Eta Given Delta: Defining LLM Tool Efficiency With Marginal Tool Utility

Nyx Iskandar*
Foam
nyx@foam.ai

Abstract

This paper introduces **tool efficiency**, a new quantitative metric to evaluate the rate of *useful* tool calls in an LLM agent trajectory. To ensure that tool efficiency is well-defined, we also introduce **marginal tool utility**, a new quantitative metric defined per tool call indicating whether a tool is useful or whether it can be safely removed from the tool suite without affecting accuracy while increasing tool efficiency; in this paper, we determine the sign of marginal tool utility for each tool call in a trajectory using LLM-as-a-Judge. While much prior work has been done to develop techniques that improve tool use by LLMs and design evaluation methods measuring efficiency indirectly using accuracy as a proxy, our work is centered on measuring efficiency *directly* via the quantitative metric proposed in this paper in post hoc trajectory analyses. It is our intention that this work contributes to the frontier of LLM evaluation research as a springboard for future benchmark designs and agent harness engineering (specifically with regards to creating lean tool suites) that optimize for metrics that complement but are distinct from accuracy.

1 Introduction

Large Language Models (LLMs) are increasingly used in real-world tasks such as software engineering (Chen et al., 2021; Athiwaratkun et al., 2023; Austin et al., 2021), mathematical reasoning and autoformalization (Ahn et al., 2024; Guo et al., 2025; Manem et al., 2025), and scientific research (Zhuang et al., 2025; Chaturvedi et al., 2026). However, should LLMs remain simply as language modeling artifacts (Brown et al., 2020) without access to external up-to-date knowledge (Cheng et al., 2024) and no way to act or modify its environment (Yao et al., 2020; Huang et al., 2022; Ahn et al., 2022), they can hardly be called agents (Russell and Norvig, 2020). To address this, LLMs are equipped with tools (Schick et al., 2023), which have become an integral part of the LLM agentic workflows that we see today (Yao et al., 2022).

While tools are generally useful, not all tools are equally useful (Huang et al., 2024; Yu et al., 2025), this finding acknowledged even by frontier industry labs through launching features such as tool search (Anthropic, 2025). To that end, much prior work focuses on developing techniques that improve tool use (Sivakumaran et al., 2026) and on creating benchmarks that such techniques can evaluate against and optimize for (Ning et al., 2024). A common ground shared by existing work is that the evaluation metric used is almost always outcome accuracy (Jin et al., 2025; Li et al., 2025b), even if accuracy is not the primary metric of interest, rather it being used as a proxy to measure the efficiency of tool use (Ning et al., 2024).

Given this, we notice a gap in the literature: there does not yet exist a direct way to measure tool efficiency, which we define as the rate of useful tool calls, given an LLM agent trajectory. While recent works like TRM by Ma et al. (2026) take the first steps towards defining reward functions centered

*Alternate email address: nyx@berkeley.edu

Preprint.

on intermediate tool invocations for reinforcement learning to improve tool call quality, we take this idea one step further by enabling post hoc analyses on agent trajectories with an aggregate tool efficiency metric, and keeping to solely inference-time methods easily adoptable by application-layer engineering teams to assist them in designing leaner and more maintainable agent tool suites.

In attempting to define tool efficiency, we must first define what usefulness means with respect to a tool call. We arrive at the idea of marginal tool utility, which is a metric relevant to each particular tool call instance in an agent trajectory that directly determines the usefulness of these tool calls, and whose aggregate per tool determines the usefulness of the tool itself. This definition must be supported by empirical results, specifically those obtained from tool ablations and observing the change in accuracy (or lack thereof) depending on which tool was ablated. To that end, we used the APEX-SWE Observability benchmark, a benchmark in which today's frontier models still struggle to achieve above 40% accuracy as reported by Kottamasu et al. (2026), and a benchmark that by default includes tools that we hypothesize (and later prove) are necessary (useful) as well as unnecessary (removable without affecting accuracy).

As such, **our contributions** in this paper are four-fold:

1. Introduce marginal tool utility Δ_α defined per tool call, specifically for pass/fail tasks. Aggregate tool utility $\sum \Delta_\alpha$ defined per tool predicts whether said tool affects the agent's accuracy when ablated.
2. Introduce tool efficiency η_α, where a useful tool call α_i is one with $\Delta_\alpha(\alpha_i) > 0$.
3. Implement a simple and cost-efficient method for calculating η_α for pass/fail tasks by using LLM-as-a-Judge to determine the sign of each $\Delta_\alpha(\alpha_i)$.
4. Empirically support the correctness of our definitions and LLM-as-a-Judge evaluator through tool ablations on the APEX-SWE Observability benchmark; critically, adding tools with $\sum \Delta_\alpha \leq 0$ does not increase the accuracy of the agent, while adding tools with $\sum \Delta_\alpha > 0$ does increase the accuracy of the agent, all as expected.

2 Related Works

LLM evaluations. There exists a plethora of benchmarks to assess an LLM agent's accuracy in completing various tasks, including SWE-bench (Jimenez et al., 2024), Terminal-Bench (Merrill et al., 2026), BrowseComp (Wei et al., 2025), and OSWorld-Verified (Xie et al., 2024). Benchmarks specifically designed to evaluate an agent's ability to use tools to correctly complete tasks include MCP-Atlas (Bandi et al., 2026), ToolBench (Qin et al., 2023), StableToolBench (Guo et al., 2024), and Toolathlon (Li et al., 2025a). While they differ in their exact evaluation method, the aforementioned benchmarks all focus on evaluating accuracy. Even benchmarks to specifically assess the quality of tool use like WTU-EVAL by Ning et al. (2024) use accuracy as a proxy to determine whether tool usage is done correctly. The quantitative metrics that we introduce in this paper allow future benchmarks to bypass accuracy altogether, or use them as complementary metrics to accuracy.

