Do Agents Know What They Can't Do? Evaluating Feasibility Awareness in Tool-Using Agents

Anonymous ACL submission

Abstract

Tool-using agents often incur substantial computational cost due to long reasoning chains and iterative tool usage. In practical scenarios, many tasks become *infeasible* under constrained tool environments, where the capabilities required for successful task completion are unavailable. Detecting infeasible tasks and stopping execution early can significantly reduce unnecessary execution cost. In this work, we propose **FeasiGen**, an automatic pipeline for constructing infeasible agent tasks by identifying the critical tools required for successful task completion. Our approach extracts tool-calling traces from successful executions across multiple agent systems, identifies critical tools consistently shared across diverse execution strategies, and masks these tools to automatically transform solvable tasks into infeasible ones. Human verification confirms that the infeasibility annotations for our constructed tasks achieve over 94% accuracy. We further introduce feasibility-aware evaluation metrics for measuring whether agents can recognize infeasible tasks and stop execution appropriately. Extensive evaluations across nine models reveal substantially weak infeasibility detection ability, with false continue rate reaching up to 73.9%. We further observe that multi-agent architectures significantly reduce erroneous execution under infeasible conditions[1].

1 Introduction

Tool-using agents have emerged as a central paradigm in modern AI systems, enabling large language models (LLMs) to solve complex tasks through external APIs, environment interaction, and multi-step planning (Patil et al., 2024; Schick et al., 2023; Yao et al., 2022). Agent execution often incurs substantial computational and interaction costs due to long reasoning chains, iterative tool usage, and complex multi-step workflows (Liu et al., 2024; Zhou et al., 2024; Jimenez et al., 2024). However, in practical deployments scenarios, many tasks become ***infeasible***, i.e., the required capabilities for successful task completion are unavailable under the current tool and environment constraints (Zhang et al., 2024; Qin et al., 2024; Guo et al., 2024). Detecting such infeasible tasks and terminating execution early can substantially reduce unnecessary computational overhead (Liu et al., 2024; Jimenez et al., 2024; Kapoor et al., 2024). As illustrated in Figure 1, when the required `Payment_API` is unavailable (i.e., task is infeasible), early infeasibility detection and termination reduce execution cost by more than 10× compared with failed execution after continued reasoning and tool interaction.

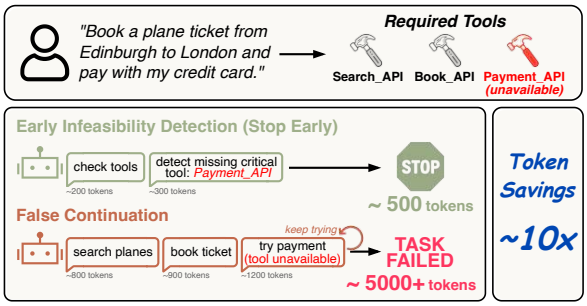

Figure 1: An example of early infeasibility detection vs. false continuation.

However, existing agent benchmarks largely assume that tasks remain solvable under the provided environments and primarily evaluate task success rates under such settings (Patil et al., 2025; Liu et al., 2024; Zhou et al., 2024; Jimenez et al., 2024; Yao et al., 2024; Xie et al., 2024; Deng et al., 2023). Current evaluation paradigms also primarily reward persistent execution and successful task completion, while providing limited assessment of whether agents can recognize violated execution requirements and terminate appropriately under constrained environments (Zhang et al., 2024; Bonagiri et al., 2025).

Constructing genuinely infeasible agent tasks

[1] Our code and data will be released upon acceptance.

also remains challenging. Existing approaches still mainly rely on human verification. Manual-annotation methods such as ToolBeHonest (Zhang et al., 2024) require domain experts to explicitly inspect task requirements and annotate violated execution conditions, making the construction process labor-intensive and difficult to scale. Yang et al. (2024) propose perturbation-based methods for constructing incomplete tool-use scenarios by modifying or replacing tools with semantically similar alternatives. However, these perturbations are not targeted at removing the critical capabilities required for successful task completion, making it difficult to guarantee that the resulting tasks are genuinely infeasible. Moreover, their pipeline still requires substantial human annotation to filter valid infeasible instances.

In this work, we introduce **FeasiGen**, an automatic pipeline for constructing infeasible agent tasks by identifying critical execution dependencies required for successful task completion. Our framework operates on agent tasks with predefined candidate tool pools, where execution is constrained to a fixed set of available tools. We perform multi-model execution trace analysis by collecting successful tool-calling traces from multiple agent systems and identifying dependencies consistently shared across diverse execution strategies. By masking these critical dependencies, our framework automatically transforms previously solvable tasks into infeasible ones under constrained tool environments. Human verification on a statistically grounded subset (95% confidence, 5% margin of error) confirms that **94%** of the constructed instances are truly infeasible under the corresponding constrained environments.

Furthermore, we introduce a feasibility-aware evaluation protocol that explicitly measures whether agents can recognize infeasible tasks and stop execution appropriately. Beyond conventional task-success evaluation, our metrics assess false continue rate on infeasible tasks as well as the efficiency trade-off between early stopping and failed execution, enabling systematic evaluation of both feasibility awareness and execution efficiency. To our knowledge, this is the first work that automatically constructs infeasible agent tasks through removing critical required capability and explicitly incorporates infeasibility detection into agent evaluation.

