34
Truncation (600)	20.8	3.49	29.2	2.86	91.7	1.20	5.33
Rolling window	100.0	1.36	95.8	1.24	100.0	0.88	1.18
Combined	25.0	2.13	22.9	1.60	89.6	0.93	3.40

Table 1: Task success and mean cost per episode (US cents) by condition and family (48 tasks per family, one episode per task and condition), plus aggregate cost per solved task. Aggregate 95% bootstrap CIs appear in Figure 3.

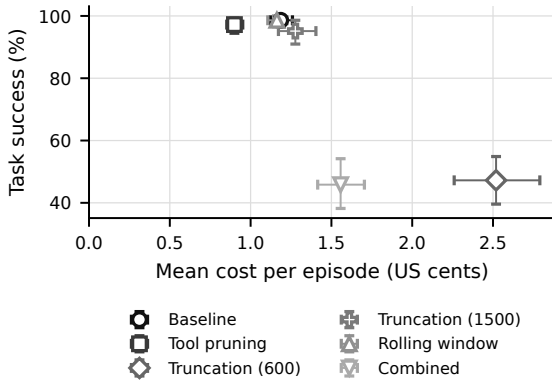


Figure 3: Cost/success frontier over all 144 tasks. Error bars are 95% bootstrap CIs. Tool pruning has lower mean cost and slightly lower observed success; aggressive truncation is worse on both axes.

tion equivalence.

Aggressive truncation is a false economy. Capping tool results at 600 characters collapses aggregate success to 47.2% while *raising* mean cost to 2.52¢ (1.19¢ baseline): the worst of both worlds,

and by far the worst cost per solved task (5.33¢, more than four times baseline). The mechanism is failure-driven lengthening: mean steps rise from 4.2 to 7.3 alongside repeated tool calls, and 51 of 144 episodes hit the step limit. The damage is family-dependent in an instructive way. DocQA (facts embedded at arbitrary positions in documents) falls to 20.8% and DataTrans (large JSON payloads cut mid-structure) to 29.2%, while FileSys, whose config files mostly fit under the cap and whose search tool returns compact matches, stays at 91.7%. Truncation attacks the component our anatomy shows is already small (tool results, 16.9% of input tokens) and pays for it with retries.

Moderate truncation does not reduce observed mean cost. At a 1,500-character cap, success recovers to 95.1% but the net saving is *negative* (-7.6%; 1.28¢ versus 1.19¢): the modest per-result reduction is offset by occasional failure-driven retries (steps 4.4 versus 4.2). Neither tested truncation budget reduces mean episode cost relative to baseline. Other budgets and content-aware meth-

ods were not evaluated.

Combining does not rescue a harmful component. The combined condition inherits truncation’s collapse (45.8% success) despite including the beneficial pruning, and still costs more than baseline (1.56¢). The tested combination does not offset the losses associated with truncation; we did not run a full factorial interaction study.

7 Discussion

Measure before optimizing. The single clearest lesson is that the intuitive ranking of remedies inverts under measurement. Practitioners often reach for tool-result truncation first because results *look* like the bulky part of a transcript; in our environments they are 16.9% of input tokens, while the invisible retransmitted prefix is 76.6%. The intervention aligned with the largest component (schema pruning) reduced mean cost with a small observed success decrease; the intervention aimed at a small component (truncation) degraded both axes at once. The decomposition identifies which components an intervention targets; its effect on success and episode length still requires measurement.

Failure is a cost multiplier. In an append-only loop, repeated history increases the input cost of additional steps, and failed episodes tend to be the long ones: truncated episodes averaged 7.3 steps against the baseline’s 4.2. Interventions must therefore be evaluated on cost per *solved* task, not cost per episode; by that metric truncation at 600 characters is over four times worse than doing nothing.

Implications for caching and serving. The tool inventory and system prompt are repeated across episodes, while task statements vary by task. Within an append-only episode, earlier history is also repeated and may be eligible for caching under the provider’s rules. The fixed-prefix share therefore does not bound all potentially cacheable input. We disabled caching and did not measure cache eligibility, hit rates, or savings. Our cost comparisons and pruning recommendation therefore apply to uncached execution. Cache creation charges, cache hits, and prefix reuse could change the ordering: family-specific tool lists or history edits may change reuse across requests. Truncating results before appending them does not by itself imply a cache miss on a later request. We have not measured these interactions (Gim et al., 2024).

Output tokens are small but not negligible.

Output is only 4.4% of tokens but 18.9% of dollars at this provider’s 5× output price. For agents built on models with longer visible reasoning, this asymmetry will grow, and the same per-request usage records support the analysis.

Limitations

Single model, single provider. Every number in this paper was measured on one small model (claude-haiku-4-5-20251001) through one provider’s API. The residual accounting closes by construction, while quadratic growth additionally assumes approximately constant mean appended length, but the specific shares (schema fraction, tool-result fraction), the success rates, and the intervention effects are model- and price-dependent and must not be quoted as general constants. Larger models may tolerate truncation better or worse; providers with different tool-use preambles will shift the fixed-prefix share.