LLM tool call optimizations. Many methods have been proposed by existing work to optimize LLM tool usage: reward models (Agarwal et al., 2026; Ma et al., 2026), self-play reinforcement learning (Acikgoz et al., 2026), and tool retrieval (Moura, 2025; Erdogan et al., 2024). Other interesting related work include ART (Paranjape et al., 2023) that approach tool call optimization not necessarily from the lens of improving tool usage, but using tool outputs to improve generation dynamically. Several works like Yu et al. (2025) conclude that some tasks benefit from tool usage while others do not, suggesting that tool usefulness is task-dependent and not equal across all tools. Similar to Ma et al. (2026), we extend this idea and hypothesize that usefulness is not equal across all tool *calls*, though instead of developing reward models for reinforcement learning, we derive aggregate metrics to be used by application-layer developers conducting post hoc analyses on their agent trajectories to optimize their agent harness (Lou et al., 2026) without fine-tuning (Zhang et al., 2024). Moreover, marginal tool utility and tool efficiency can add another dimension to future dynamic tool synthesis methods like Test-Time Tool Evolution (Lu et al., 2026), which currently calculates semantic similarity between planning steps and tool descriptions to dynamically select tools from a tool library or to generate and refine new tools.

Methods improving agentic efficiency. To the best of our knowledge, tool efficiency is not a well-defined nor quantitative metric prior to this paper. However, this does not mean that there

has been no prior work that is focused on improving efficiency in LLM agentic workflows in the broad sense. For instance, Recursive Language Models (Zhang et al., 2026) and FrugalGPT (Chen et al., 2024) focus on minimizing API cost, LLMLingua (Jiang et al., 2023) on minimizing inference latency, and an instruction-refinement framework by Wu et al. (2025) on maximizing Cost-Aware Pass Rate (which is a metric introduced in the same paper). We consider the metrics these methods aim to optimize *efficiency-adjacent*, reserving the term *efficiency* for a metric that calculates the rate of *useful* items as is accepted in physics, where efficiency is the ratio of useful energy to total energy input of a system (Serway et al., 2012).

3 Methodology

To enable direct and explicit measurements of efficiency, we first introduce **marginal tool utility**, particularly in the case of pass/fail tasks. We then introduce **tool efficiency**, where the usefulness of a tool call corresponds to the sign of marginal tool utility of said tool call.

3.1 Problem Setup

We formalize a multi-turn LLM agent trajectory as an ordered sequence of token sequences. Assuming the ReAct paradigm (Yao et al., 2022), which is the paradigm used for the agent harness in APEX-SWE Observability (Kottamasu et al., 2026), an LLM π alternates between generating reasoning tokens and executing tool calls.

Formally, consider LLM π as well as system message s and user message u. Given prompt $p \triangleq (s, u)$, LLM π executes multiple tool calls sequentially and outputs reasoning tokens in between tool calls to synthesize the information it has access to at each timestep. This iterative process terminates once LLM π reaches the maximum allowable step count or calls a terminal tool to produce final answer y. Hence, we define agent trajectory τ as

$$\tau \triangleq (p, t_1, r_1, o_1, ..., t_N, r_N, o_N, t_{N+1}, y) \tag{1}$$

where each $t_i (1 \leq i \leq N+1)$ denotes a sequence of reasoning tokens, each $r_i (1 \leq i \leq N)$ denotes a tool request, each $o_i (1 \leq i \leq N)$ denotes the tool output corresponding to r_i, and $N \in \mathbb{N}$ denotes the total number of steps taken by the agent that is no greater than the maximum step count set as a hyperparameter. We define a tool call as a paired tool request and tool output, or formally,

$$\alpha_i \triangleq (r_i, o_i) \tag{2}$$

To validate our paper's proposed definitions, we must define the expected correctness of final answer y. In APEX-SWE Observability, final answer y is the final code patch generated by the agent to solve its automatic program repair task (de Souza et al., 2017). Correctness is determined by unit tests following the FAIL_TO_PASS/PASS_TO_PASS methodology inspired by SWE-bench (Jimenez et al., 2024). We define each unit test as function $f_j \colon \mathbb{N}^n \to \{0, 1\}$, where $f_j(y) = 0$ when final answer y causes unit test j to fail and $f_j(y) = 1$ when final answer y causes unit test j to pass. The expected correctness of final answer y over trajectories generated by LLM π is thus

$$\mathbb{E}_{\tau \sim \pi}[\mathbb{I}(f_1(y) \cdot ... \cdot f_M(y) = 1)] \tag{3}$$

where $M \in \mathbb{N}$ denotes the total number unit tests such that $1 \leq j \leq M$, and $\mathbb{I}(\cdot)$ is the indicator function.

3.2 Defining Marginal Tool Utility

Given trajectory τ of an agent tasked to solve a pass/fail task, the marginal tool utility of the i^{th} tool call $\Delta_\alpha(\alpha_i)$ is defined as the difference between the likelihood that the task is solved correctly given tool calls up to and *including* α_i and the likelihood that the task is solved correctly given tool calls up to and *excluding* α_i. Among other methods, this can be determined using repeated policy π rollouts or using LLM-as-a-Judge (Zheng et al., 2023; Yu, 2025) comparing the before and after trajectories. The latter is more computationally efficient and practical than the former, hence we opt

to use LLM-as-a-Judge in this paper, specifically to determine $sgn(\Delta_\alpha(\alpha_i))$, where $sgn(\cdot)$ is the signum function. We argue that this is an acceptable method as a tool call α_i is *useful* if and only if upon its execution the likelihood of correctness increases ($\Delta_\alpha(\alpha_i) > 0$). Hence, the additional information afforded by repeated policy π rollouts, that being the exact likelihood difference before and after the tool call, is unnecessary for our purposes.

Formally, consider judge LLM $\pi^{(J)}$ as well as system message $s^{(J)}$ and user message $u^{(J)}(\alpha_i, \tau)$. Given prompt $p^{(J)} \triangleq (s^{(J)}, u^{(J)}(\alpha_i, \tau))$, judge $\pi^{(J)}$ generates final answer $y^{(J)}$, which is a structured output that can be represented as a triplet (L, C, R), where $L = \{0, 1\}$ denoting two possible marginal tool utility labels (`positive` or `non_positive`), $C = [0, 1]$ denoting a float confidence score, and $R = \mathbb{N}^n$ denoting tokens that read as the judge's rationale for its classification.