Using our pipeline, we construct a feasibility-aware benchmark containing infeasible agent tasks drawn from four agent datasets that span diverse domains, and evaluate nine modern models under both single-agent and multi-agent architectures. Experimental results reveal that current agent systems generally exhibit weak infeasibility detection ability, with false continue rate ranging from 23.5% to 73.9%. We further observe that multi-agent architectures substantially improve infeasibility detection, reducing the average false continue rate from 54.6% to 17.5%. In addition, enabling reasoning mode consistently improves infeasibility detection and reduces unnecessary token consumption through earlier stopping behavior. Finally, we identify signs of datasets overfitting in several models, which are able to generate correct answers even when the critical capabilities required for task completion are unavailable.

Our work makes the following contributions:

- We propose FeasiGen, an automatic pipeline for constructing infeasible agent tasks by identifying critical execution dependencies required for successful task completion and masking the corresponding capabilities under constrained environments.
- We introduce feasibility-aware evaluation metrics for systematically measuring infeasibility detection, false continuation behavior, and execution efficiency beyond conventional task success rate evaluation.
- We construct a benchmark drawn from four agent datasets spanning diverse domains, and conduct evaluations across nine LLMs under both single-agent and multi-agent architectures, revealing limitations in current agents' infeasibility detection capabilities.

2 Background

Tool-Using Agent Benchmarks. LLMs have increasingly been extended as tool-using agents that interact with external environments through API calls and multi-step decision making (Yao et al., 2022; Schick et al., 2023; Patil et al., 2024). Building on this paradigm, recent benchmarks evaluate function calling, tool usage, and agent execution across diverse environments, including BFCL (Patil et al., 2025), API-Bank (Li et al., 2023), StableToolBench (Guo et al., 2024), and τ-bench (Yao et al., 2024). These benchmarks primarily focus on execution correctness, task completion, and efficiency under solvable environments, while largely

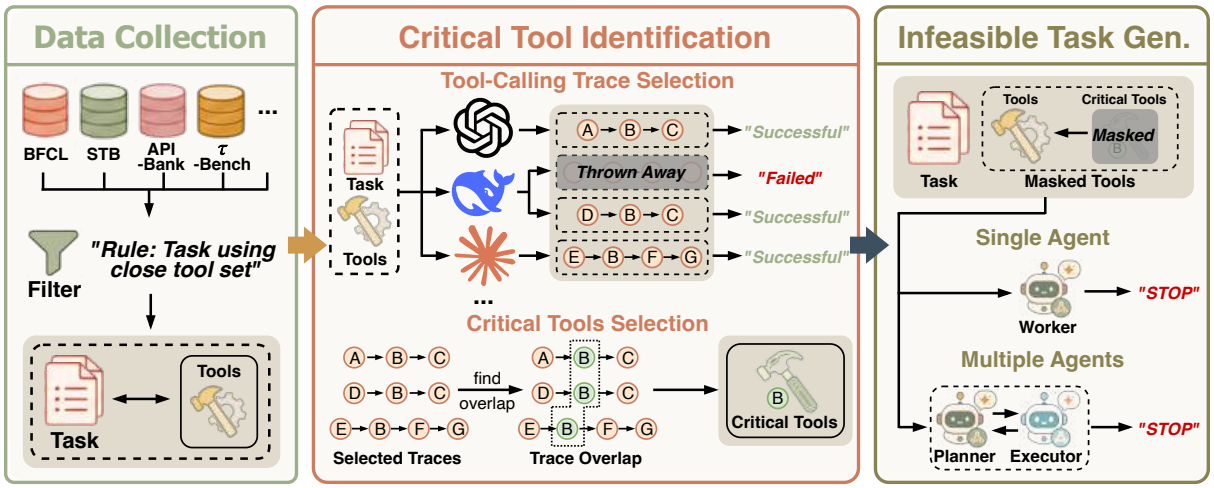

Figure 2: Overview of the FeasiGen pipeline. **Step 1:** Collect tasks operating over a closed tool set from existing datasets. **Step 2:** Each task is run by various agents, and we identify the critical tools as the intersection of tools shared across successful traces. **Step 3:** Mask these tools to render the task infeasible for evaluation.

assuming that tasks remain achievable given the provided tools and environmental constraints. As a result, they rarely consider scenarios where tasks may become inherently infeasible due to missing critical capabilities.

Infeasible Tasks and Feasibility-Aware Evaluation. In realistic deployments, many agent tasks become *infeasible* when the critical capabilities required for successful task completion are unavailable under the current environment. For example, a payment-processing task becomes infeasible once the required payment API is unavailable, even if other related tools remain accessible. Existing evaluation paradigms mainly reward persistent execution and final task success, potentially overlooking brittle behaviors where agents continue reasoning and invoking tools even after successful task completion has become impossible, resulting in severe token waste.