Synthetic environments and short horizons. Our task families use deterministic mock tools, modest corpora, and trajectories of at most 12 steps. Real agents face noisier tools, larger results, and longer horizons; our own scaling analysis implies the history components grow in importance with horizon, so these shares need not describe long-running agents. The rolling-window result in particular (small savings) is a statement about short horizons, not a general verdict. Component shares also depend on our chosen prompt length, tool inventory, and mock-result sizes. We did not sweep those design choices.

Simple interventions, simple checks. The interventions are deliberately the zero-effort heuristics; learned compressors (Jiang et al., 2023; Kang et al., 2026) may achieve better frontiers. Success checks are strict string/numeric matches, which can undercount semantically correct but misformatted answers, and prompt caching, which directly targets the retransmitted prefix we measure, was disabled to keep accounting clean; measuring the anatomy under caching is future work.

Execution budgets. Across all 4891 requests, 11 stopped at the 1,024-token output limit (0.22%). Table 3 separates the main conditions from the baseline repeats. These stops can affect assistant-history size and subsequent behavior; their rarity does not establish that a larger cap would leave outcomes unchanged. The 12-step budget right-

censors the continuation of episodes that exhaust it, including 51 of 144 under 600-character truncation. Reported costs and lengths are realized values under this budget. For the same trajectories continued beyond the cap, accumulated costs and lengths could only increase; eventual success and cost per solved task remain unknown. We did not evaluate alternative output or step budgets.

Ethics Statement

All data is synthetic and generated from a recorded seed; no human subjects, personal data, or scraped content are involved. The experiments consumed a modest API budget (\$13.49) and we report exact token totals to make the environmental and monetary footprint transparent. Cost-reduction techniques evaluated here could be applied to any agent, beneficial or harmful; we see no elevated dual-use risk beyond that of the underlying agent technology. We follow the ACL Code of Ethics.

Reproducibility Statement

The accompanying reproducibility package contains the seeded task generator, deterministic mock tools, frozen per-step usage log, fixed-prefix token counts, and analysis code with a fixed bootstrap seed. The offline verification regenerates the reported numeric macros from those records. Fresh model runs may differ and incur API charges; the historical measured spend was \$13.49. The package does not require API access to reproduce the reported analysis.

Acknowledgments

The author used AI assistants for drafting and copy editing, including OpenAI Codex for camera-ready editing and offline verification. The reported experimental measurements come from the frozen API run logs. The author is responsible for the content, including the interpretation of the results.

References

- Lingjiao Chen, Matei Zaharia, and James Zou. 2023. [FrugalGPT: How to use large language models while reducing cost and improving performance](#). *Preprint*, arXiv:2305.05176.
- Mehmet Hamza Erol, Batu El, Mirac Suzgun, Mert Yuksekgonul, and James Zou. 2025. [Cost-of-pass: An economic framework for evaluating language models](#). *Preprint*, arXiv:2504.13359.
- In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. 2024. [Prompt cache: Modular attention reuse for low-latency inference](#). In *Proceedings of Machine Learning and Systems (MLSys)*.
- Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. [LLMLingua: Compressing prompts for accelerated inference of large language models](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. [LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.
- Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. [SWE-bench: Can language models resolve real-world GitHub issues?](#) In *The Twelfth International Conference on Learning Representations (ICLR)*.
- Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. 2026. [ACON: Optimizing context compression for long-horizon LLM agents](#). In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*.
- Jiayu Liu, Cheng Qian, Zhaochen Su, Qing Zong, Shijue Huang, Bingxiang He, and Yi R. Fung. 2025. [CostBench: Evaluating multi-turn cost-optimal planning and adaptation in dynamic environments for LLM tool-use agents](#). *Preprint*, arXiv:2511.02734.
- Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. [Lost in the middle: How language models use long contexts](#). *Transactions of the Association for Computational Linguistics*, 12.
- Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2024b. [AgentBench: Evaluating LLMs as agents](#). In *The Twelfth International Conference on Learning Representations (ICLR)*.
- Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. [GAIA: a benchmark for general AI assistants](#). *Preprint*, arXiv:2311.12983.
- Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. [MemGPT: Towards LLMs as operating systems](#). *Preprint*, arXiv:2310.08560.

- Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. [The Berkeley function calling leaderboard \(BFCL\): From tool use to agentic evaluation of large language models](#). In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. [ToolLLM: Facilitating large language models to master 16000+ real-world APIs](#). In *The Twelfth International Conference on Learning Representations (ICLR)*.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. [Toolformer: Language models can teach themselves to use tools](#). In *Advances in Neural Information Processing Systems 36 (NeurIPS)*.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. [Reflexion: Language agents with verbal reinforcement learning](#). In *Advances in Neural Information Processing Systems 36 (NeurIPS)*.
- Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, and Mi Zhang. 2024. [Efficient large language models: A survey](#). *Transactions on Machine Learning Research*.
- Ningning Wang, Xavier Hu, Pai Liu, He Zhu, Yue Hou, Heyuan Huang, Shengyu Zhang, Jian Yang, Jiaheng Liu, Ge Zhang, Changwang Zhang, Jun Wang, Yuchen Eleanor Jiang, and Wangchunshu Zhou. 2025. [Efficient agents: Building effective agents while reducing cost](#). *Preprint*, arXiv:2508.02694.
- Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. [\$\tau\$ -bench: A benchmark for tool-agent-user interaction in real-world domains](#). *Preprint*, arXiv:2406.12045.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. [ReAct: Synergizing reasoning and acting in language models](#). In *The Eleventh International Conference on Learning Representations (ICLR)*.
- Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2026. [A survey on evaluation of LLM-based agents](#). In *Findings of the Association for Computational Linguistics: ACL 2026*.
- Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024a. [WebArena: A realistic web environment for building autonomous agents](#). In *The Twelfth International Conference on Learning Representations (ICLR)*.
- Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, Shengen Yan, Guohao Dai, Xiao-Ping Zhang, Yuhang Dong, and Yu Wang. 2024b. [A survey on efficient inference for large language models](#). *Preprint*, arXiv:2404.14294.

A Tool Inventory and Prompt Sizes

The full inventory contains 9 tools: `search_docs`, `read_doc`, `list_tables`, `read_table`, `list_files`, `read_file`, `search_files`, `write_file`, `submit_answer`, each with a 2 to 4 sentence description and typed parameters. Measured with the provider’s token-counting endpoint on the exact request shapes: system prompt 110 tokens; full inventory plus tool-use preamble 1602 tokens; family-pruned inventories average 945 tokens (the preamble is a fixed cost of enabling any tools); task statements average 55 tokens.

B Condition Details

Rolling window: requests carry the task statement, a one-line note stating how many earlier exchanges were elided, and the last 3 (assistant, tool-result) exchange pairs; pairs are dropped whole to preserve tool-call/result pairing. Truncation: tool results longer than the cap are cut at the cap and suffixed with [TRUNCATED: k of m characters omitted]. Pruning: the tool list contains only the task family’s tools plus `submit_answer`. Episodes that end without a tool call receive one corrective user nudge before being marked failed.

C Example Tasks

DocQA (3 hops): *In which city is the employer of the lead of Project Kestrel headquartered?* The agent must find the project overview, the lead’s profile, and the employer’s company brief. **Data-Trans**: *Compute the total revenue (quantity times unit_price) from orders with status shipped placed by customer C07 in 2025; submit the number rounded to 2 decimals.* **FileSys** (write variant): *Find the value of the setting flush_interval_ms under /app/config, write exactly that value to /report/flush_interval_ms.txt, then submit the value.*

Condition	Step	n	Mean	95% CI
Baseline	0	144	1775	[1774, 1777]
Baseline	1	144	2049	[1965, 2150]
Baseline	2	131	2258	[2241, 2273]
Baseline	3	119	2978	[2715, 3254]
Baseline	4	44	3068	[3049, 3087]
Baseline	5	8	2990	[2956, 3024]
Baseline	6	8	3255	[3223, 3286]
Window	0	144	1775	[1774, 1777]
Window	1	144	2049	[1965, 2151]
Window	2	131	2258	[2241, 2273]
Window	3	119	2958	[2714, 3221]
Window	4	41	2734	[2718, 2750]
Window	5	8	2483	[2461, 2505]
Window	6	8	2566	[2546, 2583]

Table 2: Per-request input means and pointwise, family-stratified task-bootstrap intervals from the frozen main runs. Values are rounded to whole tokens. Step index 0 is the first request.

Condition	Output stops	Step stops
Baseline	1	0
Tool pruning	0	0
Truncation (600)	5	51
Truncation (1500)	0	1
Rolling window	1	0
Combined	2	47
Baseline repeats	2	0

Table 3: Output stops count requests whose recorded stop reason is `max_tokens`; step stops count episodes recorded as `max_steps`. Main conditions each contain 144 episodes. The additional baseline repeats contain 90 episodes. Counts use different units and should not be added together.

D Step Counts and Execution Budgets

Table 2 supplements Figure 2 with all observed baseline and window steps. At each step, we re-sample continuing task records within each represented family, retaining its observed count (10,000 resamples, seed $12345 + t$). Intervals describe the conditional mean for that step’s continuing tasks; they are pointwise, not simultaneous bands or uncertainty for a fixed cohort’s cumulative cost. The eight-task tail is particularly limited. It is reported here but excluded from the original fit’s ten-task threshold.