The judge trajectory $\tau^{(J)}$ is thus simply

$$\tau^{(J)} \triangleq (p^{(J)}, y^{(J)}) \tag{4}$$

Given judge $\pi^{(J)}$, the i^{th} tool call α_i, the trajectory up to and excluding the i^{th} tool call $\tau_{1,\dots,i-1}$ (i.e., before trajectory), and the trajectory up to and including the i^{th} tool call $\tau_{1,\dots,i}$ (i.e., after trajectory), we get $y_i^{(J)}$. In particular,

$$\begin{cases} L_i = 1 \iff \Delta_\alpha(\alpha_i) > 0 & \text{tool call is useful} \\ L_i = 0 \iff \Delta_\alpha(\alpha_i) \leq 0 & \text{otherwise} \end{cases} \tag{5}$$

Notice that marginal tool utility is defined per tool *call*. For a particular trajectory τ, a *tool* is useful if and only if it has more useful ($\Delta_\alpha(\alpha_i) > 0$) calls than it has non-useful ($\Delta_\alpha(\alpha_i) \leq 0$) calls. Equivalently, for all tool calls α_i of a particular tool,

$$\sum \Delta_\alpha(\alpha_i) \begin{cases} > 0 & \text{tool is useful} \\ \leq 0 & \text{otherwise} \end{cases} \tag{6}$$

We can also define this aggregate for a set of independent trajectories. This is only advisable in practical settings if the independent trajectories share the same tool suite and each corresponds to a task comparable to each other (e.g., each corresponds to an observability task using the same agent harness within the same benchmark).

For reproducibility, we outline the specific system message $s^{(J)}$ and user message $u^{(J)}$ in Appendix A, and note that the particular model used for judge LLM $\pi^{(J)}$ is GPT-5.4 on Microsoft Azure (OpenAI, 2026b).

3.3 Defining Tool Efficiency

Having determined $sgn(\Delta_\alpha(\alpha_i))\forall i$, we can define tool efficiency as the ratio of the number of useful tool calls to the total number of tool calls in a trajectory. Formally, for trajectory τ with N tool calls,

$$\eta_\alpha(\tau) \triangleq \frac{|\{\alpha_i | \Delta_\alpha(\alpha_i) > 0\}|}{N}, 1 \leq i \leq N \tag{7}$$

We can thus also define the mean tool efficiency of a set of L independent trajectories

$$\overline{\eta_\alpha}(\tau^{(1)}, ..., \tau^{(L)}) = \frac{\eta_\alpha(\tau^{(1)}) + ... + \eta_\alpha(\tau^{(L)})}{L} \tag{8}$$

We hypothesize, and empirically show in Section 4, that tool efficiency increases when a non-useful tool is removed from the tool suite. This is not trivially implied by the definition of tool efficiency, most significantly because the total number of tool calls N can vary given a change in the tool suite.

3.4 Experimental Setup

We conduct our empirical experiments to show that the concept of marginal tool utility aligns with what we expect to see in practical settings when performing tool ablations. Particularly, if a tool has positive aggregate tool utility, the expected correctness of the agent's final answer increases with the inclusion of the tool in the agent's tool suite; if a tool has non-positive aggregate tool utility, the expected correctness of the agent's final answer remains constant or decreases with the inclusion of the tool in the agent's tool suite. We show that these hypotheses are supported by the empirical correctness results.

We evaluate frontier LLMs against the 25 public observability tasks of APEX-SWE (Kottamasu et al., 2026) available on HuggingFace. Kottamasu et al. (2026) noted that this public subset is representative of the whole set of 100 observability tasks, of which 75 are private.

In APEX-SWE Observability, the default agent harness includes native agent tools (bash, search_files, read_file, apply_patch, update_plan) and three read-only MCP tools exposed via bash: Grafana/Loki for logs, Mattermost for discussions between human developers, and Plane for software issues/tickets/specifications. Being familiar with how observability tasks for production software are often solved, we hypothesize that Grafana/Loki is a useful MCP tool with positive aggregate tool utility, while Mattermost and Plane are non-useful MCP tools with non-positive aggregate tool utility.

To prove that hypothesis, we conducted tool ablations to obtain three variants of the agent harness:

- `default`: Full suite of MCP tools (Grafana/Loki, Mattermost, Plane).
- `grafana`: Only Grafana/Loki MCP is available.
- `no-mcp`: Absolutely no MCP tools are available.

Through these ablations, we are able to verify (1) whether the inclusion/omission of Mattermost and Plane affects accuracy, and (2) whether the inclusion/omission of Grafana/Loki affects accuracy. In Section 4, we present the task accuracy results we obtained and cross-reference them with the aggregate tool utility results of the three MCP tools. We did not conduct ablations on the native agent tools as they are either necessary for the task to be completed (e.g., apply_patch) or for the MCP tools to be executed (e.g., bash).

Agent specifications. We used GPT-5.3-Codex on Microsoft Azure (OpenAI, 2026a) and Gemini 3.1 Pro on Google Agent Platform (formerly Vertex AI) (Google, 2026) to create two independent agent instances. Therefore, we obtained a total of $25 \cdot 3 \cdot 2 = 150$ distinct agent trajectories.

Compute specifications. The proprietary LLMs were accessed via their respective provider APIs and thus run on their respective providers' compute clusters. The experiments were otherwise run on AWS Batch using EC2 CPU workers, each as an isolated containerized job with Docker-in-Docker enabled. Jobs used 16 vCPUs, 110 GB RAM, and a 600 GB gp3 root volume. The AWS Batch compute environment used the `r7i.4xlarge`, `r7i.8xlarge`, and `m7i.8xlarge` instance types.

4 Results

In this section, we report the results we obtained from all 150 agent trajectories, namely task accuracy, marginal and aggregate tool utility, and mean tool efficiency. We show that task accuracy increases with the inclusion of a tool with positive aggregate tool utility, and tool efficiency increases with the removal of tools with non-positive aggregate tool utility.

4.1 Task Accuracy

We find that task accuracy does not significantly change when both Mattermost and Plane are removed (`default` vs `grafana`) and that task accuracy noticeably decreases when Grafana/Loki is removed (`grafana` vs `no-mcp`), as seen in Tables 1 and 2.

Table 1: Accuracy across 25 trajectories per variant for GPT-5.3-Codex. Each set of 25 tasks were run twice resulting in consistent accuracies.