Recent work (Yang et al., 2024) introduces perturbation strategies by adding or replacing APIs with semantically similar alternatives, or by removing information from user instructions. However, these methods do not identify the indispensable capabilities truly required for task completion. As a result, although such perturbations may increase task difficulty, they cannot guarantee that the constructed tasks are genuinely infeasible, since agents may still solve them through alternative tools or different execution strategies. Consequently, additional human annotation is still required to determine whether the constructed tasks are infeasible.

Critical Dependency Identification. Related ideas of identifying indispensable components through intersections across execution traces have been widely explored in program analysis, workflow mining, and fault localization (Weiser, 1984; Jones et al., 2002; Agrawal et al., 1998; van der Aalst et al., 2004). In these settings, components consistently shared across successful execution traces are often treated as critical dependencies required for successful execution. Similar intuitions also appear in invariant dependency discovery and causal analysis (Arjovsky et al., 2019; Peters et al., 2016), where factors repeatedly shared across diverse successful outcomes are more likely to correspond to essential dependencies rather than strategy-specific behaviors. These observations suggest that execution patterns consistently shared across diverse successful trajectories can provide useful signals for identifying the critical capabilities required for task completion.

3 Infeasible Agent Task Construction

As shown in Figure 2, our FeasiGen pipeline contains three stages. First, we collect tasks from existing agent datasets and retain only those operating over a closed tool set (§3.1). Second, we run various agent systems on each task, keep only the successful execution traces, and identify *critical tools* as the intersection of tools shared across all successful traces (§3.2). Third, we mask these critical tools from the candidate tool pool to make each task infeasible. (§3.3).

3.1 Data Collection

We collect candidate tasks from multiple tool-use agent datasets spanning diverse domains, and retain only tasks operating over a closed tool set, where each task is associated with a fixed and fully predefined set of available tools. Formally, each instance is represented as $(q_i, \mathcal{T}_i)$, where q_i denotes the user query and $\mathcal{T}_i$ denotes the corresponding candidate tool pool available to the agent. The resulting tasks and their associated tool sets provide a controlled foundation for identifying and masking critical tools in later stages.

3.2 Critical Tool Identification

In this step, we collect tool-calling traces from successful executions, and identify critical tools by analyzing the overlaps across the collected traces.

Selecting tool-calling traces. To identify the execution dependencies required for successful task completion, we execute multiple agent systems on the original solvable tasks and record their successful tool-calling traces. Specifically, for each task q_i, we evaluate a collection of agent systems $\mathcal{M} = \{m_1, m_2, \ldots, m_k\}$. Each agent interacts with the environment using only the provided candidate tool pool $\mathcal{T}_i$. For every successful execution, we collect the corresponding tool-calling trace:

$$\Gamma_i^{(j)} = [t_1, t_2, \ldots, t_n],$$

where $\Gamma_i^{(j)}$ denotes the ordered sequence of tool calls generated by model m_j for task q_i.

Select critical tools. After collecting successful execution traces from multiple agent systems, we identify critical execution dependencies that are consistently required for successful task completion. For each task q_i, we first extract the set of tools appearing in each successful execution trace:

$$S_i^{(j)} = \{t \mid t \in \Gamma_i^{(j)}\}.$$

We then compute the shared tools dependencies across different successful executions:

$$C_i = \bigcap_{j=1}^{k} S_i^{(j)},$$

where C_i represents the set of tools consistently shared across successful execution strategies from different agent systems.

Tools consistently shared across diverse successful execution strategies are regarded as critical tools required for successful task completion.

3.3 Infeasible Tasks Generation

After identifying the critical tools C_i, we mask them from the candidate tool pool to produce a constrained tool environment:

$$\mathcal{T}_i' = \mathcal{T}_i \setminus C_i.$$

Removing these critical tools eliminates the capabilities required for successful task completion, thereby transforming the original solvable task into an infeasible one. Each constructed Task$_{infeasible}$ is used to evaluate whether agents can recognize missing critical tools and terminate with a STOP decision under constrained environments.

We perform human verification on a statistically grounded subset of the constructed tasks. Following (Jang et al., 2025), we determine the verification subset size using Cochran's formula (Cochran, 1977) (95% confidence, 5% margin of error, $p = 0.5$), yielding 283 samples. Among them, over 94% are confirmed to be truly infeasible under the corresponding constrained environments (Appendix A), while additional analysis further verifies that the identified masked tools correspond to genuinely critical execution dependencies rather than arbitrary task-breaking perturbations (Appendix B).

4 Experimental Setup

4.1 Datasets

Following the pipeline in § 3, we construct infeasible tasks (Task$_{infeasible}$) from four closed-tool agent datasets spanning diverse domains and tool environments. BFCL (Patil et al., 2025) covers function-calling tasks ranging from single API invocations to complex multi-step tool chains. StableToolBench (Guo et al., 2024) provides reproducible multi-tool scenarios built on a large real-world API corpus. API-Bank (Li et al., 2023) focuses on tool-augmented reasoning across diverse API categories. τ-bench (Yao et al., 2024) features stateful multi-turn agent interactions in retail and airline domains. Table 1 reports the size of each original dataset (Task$_{original}$) together with the number of constructed Task$_{infeasible}$[2].

4.2 Models

To construct infeasible tasks, we use GPT-5.5, DeepSeek-V4-Pro, and Claude Opus 4.7 (Anthropic, 2026) to generate successful tool-calling traces on the original datasets. Shared critical tools

[2]We retain only tasks that are successfully executed by all LLMs sharing the same critical tools, making Task$_{infeasible}$ smaller than Task$_{original}$.

Dataset	$\text{Task}_{original}$	$\text{Task}_{infeasible}$
BFCL	1,453	445
StableToolBench	500	300
API-Bank	209	184
τ-bench	165	107
Total	**2,327**	**1,036**

Table 1: Statistics of generated infeasible tasks. For each source dataset, we pair original tasks ($\text{Task}_{original}$) with constructed infeasible tasks ($\text{Task}_{infeasible}$).

across these execution traces are then identified and masked to construct $\text{Task}_{infeasible}$.

We evaluate nine SOTA models spanning both proprietary API-based systems and open-source models from the GPT, DeepSeek, Qwen and LLaMA families under both single-agent and multi-agent architectures (planner–executor). The evaluation includes GPT-5.5, GPT-OSS-120B (OpenAI, 2026, 2025), DeepSeek-V4-Flash, DeepSeek-V4-Pro (DeepSeek, 2026), and multiple Qwen3.5 (Qwen, 2025) and Llama3.1 (Meta, 2024) variants, the latter two families being independent of the task construction process, covering both standard and reasoning mode inference settings.

During evaluation, every model is required to judge whether the task is feasible before execution, following the instructions detailed in Appendix F. When required tools are unavailable, the model should respond with an explicit `STOP` signal, regarded as the model's infeasibility detection.

4.3 Metrics

For each task i, let $y_i \in \{\textit{infeasible}, \textit{feasible}\}$ denote the ground-truth feasibility label and $\hat{y}_i \in \{\textit{infeasible}, \textit{feasible}\}$ for the agent's decision. $s_i \in \{0, 1\}$ indicates whether task i was completed successfully.

False Continue Rate (FCR) measures how often the agent continues executing on a $\text{Task}_{infeasible}$ instead of stopping:

$$FCR = P(\hat{y} = \text{feasible} \mid y = \text{infeasible}).$$

A high FCR indicates that the model fails to recognize missing capabilities and wastes resources on impossible tasks.

Success Rate (SR) measures the proportion of $\text{Task}_{original}$ that are successfully completed, and serves as the execution-capability baseline independent of feasibility decisions:

$$SR = \frac{|\{i : y_i = \text{feasible},\ s_i = 1\}|}{|\{i : y_i = \text{feasible}\}|}.$$

Token Cost to Early Stop ($TC_{\text{early-stop}}$) & Token Cost to Task Failure ($TC_{\text{task-failure}}$). Both metrics are defined exclusively over *infeasible* tasks and measure how many tokens are consumed under each outcome. $TC_{\text{early-stop}}$ is the average token cost on $\text{Task}_{infeasible}$ where the agent correctly detects infeasibility and stops early:

$$TC_{\text{early-stop}} = \frac{1}{|\mathcal{S}_{\text{stop}}|} \sum_{i \in \mathcal{S}_{\text{stop}}} \text{tok}(i),$$

where $\mathcal{S}_{\text{stop}}$ denotes the set of $\text{Task}_{infeasible}$ on which the agent correctly detects infeasibility and terminates, and $\text{tok}(i)$ counts all tokens (input and output) consumed up to and including the termination decision. $TC_{\text{task-failure}}$ is the average token cost on $\text{Task}_{infeasible}$ where the agent fails to detect infeasibility and continues executing until the task ultimately fails:

$$TC_{\text{task-failure}} = \frac{1}{|\mathcal{S}_{\text{fail}}|} \sum_{i \in \mathcal{S}_{\text{fail}}} \text{tok}(i),$$

where $\mathcal{S}_{\text{fail}}$ denotes the set of $\text{Task}_{infeasible}$ on which the agent fails to detect infeasibility and continues executing until task failure, and $\mathcal{S}_{\text{stop}} \cup \mathcal{S}_{\text{fail}} = \mathcal{S}$. A large $TC_{\text{task-failure}}/TC_{\text{early-stop}}$ ratio indicates that failing to detect infeasibility wastes substantially more tokens than stopping early.

5 Results

Detecting Infeasible Tasks Remains Challenging for LLM Agents. Table 2a reports task Success Rate (SR) on $\text{Task}_{original}$. Among all evaluated models, DeepSeek-V4-Pro achieves the strongest overall task successful execution performance, demonstrating consistently high task completion rates across all benchmarks. In contrast, τ-bench is substantially more challenging for all models, resulting in noticeably lower success rates compared with the other datasets.