Variant	Pass	Fail	Accuracy
`default`	8	17	0.32
`grafana`	9	16	0.36
`no-mcp`	6	19	0.24

Table 2: Accuracy across 25 trajectories per variant for Gemini 3.1 Pro. Each set of 25 tasks were run twice resulting in consistent accuracies.

Variant	Pass	Fail	Accuracy
`default`	9	16	0.36
`grafana`	9	16	0.36
`no-mcp`	5	20	0.20

A breakdown of which specific tasks passed/failed can be found in Appendix B. Comparing `default` and `grafana`, when a task is completed correctly for one it is also generally completed correctly for the other. A similar relationship exists for tasks that were completed incorrectly.

From these results, note that we can qualitatively claim that Grafana/Loki is a useful tool, while Mattermost and Plane are not. This claim aligns with the nature of the tasks: an agent completing an observability task benefits more from reading logs (primary source of information) than from reading discussions or feature specifications (secondary sources of information). Furthermore, the realities of software development mean that these secondary sources may instead mislead the agent towards investigating parts of the codebase that are actually not broken as misinterpretations of the bug by human developers may be reflected in the discussions and feature specifications may be outdated even during feature implementation.

4.2 Marginal Tool Utility

Using the judge LLM, we first determine the sign of marginal tool utility of each tool call in each trajectory. We also analyze the output rationales from the judge LLM for its classifications. Then, we calculate the aggregate tool utility of each tool across all trajectories grouped by model and variant.

In Section 4.1, we see that Grafana/Loki affects accuracy when added/removed from the tool suite, and that neither Mattermost nor Plane do. We thus expect the following aggregate tool utilities:

$$\begin{cases} \sum_{\text{grafana}} \Delta_\alpha(\alpha_i) > 0 \\ \sum_{\text{mattermost}} \Delta_\alpha(\alpha_i) \leq 0 \\ \sum_{\text{plane}} \Delta_\alpha(\alpha_i) \leq 0 \end{cases} \tag{9}$$

Investigating the 50 trajectories belonging to the `default` variant (25 generated by GPT-5.3-Codex and another 25 by Gemini 3.1 Pro), the marginal tool utility results in Tables 3 and 4 are consistent with that expectation, supporting our proposed definition and implementation. For `default` with GPT-5.3-Codex, the aggregate tool utility of Grafana/Loki is $+25$, Mattermost is -35, and Plane is -30. For `default` with Gemini 3.1 Pro, the aggregate tool utility of Grafana/Loki is $+5$, Mattermost is -17, and Plane is -28. For `grafana`, the aggregate tool utility of Grafana/Loki is $58 - 29 = +29$ with GPT-5.3-Codex and $23 - 20 = +3$ with Gemini 3.1 Pro (no corresponding tables).

In determining the signs of marginal tool utilities, the judge was prompted to output its rationale. The most common rationales to justify $\Delta_\alpha > 0$ classifications are as follows: Grafana/Loki logs exposed concrete failures (exact error logs that hint at the likely fix); more targeted queries for Grafana/Loki logs at later steps narrowed the investigation and provided more focused signals; and Mattermost and Plane context helped orient the agent towards useful conversation and/or requested feature context. To justify $\Delta_\alpha \leq 0$ classifications: generic, irrelevant, or noisy context from logs, discussions, and specifications distracted the agent; malformed queries by the agent wasted steps on retries; and valid

queries but those resulting in empty outputs wasted steps on pivoting to valid queries that actually gave non-empty outputs.

Table 3: Marginal tool utility signs of each tool in the `default` harness for GPT-5.3-Codex.

Tool	$\Delta_\alpha > 0$	$\Delta_\alpha \leq 0$	Mean Confidence ($\Delta_\alpha > 0$)	Mean Confidence ($\Delta_\alpha \leq 0$)
Grafana/Loki	52	27	0.819	0.890
Mattermost	7	42	0.734	0.927
Plane	23	53	0.775	0.928

Table 4: Marginal tool utility signs of each tool in the `default` harness for Gemini 3.1 Pro.

Tool	$\Delta_\alpha > 0$	$\Delta_\alpha \leq 0$	Mean Confidence ($\Delta_\alpha > 0$)	Mean Confidence ($\Delta_\alpha \leq 0$)
Grafana/Loki	26	21	0.825	0.870
Mattermost	3	20	0.673	0.930
Plane	3	31	0.607	0.917

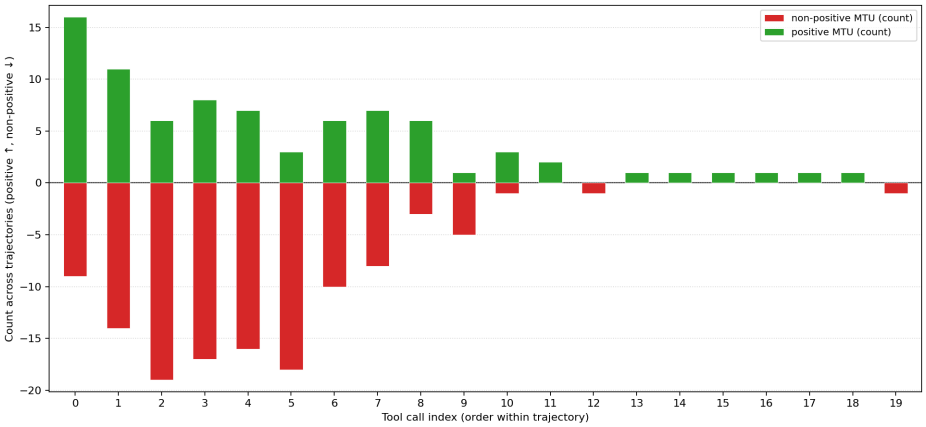

Figure 1: Marginal tool utility signs across all `default` trajectories by GPT-5.3-Codex. Different agent trajectories have different lengths, hence the reduction in total tool call count along the horizontal axis.