Table 2b reports the False Continue Rate (FCR) when models are evaluated on $\text{Task}_{infeasible}$. The results reveal that detecting infeasible tasks and terminating execution appropriately remains a major challenge for current agent systems. All evaluated models exhibit relatively high FCRs, with even the best-performing model, GPT-5.5, still reaching an average FCR of 23.5%. This indicates that current agents frequently continue execution despite missing critical capabilities required for task completion. Although GPT-5.5 does not obtain the highest SR, its lower FCR suggests that it is more capable of distinguishing solvable tasks from infea-

Model	BFCL	STB	API-Bank	τ-bench	*Avg.*
GPT-OSS-120B	79.8	57.8	50.7	12.1	50.1
GPT-5.5	83.3	55.9	45.9	19.4	51.1
DeepSeek-V4-Flash	86.5	72.0	50.7	17.0	56.5
DeepSeek-V4-Pro	86.2	75.9	59.8	16.4	**59.6**
Qwen3.5-9B	85.5	69.3	57.4	19.4	57.9
Qwen3.5-27B	86.0	66.5	61.2	23.0	59.2
Qwen3.5-35B-A3B	85.6	67.6	57.4	23.0	58.4
Qwen3.5-122B-A10B	85.3	66.1	59.8	20.6	58.0
Qwen3.5-397B-A17B	86.2	65.8	58.4	25.5	59.0

(a) Task Success Rate (%) on Task$_{original}$.

Model	BFCL	STB	API-Bank	τ-bench	*Avg.*
GPT-OSS-120B	25.8	31.0	**15.2**	**25.2**	24.3
GPT-5.5	**23.8**	**13.0**	21.7	35.5	**23.5**
DeepSeek-V4-Flash	29.4	44.3	48.9	99.1	55.4
DeepSeek-V4-Pro	28.8	40.7	55.4	93.5	54.6
Qwen3.5-9B	49.7	68.7	79.9	97.2	73.9
Qwen3.5-27B	27.0	45.0	45.1	78.5	48.9
Qwen3.5-35B-A3B	37.8	61.7	53.3	91.6	61.1
Qwen3.5-122B-A10B	38.4	47.7	59.2	86.0	57.8
Qwen3.5-397B-A17B	25.8	42.0	43.5	67.3	44.7

(b) False Continue Rate (%, lower is better) on Task$_{infeasible}$.

Table 2: Performance across model families under two task settings.

sible ones, rather than blindly attempting execution. We further observe a clear gap between API and open-source models. GPT-5.5 and GPT-OSS-120B both maintain average FCR below 25%, while all Qwen3.5 variants exceed 44%. This suggests that open-source Qwen models are substantially more likely to persist in attempting infeasible tasks, potentially leading to unsafe or unreliable execution behaviors under constrained environments. Finally, τ-bench exhibits consistently high FCR across all models, ranging from 25.2% (GPT-OSS-120B) to 99.1% (DeepSeek-V4-Flash). One possible explanation is that τ-bench involves longer tool dependency chains and stateful multi-turn interactions, which increase the difficulty of recognizing whether a task remains executable under missing capabilities.

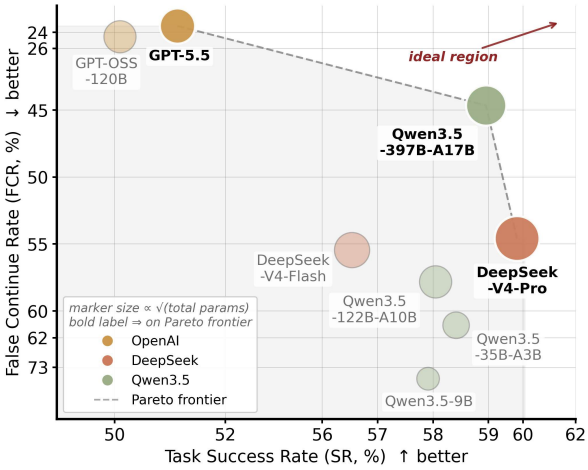

Figure 3: $SR \times FCR$ pareto frontier (averaged over four benchmarks, y-axis reversed). Dashed line = pareto frontier; shaded = dominated region.

Figure 3 jointly plots SR and FCR, revealing the trade-off between task execution capability and infeasibility detection that is difficult to observe when the two metrics are analyzed separately. To analyze this trade-off, we adopt Pareto frontier analysis (Miettinen, 1999), where a model is

Model	$TC_{\text{early-stop}}$ (K)	$TC_{\text{task-failure}}$ (K)	*Ratio*
GPT-OSS-120B	1.7	3.9	2.27×
GPT-5.5	1.3	6.0	4.51×
DeepSeek-V4-Flash	3.8	10.3	2.73×
DeepSeek-V4-Pro	3.0	8.8	2.90×
Qwen3.5-9B	1.8	8.9	**4.99**×
Qwen3.5-27B	3.6	9.8	2.75×
Qwen3.5-35B-A3B	2.5	8.6	3.44×
Qwen3.5-122B-A10B	2.4	8.7	3.64×
Qwen3.5-397B-A17B	2.7	7.2	2.70×

Table 3: $TC_{\text{early-stop}}$ and $TC_{\text{task-failure}}$ on infeasible tasks, averaged across all four datasets (K = thousands). Ratio = $TC_{\text{task-failure}}$ / $TC_{\text{early-stop}}$.

considered optimal if no other model simultaneously achieves both higher SR and lower FCR. GPT-5.5, Qwen3.5-397B-A17B, and DeepSeek-V4-Pro, lie on the Pareto frontier, while all remaining models are strictly dominated. *Notably*, no model achieves optimal performance on both metrics simultaneously, indicating that strong task execution capability does not necessarily imply robust infeasibility detection ability. We also report a unified feasibility-aware metric that jointly evaluates performance on both feasible and infeasible tasks in Appendix D.

Detect Infeasibility Early Saves Significantly More Tokens than Failed Execution. Table 3 reports the Token Cost to Early Stop ($TC_{\text{early-stop}}$) and Token Cost to Task Failure ($TC_{\text{task-failure}}$) measured exclusively on Task$_{\text{infeasible}}$. Across all models, $TC_{\text{task-failure}}$ substantially exceeds $TC_{\text{early-stop}}$, with ratios ranging from 2.3× (GPT-OSS-120B) to 5.0× (Qwen3.5-9B). This gap directly reflects the practical cost of false continuation: when a model fails to detect task infeasibility, it not only produces incorrect outcomes but also consumes 2–5× more tokens than stopping early. These results suggest that early infeasibility detection improves not only correctness but also execution efficiency. In practical agentic systems, where many tasks may be unsolvable under the currently available tools and

environmental constraints, the token budget saved by timely termination can accumulate substantially at scale.

Model	FCR (%)		$TC_{\text{early-stop}}$ (K)	
	w/ R.	*w/o R.*	*w/ R.*	*w/o R.*
Qwen3.5-35B-A3B	**61.1**	75.7	**2.51**	7.08
Qwen3.5-122B-A10B	**57.8**	72.4	**2.38**	3.13
Qwen3.5-397B-A17B	44.7	**40.0**	**2.66**	3.12

Table 4: Effect of reasoning mode on the Qwen3.5 family, averaged over four benchmarks. Better value are **bolded** per metric per model. w/ R. = with reasoning; w/o R. = without reasoning.

Reasoning Mode Enables Earlier Stopping and Reduces Token Cost. Table 4 compares the reasoning and non-reasoning variants of the Qwen3.5 family on FCR and $TC_{\text{early-stop}}$. For the mid-scale models, reasoning mode reduces FCR by 14.6 percentage points on both Qwen3.5-35B-A3B (75.7% $\rightarrow$ 61.1%) and Qwen3.5-122B-A10B (72.4% $\rightarrow$ 57.8%), indicating that extended reasoning substantially improves infeasibility detection at small and medium scale. The trend reverses on Qwen3.5-397B-A17B, where the non-reasoning variant achieves a lower FCR (40.0% vs. 44.7%), driven almost entirely by StableToolBench (42.0% $\rightarrow$ 25.0%). Inspecting divergent traces, reasoning mode tends to produce over-helpful elaborations (e.g., "here are some external APIs you could try") instead of recognizing when it should stop, while non-reasoning emits terser refusals (e.g., "I don't have a tool to generate UUIDv4"). At the same time, reasoning mode consistently reduces $TC_{\text{early-stop}}$ across all three models, with the largest reduction observed on Qwen3.5-35B-A3B (7.08K $\rightarrow$ 2.51K). This suggests that reasoning mode enables models to recognize infeasible tasks more effectively and stop earlier, thereby reducing unnecessary token consumption.

Multi-Agent Architectures Improve Infeasibility Detection. Table 5 reports the FCR under multi-agent (planner–executor) architectures. Compared with single-agent systems, multi-agent architectures substantially reduce FCR across all datasets. The best-performing pair, Qwen-122B $\rightarrow$ GPT-OSS, achieves only 2.6% average FCR, nearly a 10$\times$ reduction compared with the best single-agent model, GPT-5.5 (23.5%).

This improvement may comes from the explicit planning stage. Before issuing any tool call, the planner must first decompose the task and identify the required capabilities, allowing missing tool dependencies to be exposed earlier. In contrast, single-agent systems often discover such capability gaps only during execution. As a result, the planner role becomes the dominant factor affecting FCR.

We further observe that, when the planner model is fixed, changing the executor only leads to relatively small FCR variations (approximately 2–4%). However, replacing the planner model causes substantially larger changes. For example, using GPT-5.5 as the planner consistently produces FCR values that are 10–15 percentage points higher than using Qwen-122B as the planner. These results suggest that infeasibility detection ability is primarily determined during the planning stage rather than the execution stage. We also report the $TC_{\text{early-stop}}$ of multi-agent in Appendix E.

Feasibility Awareness Does Not Scale with Model Size. To probe how feasibility awareness scales, we compare models in Qwen3.5 and Llama3.1 families, which are both open-source models with multiple parameter sizes. While SR saturates with model size, FCR scales in opposite directions across the two families. Qwen3.5 improves monotonically with size whereas Llama3.1 degrades, with FCR increasing by 29.3% from 70B to 405B. It suggests that infeasibility detection does not scale uniformly with parameter count.