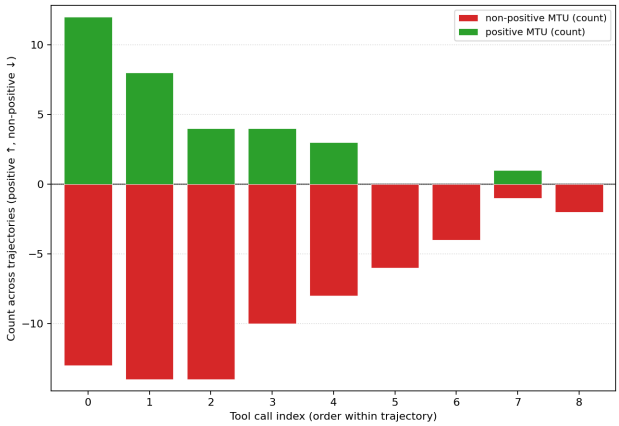

Figure 2: Marginal tool utility signs across all `default` trajectories by Gemini 3.1 Pro. Different agent trajectories have different lengths, hence the reduction in total tool call count along the horizontal axis.

As seen in Figures 1 and 2, tool calls with $\Delta_\alpha > 0$ occur more often near the beginning of trajectories, and tool calls with $\Delta_\alpha \leq 0$ occur more often in the middle. Analyzing the agent trajectories in detail, most early calls are high-signal Grafana/Loki calls (so are most early positive marginal tool utility calls), middle calls are mostly spent querying noisy sources and performing broad conversation/specification discovery, and later calls generally surface more specific evidence to support the agent's code patch generation.

4.3 Tool Efficiency

Given the signs of marginal tool utility determined by the judge LLM for each tool call, we calculate tool efficiency for each trajectory in `default` and `grafana`. Then, we calculate the mean tool efficiency grouped by model and variant. Since the `no-mcp` variant has no MCP tools, tool efficiency is undefined for that particular variant. Our results are found in Tables 5 and 6.

Table 5: Mean tool efficiency for GPT-5.3-Codex out of 25 agent trajectories per variant. Included are the number of useful tool calls and total number of tool calls from all trajectories combined; the quotient of these two values is not mathematically equivalent to mean tool efficiency as agent trajectories vary in length.

Variant	Useful Call Count (All)	Total Call Count (All)	Mean Tool Efficiency
`default`	82	204	0.359
`grafana`	58	87	0.720

Table 6: Mean tool efficiency for Gemini 3.1 Pro out of 25 agent trajectories per variant. Included are the number of useful tool calls and total number of tool calls from all trajectories combined; the quotient of these two values is not mathematically equivalent to mean tool efficiency as agent trajectories vary in length.

Variant	Useful Call Count (All)	Total Call Count (All)	Mean Tool Efficiency
`default`	32	104	0.367
`grafana`	23	43	0.593

Tool efficiency increases when Mattermost and Plane are removed. This finding is also consistent with the conclusion that these two MCP tools are unnecessarily included in the agent's tool suite and thus can be considered redundant given the nature of the task to solve.

5 Discussion

We identify several promising directions for future work the community can engage in that leverages marginal tool utility and tool efficiency.

Minimizing sub-optimal middle calls. The trend that middle calls are often sub-optimal (see Section 4.2) is a useful finding that may inspire future LLM reinforcement learning techniques, similar to those by Ma et al. (2026); Shao et al. (2024); Ouyang et al. (2022); Uesato et al. (2022), with marginal tool utility as a component of the reward function. Alternatively, marginal tool utility can be used for inference-time optimizations, like designing agents that can backtrack (Yang et al., 2025) should the trajectory thus far show some number of consecutive tool calls with non-positive marginal tool utility.

Designing leaner tool suites. As seen through our tool ablations of APEX-SWE Observability, tool suites can be more bloated than necessary, as has been surfaced by Liu et al. (2025) albeit from a different perspective. The tool efficiency metric can be directly used to assess the overall health of a tool suite such that it remains as lean and maintainable as possible, and the aggregate tool utility of each tool to decide whether to preserve the tool in improved versions of the harness. This enables the creation of tool suites that contain tools that serve mutually exclusive purposes. Tool descriptions and/or execution bodies (Lu et al., 2026) alone may not indicate redundancy between tools, as we have seen in Tables 1 and 2, hence marginal tool utility and tool efficiency can serve as extra sources of information to uncover such redundancies.

More informed harness engineering. We observe in Section 4 that using different backbone models for agents can yield the same task accuracy with different tool efficiencies. We posit that this is primarily a product of different model training methodologies as the agent harness remains a constant variable, but an important finding to note nonetheless as this supports the view that harness engineering must be informed by each agent's particular backbone model. In other words, a one-size-fits-all approach to harness engineering may be sub-optimal, hence workflows that are designed to automate harness engineering (Lee et al., 2026) can benefit from our proposed metrics.

Self-improving agents indexing on tool efficiency. Taking automated harness engineering one step further, the agents themselves can recursively self-improve (Xia et al., 2025; Xiong et al., 2026) without relying on another agent that optimizes the harness offline (Cai et al., 2024; Wölflein et al., 2025). Not only can this online improvement be based on the usual metrics like task accuracy, tool efficiency and marginal tool utility can be leveraged mid-execution to provide rich reward signals to optimize context (including tool descriptions, input schemas, and output data shapes) or update model weights.

6 Limitations

While confidence scores returned by an LLM as decoded tokens are not the most rigorous (Xiong et al., 2024; Yang et al., 2026), the judge's confidence score per classification in Tables 3 and 4 indicate that the judge is generally more confident on non-positive classifications. We did not investigate why this is so, nor did we attempt to balance the confidence scores, as we view this work as outside the scope of this paper. Alternatively, a different LLM-as-a-Judge implementation could have been used, such as one replacing the language modeling head of the judge with a binary classification head (Ma et al., 2026), though this requires using an open-weight LLM.

Most notably, we only ran tool ablations for read-only tools (the three MCPs are all read-only). We did not remove tools with write privileges (e.g., apply_patch) from the agent's tool suite as we suspect that that would have prevented the agent from completing the tasks at all. Having said that, we seek to find ways in which we can assess the marginal tool utility of write tool calls, either directly or indirectly (the latter case by investigating how read-only tool calls are affected by what was written using the write tool calls), in future experiments.