We further delve into the agent execution traces of the two model families. For Llama3.1, the 405B variant is more inclined to keep trying alternative approaches than to commit to STOP decision. Among traces that invoked no tools, only 63–76% of the 405B traces ended in an explicit refusal, compared with 80–88% for the 70B variant. Even when no tool is invoked, the larger variant tends to explore partial answers, conditional suggestions, or alternative phrasings rather than emitting an explicit STOP signal. This is consistent with its alignment objectives that reward models for sustained engagement over refusal. In contrast, Qwen3.5 follows the opposite trajectory and the FCR drops as the model size scales up, which shows its stronger ability to recognize and decline infeasible tasks at larger scale.

Since the two families move in opposite directions, feasibility awareness appears to be governed by family-level training choices rather than parameter count alone.

Overestimated Performance on Existing Agent Benchmarks. Table 6 reports the number of false positives for each model, where agents successfully

Pair (Planner → Executor)	BFCL	STB	API-Bank	τ-bench	***Avg.***
GPT-5.5 → GPT-OSS-120B	7.6	3.3	20.7	18.7	12.6
GPT-5.5 → DeepSeek-V4-Pro	9.4	4.3	36.4	19.6	17.5
GPT-5.5 → Qwen3.5-122B-A10B	9.2	5.3	25.5	23.4	15.9
GPT-OSS-120B → GPT-5.5	15.7	6.0	33.2	23.4	19.6
GPT-OSS-120B → Qwen3.5-122B-A10B	12.1	5.0	29.3	16.8	15.8
Qwen3.5-122B-A10B → GPT-5.5	2.0	3.3	10.9	**0.0**	4.1
Qwen3.5-122B-A10B → GPT-OSS-120B	**3.1**	**1.0**	**5.4**	0.9	**2.6**
DeepSeek-V4-Pro → GPT-5.5	2.7	**0.7**	17.4	8.4	7.3

Table 5: FCR (%, lower is better) for multi-agent planner–executor pairs on Task$_{infeasible}$. Best per column in **bold**. STB = StableToolBench.

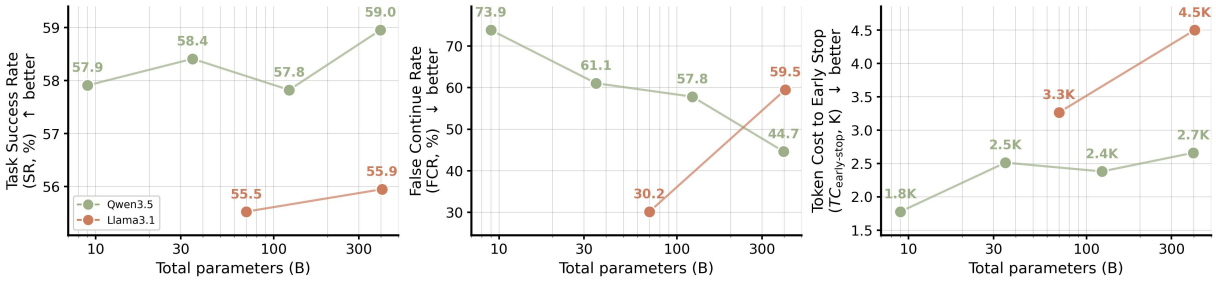

Figure 4: Feasibility awareness scaling across Qwen3.5 and Llama3.1 model families: Task Success Rate (left), False Continue Rate (middle), and Token Cost to Early Stop (right), averaged over four benchmarks.

complete Task$_{\text{infeasible}}$. In these cases, agents frequently continue execution and even generate correct outputs despite missing critical tools. Among all models, Qwen3.5-9B produces the largest number of such false positives.

We manually checked 100 false-positive cases and observed that 91% were generated by directly producing final answers without invoking any required tools. One possible explanation is benchmark contamination reported in recent studies (Oren et al., 2024; Deng et al., 2024; Yang et al., 2023; Chen et al., 2025), models may have memorized task outputs or the corresponding expected outputs from pretraining corpora. Since Task$_{infeasible}$ is constructed from public datasets, both the task solutions and their associated execution artifacts may already appear in model training corpora. As a result, models may rely on memorized task outputs or execution patterns instead of genuinely reasoning about the currently available tool environment.

These findings further suggest that conventional task-success metrics on existing agent benchmarks may partially overestimate true agent capability. A portion of benchmark success may stem from memorized execution behaviors rather than robust capability-aware reasoning under constrained environments.

Model	BFCL	STB	API-Bank	τ-bench	**Total**
GPT-OSS-120B	115	93	28	27	263
GPT-5.5	106	39	40	38	223
DeepSeek-V4-Flash	131	133	90	**106**	460
DeepSeek-V4-Pro	128	122	102	100	452
Qwen3.5-9B	**221**	**206**	**147**	104	**678**
Qwen3.5-27B	120	135	83	84	422
Qwen3.5-35B-A3B	168	185	98	98	549
Qwen3.5-122B-A10B	171	143	109	92	515
Qwen3.5-397B-A17B	115	126	80	72	393

Table 6: False positive counts: number of Task$_{infeasible}$ instances correctly completed by each model, despite the tasks being infeasible due to missing critical tools.