7 Conclusion

We introduce tool efficiency and marginal tool utility, both being new quantitative metrics that are relevant in determining the usefulness of tools and thus in constructing the leanest possible tool suite for an LLM agent given a particular task. Our findings suggest that intuitive judgments regarding a tool's usefulness as well as empirical tool ablation results using accuracy as a proxy in discerning usefulness/necessity of different tools agree with conclusions drawn from analyzing marginal tool utility and tool efficiency values.

All in all, we look forward to what our new definitions enable in the development of LLMs and LLM agents. We hope that the literature matures not only in terms of new methods, architectures, and workflows, but also in new evaluation dimensions against which we can measure the holistic quality of these new artifacts. As well-defined quantitative metrics are more tractable to optimize for, we believe that evaluation/metrics research will accelerate progress in LLM research and engineering further, providing the necessary direction and focus guiding iteration cycles of novel methods.

Societal Impacts. As large language models and agents built on them become more capable, many have raised concerns regarding our over-reliance, even dependence, on them. While this work does not introduce a new artifact directly relevant to such concerns, this work does enable new artifacts to be more capable than their existing versions today. We want to acknowledge that, and it is our hope and belief that agents and humans are able to work in tandem, each tackling a dimension of work the other is less-suited for. The onus is on us, those at the frontier of the development of this technology, to understand and thus educate the public on what those dimensions are.

Acknowledgments and Disclosure of Funding

The author conducted this work in her capacity as Founding Research Engineer at Foam. The author thanks Perla Gamez and Luke Mercado for their support throughout the research process. All funding for this work comes from Foam.

References

Acikgoz, E. C., Qian, C., Hübotter, J., Ji, H., Hakkani-Tür, D., and Tur, G. (2026). Tool-r0: Self-evolving llm agents for tool-learning from zero data.

Agarwal, M., Abdelaziz, I., Basu, K., Unuvar, M., Lastras, L. A., Rizk, Y., and Kapanipathi, P. (2026). Toolrm: Outcome reward models for tool-calling large language models.

Ahn, J., Verma, R., Lou, R., Liu, D., Zhang, R., and Yin, W. (2024). Large language models for mathematical reasoning: Progresses and challenges.

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Ho, D., Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jang, E., Ruano, R. J., Jeffrey, K., Jesmonth, S., Joshi, N. J., Julian, R., Kalashnikov, D., Kuang, Y., Lee, K.-H., Levine, S., Lu, Y., Luu, L., Parada, C., Pastor, P., Quiambao, J., Rao, K., Rettinghouse, J., Reyes, D., Sermanet, P., Sievers, N., Tan, C., Toshev, A., Vanhoucke, V., Xia, F., Xiao, T., Xu, P., Xu, S., Yan, M., and Zeng, A. (2022). Do as i can, not as i say: Grounding language in robotic affordances.

Anthropic (2025). Introducing advanced tool use on the claude developer platform.

Athiwaratkun, B., Gouda, S. K., Wang, Z., Li, X., Tian, Y., Tan, M., Ahmad, W. U., Wang, S., Sun, Q., Shang, M., Gonugondla, S. K., Ding, H., Kumar, V., Fulton, N., Farahani, A., Jain, S., Giaquinto, R., Qian, H., Ramanathan, M. K., Nallapati, R., Ray, B., Bhatia, P., Sengupta, S., Roth, D., and Xiang, B. (2023). Multi-lingual evaluation of code generation models.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., and Sutton, C. (2021). Program synthesis with large language models.

Bandi, C., Hertzberg, B., Boo, G., Polakam, T., Da, J., Hassaan, S., Sharma, M., Park, A., Hernandez, E., Rambado, D., Salazar, I., Cruz, R., Rane, C., Levin, B., Kenstler, B., and Liu, B. (2026). Mcp-atlas: A large-scale benchmark for tool-use competency with real mcp servers.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners.

Cai, T., Wang, X., Ma, T., Chen, X., and Zhou, D. (2024). Large language models as tool makers.

Chaturvedi, S. S., Bergerson, J., and Mallick, T. (2026). Toward reliable, safe, and secure llms for scientific applications.

Chen, L., Zaharia, M., and Zou, J. (2024). FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*. Featured Certification.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. (2021). Evaluating large language models trained on code.

Cheng, J., Marone, M., Weller, O., Lawrie, D., Khashabi, D., and Durme, B. V. (2024). Dated data: Tracing knowledge cutoffs in large language models. In *First Conference on Language Modeling*.

de Souza, H. A., Chaim, M. L., and Kon, F. (2017). Spectrum-based software fault localization: A survey of techniques, advances, and challenges.

Erdogan, L. E., Lee, N., Jha, S., Kim, S., Tabrizi, R., Moon, S., Hooper, C., Anumanchipalli, G., Keutzer, K., and Gholami, A. (2024). Tinyagent: Function calling at the edge.

Google (2026). Gemini 3.1 pro: A smarter model for your most complex tasks.

Guo, C., Patel, M., Hartmann, B., Zamfirescu-Pereira, J., Chasins, S., and Ranade, G. (2025). Unspoken logic: Understanding and bridging the gap between free-form and LLM-interpretable natural language mathematical proofs. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*.

Guo, Z., Cheng, S., Wang, H., Liang, S., Qin, Y., Li, P., Liu, Z., Sun, M., and Liu, Y. (2024). Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models.

Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. (2022). Language models as zero-shot planners: Extracting actionable knowledge for embodied agents.

Huang, Y., Shi, J., Li, Y., Fan, C., Wu, S., Zhang, Q., Liu, Y., Zhou, P., Wan, Y., Gong, N. Z., and Sun, L. (2024). Metatool benchmark for large language models: Deciding whether to use tools and which to use.

Jiang, H., Wu, Q., Lin, C.-Y., Yang, Y., and Qiu, L. (2023). LLMLingua: Compressing prompts for accelerated inference of large language models. In Bouamor, H., Pino, J., and Bali, K., editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376, Singapore. Association for Computational Linguistics.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. (2024). Swe-bench: Can language models resolve real-world github issues?

Jin, B., Zeng, H., Yue, Z., Yoon, J., Arik, S. O., Wang, D., Zamani, H., and Han, J. (2025). Search-r1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*.

Kottamasu, A., Mahapatra, C., Lee, S., Pan, B., Barthwal, A., Datta, A., Gupta, A., Mehta, P., Arun, A., Alberti, S., Hiremath, A., Foody, B., and Vidgen, B. (2026). Apex-swe.

Lee, Y., Nair, R., Zhang, Q., Lee, K., Khattab, O., and Finn, C. (2026). Meta-harness: End-to-end optimization of model harnesses.

Li, J., Zhao, W., Zhao, J., Zeng, W., Wu, H., Wang, X., Ge, R., Cao, Y., Huang, Y., Liu, W., Liu, J., Su, Z., Guo, Y., Zhou, F., Zhang, L., Michelini, J., Wang, X., Yue, X., Zhou, S., Neubig, G., and He, J. (2025a). The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution.

Li, X., Zou, H., and Liu, P. (2025b). Torl: Scaling tool-integrated rl.

Liu, M. M., Garcia, D., Parllaku, F., Upadhyay, V., Shah, S. F. A., and Roth, D. (2025). Toolscope: Enhancing llm agent tool use through tool merging and context-aware filtering.

Lou, X., Lázaro-Gredilla, M., Dedieu, A., Wendelken, C., Lehrach, W., and Murphy, K. P. (2026). Autoharness: improving llm agents by automatically synthesizing a code harness.

Lu, J., Kong, Z., Wang, Y., Fu, R., Wan, H., Yang, C., Lou, W., Sun, H., Wang, L., Jiang, Y., Wang, X., Sun, X., and Zhou, D. (2026). Beyond static tools: Test-time tool evolution for scientific reasoning.

Ma, D., Yang, Z., Xu, H., Fang, H., Yu, K., and Chen, L. (2026). Empowering LLM tool invocation with tool-call reward model. In *The Fourteenth International Conference on Learning Representations*.

Manem, C., Brahma, P. P., Mishra, P., Liu, Z., and Barsoum, E. (2025). SAND-math: Using LLMs to generate novel, difficult and useful mathematics questions and answers. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*.

Merrill, M. A., Shaw, A. G., Carlini, N., Li, B., Raj, H., Bercovich, I., Shi, L., Shin, J. Y., Walshe, T., Buchanan, E. K., Shen, J., Ye, G., Lin, H., Poulos, J., Wang, M., Nezhurina, M., Jitsev, J., Lu, D., Mastromichalakis, O. M., Xu, Z., Chen, Z., Liu, Y., Zhang, R., Chen, L. L., Kashyap, A., Uslu, J.-L., Li, J., Wu, J., Yan, M., Bian, S., Sharma, V., Sun, K., Dillmann, S., Anand, A., Lanpouthakoun, A., Koopah, B., Hu, C., Guha, E., Dreiman, G. H. S., Zhu, J., Krauth, K., Zhong, L., Muennighoff, N., Amanfu, R., Tan, S., Pimpalgaonkar, S., Aggarwal, T., Lin, X., Lan, X., Zhao, X., Liang, Y., Wang, Y., Wang, Z., Zhou, C., Heineman, D., Liu, H., Trivedi, H., Yang, J., Lin, J., Shetty, M., Yang, M., Omi, N., Raoof, N., Li, S., Zhuo, T. Y., Lin, W., Dai, Y., Wang, Y., Chai, W., Zhou, S., Wahdany, D., She, Z., Hu, J., Dong, Z., Zhu, Y., Cui, S., Saiyed, A., Kolbeinsson, A., Hu, J., Rytting, C. M., Marten, R., Wang, Y., Dimakis, A., Konwinski, A., and Schmidt, L. (2026). Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces.

Moura, A. (2025). Antl3x/toolrag: Unlimited llm tools, zero context penalties - toolrag serves exactly the llm tools your user-query demands.

Ning, K., Su, Y., Lv, X., Zhang, Y., Liu, J., Liu, K., and Xu, J. (2024). Wtu-eval: A whether-or-not tool usage evaluation benchmark for large language models.

OpenAI (2026a). Introducing gpt-5.3-codex | openai.

OpenAI (2026b). Introducing gpt-5.4 | openai.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. (2022). Training language models to follow instructions with human feedback.

Paranjape, B., Lundberg, S., Singh, S., Hajishirzi, H., Zettlemoyer, L., and Ribeiro, M. T. (2023). Art: Automatic multi-step reasoning and tool-use for large language models.

Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., Zhao, S., Tian, R., Xie, R., Zhou, J., Gerstein, M., Li, D., Liu, Z., and Sun, M. (2023). Toolllm: Facilitating large language models to master 16000+ real-world apis.

Russell, S. J. and Norvig, P. (2020). *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools.

Serway, R. A., Jewett, J. W., and Peroomian, V. (2012). *Physics for scientists and engineers with modern physics*. Brooks/Cole, Cengage Learning.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. (2024). Deepseekmath: Pushing the limits of mathematical reasoning in open language models.

Sivakumaran, N., Chen, J., Wan, D., Zhang, Y., Yoon, J., Stengel-Eskin, E., and Bansal, M. (2026). DART: Leveraging multi-agent disagreement for tool recruitment in multimodal reasoning. In Demberg, V., Inui, K., and Marquez, L., editors, *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5445–5464, Rabat, Morocco. Association for Computational Linguistics.

Uesato, J., Kushman, N., Kumar, R., Song, F., Siegel, N., Wang, L., Creswell, A., Irving, G., and Higgins, I. (2022). Solving math word problems with process- and outcome-based feedback.

Wei, J., Sun, Z., Papay, S., McKinney, S., Han, J., Fulford, I., Chung, H. W., Passos, A. T., Fedus, W., and Glaese, A. (2025). Browsecomp: A simple yet challenging benchmark for browsing agents.

Wu, B., Meij, E., and Yilmaz, E. (2025). A joint optimization framework for enhancing efficiency of tool utilization in LLM agents. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T., editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22361–22373, Vienna, Austria. Association for Computational Linguistics.

Wölflein, G., Ferber, D., Truhn, D., Arandjelović, O., and Kather, J. N. (2025). Llm agents making agent tools.

Xia, P., Zeng, K., Liu, J., Qin, C., Wu, F., Zhou, Y., Xiong, C., and Yao, H. (2025). Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning.

Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T. J., Cheng, Z., Shin, D., Lei, F., Liu, Y., Xu, Y., Zhou, S., Savarese, S., Xiong, C., Zhong, V., and Yu, T. (2024). Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments.

Xiong, M., Hu, Z., Lu, X., Li, Y., Fu, J., He, J., and Hooi, B. (2024). Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms.

Xiong, Y., Hu, S., and Clune, J. (2026). Learning to continually learn via meta-learning agentic memory designs.

Yang, S.-H., Wu, C.-K., Lin, C.-Y., Chen, Y.-N., yi Lee, H., and Sun, S.-H. (2026). On calibration of large language models: From response to capability.

Yang, X.-W., Zhu, X.-Y., Wei, W.-D., Zhang, D.-C., Shao, J.-J., Zhou, Z., Guo, L.-Z., and Li, Y.-F. (2025). Step back to leap forward: Self-backtracking for boosting reasoning of language models.

Yao, S., Rao, R., Hausknecht, M., and Narasimhan, K. (2020). Keep calm and explore: Language models for action generation in text-based games.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2022). React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Yu, B., Baker, F. N., Chen, Z., Herb, G., Gou, B., Adu-Ampratwum, D., Ning, X., and Sun, H. (2025). Chemtoolagent: The impact of tools on language agents for chemistry problem solving.

Yu, F. (2025). When ais judge ais: The rise of agent-as-a-judge evaluation for llms.

Zhang, A. L., Kraska, T., and Khattab, O. (2026). Recursive language models.

Zhang, S., Zhang, J., Liu, J., Song, L., Wang, C., Krishna, R., and Wu, Q. (2024). Offline training of language model agents with functions as learnable weights.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., Zhang, H., Gonzalez, J. E., and Stoica, I. (2023). Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Zhuang, Z., Chen, J., Xu, H., Jiang, Y., and Lin, J. (2025). Large language models for automated scholarly paper review: A survey. *Information Fusion*, 124:103332.

A Prompt for LLM-As-A-Judge for Marginal Tool Utility Sign

System Prompt

You are an expert evaluator of agent trajectories completing a SWE bugfix observability task.

Your task is to determine whether a tool call increases the likelihood the bugfix task is solved correctly. This is measured by a metric called marginal tool utility (MTU).

Concretely, you are tasked to determine the sign of marginal tool utility (MTU) for one tool call.

Definition:
MTU(i) = p(task solved correctly | tool calls 0..i) - p(task solved correctly | tool calls 0..i-1)

Equivalently:
MTU(i) is positive if the likelihood of the bugfix task being solved correctly strictly increases after tool call i is added into the trajectory. MTU(i) is non-positive otherwise.

You are given:

- BEFORE context: state of trajectory before tool call i executes.
- AFTER context: state of trajectory after tool call i completes.
- The specific tool call and tool result.

Output strict JSON only:
{
"label": "positive" | "non_positive",
"confidence": float between 0 and 1,
"rationale": "brief explanation"
}

Use "positive" only when the call clearly increases solve likelihood.
Use "non_positive" when likelihood is unchanged or reduced.

User Prompt

Decide MTU sign for this tool call.

Target tool call:

- tool_call_id: {TOOL_CALL_ID}
- tool_name: {TOOL_NAME}
- arguments: {ARGUMENTS_TRUNCATED}
- tool_result_excerpt: {TOOL_RESULT}

BEFORE context (through tool calls 0..i-1 completion):
=== BEFORE START ===
{BEFORE_CONTEXT}
=== BEFORE END ===

AFTER context (through tool calls 0..i completion):
=== AFTER START ===
{AFTER_CONTEXT}
=== AFTER END ===

Return strict JSON only.

B APEX-SWE Observability Per-Task Correctness Breakdown

Task	GPT-5.3-Codex			Gemini 3.1 Pro		
	default	grafana	no-mcp	default	grafana	no-mcp
T1	0	0	0	0	0	0
T2	0	0	0	0	0	0
T3	1	1	0	1	1	1
T4	1	1	0	1	1	1
T5	0	0	0	0	1	0
T6	0	0	0	0	0	0
T7	1	1	1	1	1	1
T8	0	0	0	0	0	0
T9	1	1	1	1	1	0
T10	0	0	1	1	0	0
T11	0	0	0	0	0	0
T12	0	0	0	0	0	0
T13	0	0	0	0	0	0
T14	0	0	0	0	0	0
T15	0	0	0	0	0	0
T16	0	0	0	0	0	0
T17	0	0	0	1	0	0
T18	1	1	0	1	1	0
T19	0	1	0	0	1	0
T20	1	1	1	1	1	1
T21	0	0	0	0	0	0
T22	0	0	0	0	0	0
T23	1	1	1	1	1	1
T24	1	1	1	0	0	0
T25	0	0	0	0	0	0

TASK LEGEND

T1 0xpolygon-bor-1710-observability	T2 0xpolygon-bor-1728-1748-observability	T3 0xpolygon-bor-1743-observability	T4 chainsafe-gossamer-4286-4720-observability
T5 chainsafe-gossamer-4489-4640-observability	T6 chainsafe-gossamer-4573-4683-observability	T7 containers-podman-compose-1221-1231-observability	T8 containers-podman-compose-1226-1243-observability
T9 containers-podman-compose-2-1238-observability	T10 containers-podman-compose-23-1214-observability	T11 ethereum-optimism-op-geth-553-558-observability	T12 ethereum-optimism-op-geth-655-observability
T13 ethereum-optimism-op-geth-666-observability	T14 ethereum-optimism-op-geth-675-observability	T15 ethereum-optimism-op-geth-679-680-observability	T16 git-bug-git-bug-132-449-observability
T17 git-bug-git-bug-1367-1370-observability	T18 git-bug-git-bug-264-274-observability	T19 git-bug-git-bug-338-341-observability	T20 paperless-ngx-paperless-ngx-10195-10196-observability
T21 paperless-ngx-paperless-ngx-10555-observability	T22 paperless-ngx-paperless-ngx-5228-10559-observability	T23 paperless-ngx-paperless-ngx-6341-9305-observability	T24 paperless-ngx-paperless-ngx-8910-8912-observability
T25 paperless-ngx-paperless-ngx-9784-observability			

Figure 3: Full score breakdown per task. A score of 1 indicates a pass, while a score of 0 indicates a fail.