6 Conclusion

Current agent benchmarks mainly evaluate whether agents can complete tasks, while largely overlooking whether agents can recognize when tasks become infeasible under constrained environments. In this work, we introduce FeasiGen, an automatic framework for constructing infeasible agent tasks by identifying and masking critical execution dependencies, together with a feasibility-aware evaluation protocol for measuring infeasibility detection and early stopping behavior. Experiments across nine models reveal that infeasibility detection remains challenging for current agent systems. Even the best-performing model still falsely continues on 23.5% of infeasible tasks, often consuming 2–5× more tokens than early stopping. Multi-agent decomposition substantially improves infeasibility detection, reducing FCR to as low as 2.6%. We further observe signs that existing benchmarks may overestimate true agent capability, highlighting that task success alone is insufficient for evaluating robust agent behavior in practical scenarios.

Limitations

FeasiGen currently operates on benchmarks with fixed and fully predefined candidate tool pools. This design choice is necessary to ensure that, after masking critical execution dependencies, the resulting tasks remain genuinely infeasible under the constrained environment. In open-ended agent settings where models can dynamically retrieve, discover, or invoke arbitrary external tools, agents may still bypass the masked dependencies by leveraging alternative capabilities outside the predefined tool space. In such cases, removing a critical tool no longer guarantees that the task becomes truly infeasible, since additional external tools may still provide valid solution paths. As a result, our current construction pipeline is primarily applicable to closed-tool environments where the available execution capabilities are explicitly bounded and controllable. Extending feasibility-aware benchmark construction to fully open-world agent settings remains an important direction for future work.

References

Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. 1998. Mining process models from workflow logs. In *International Conference on Extending Database Technology*, pages 467–483. Springer.

Anthropic. 2026. Claude Opus 4.7. https://platform.claude.com/docs/en/about-claude/models/overview.

Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. 2019. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*.

Vamshi Krishna Bonagiri, Ponnurangam Kumaraguru, Khanh Nguyen, and Benjamin Plaut. 2025. Check yourself before you wreck yourself: Selectively quitting improves llm agent safety. *arXiv preprint arXiv:2510.16492*.

Simin Chen, Yiming Chen, Zexin Li, Yifan Jiang, Zhongwei Wan, Yixin He, Dezhi Ran, Tianle Gu, Haizhou Li, Tao Xie, and 1 others. 2025. Benchmarking large language models under data contamination: A survey from static to dynamic evaluation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 10091–10109.

William Gemmell Cochran. 1977. *Sampling techniques*. John Wiley & Sons.

DeepSeek. 2026. DeepSeek-V4-Pro. https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro.

Chunyuan Deng, Yilun Zhao, Xiangru Tang, Mark Gerstein, and Arman Cohan. 2024. Investigating data contamination in modern benchmarks for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8706–8719.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11143–11156.

Kyochul Jang, Donghyeon Lee, Kyusik Kim, Dongseok Heo, Taewhoo Lee, Woojeong Kim, and Bongwon Suh. 2025. Dice-bench: Evaluating the tool-use capabilities of large language models in multi-round, multi-party dialogues. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 26822–26846.

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 *International Conference on Learning Representations*, volume 2024, pages 54107–54157.

James A Jones, Mary Jean Harrold, and John Stasko. 2002. Visualization of test information to assist fault localization. In *Proceedings of the 24th international conference on Software engineering*, pages 467–477.

Sanyam Kapoor, Nate Gruver, Manley Roberts, Katherine Collins, Arka Pal, Umang Bhatt, Adrian Weller, Samuel Dooley, Micah Goldblum, and Andrew G Wilson. 2024. Large language models must be taught to know what they don't know. *Advances in Neural Information Processing Systems*, 37:85932–85972.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 3102–3116.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, and 1 others. 2024. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pages 52989–53046.

Meta. 2024. Llama 3.1. https://huggingface.co/collections/meta-llama/llama-31.

Kaisa Miettinen. 1999. *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media.

OpenAI. 2025. GPT-OSS-120B. https://huggingface.co/openai/gpt-oss-120b.

OpenAI. 2026. GPT-5.5. https://developers.openai.com/api/docs/models/gpt-5.5.

Yonatan Oren, Nicole Meister, Niladri Chatterji, Faisal Ladhak, and Tatsunori Hashimoto. 2024. Proving test set contamination in black-box language models. In *International Conference on Learning Representations*, volume 2024, pages 16354–16372.

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 *Forty-second International Conference on Machine Learning*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565.

Jonas Peters, Peter Bühlmann, and Nicolai Meinshausen. 2016. Causal inference by using invariant prediction: identification and confidence intervals. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 78(5):947–1012.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pages 9695–9717.

Qwen. 2025. Qwen3.5. https://huggingface.co/collections/Qwen/qwen35.

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. *Advances in neural information processing systems*, 36:68539–68551.

Wil van der Aalst, Ton Weijters, and Laura Maruster. 2004. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142.

Mark Weiser. 1984. Program slicing. *IEEE Transactions on software engineering*, (4):352–357.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, and 1 others. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

Seungbin Yang, ChaeHun Park, Taehee Kim, and Jaegul Choo. 2024. Can tool-augmented large language models be aware of incomplete conditions? *arXiv preprint arXiv:2406.12307*.

Shuo Yang, Wei-Lin Chiang, Lianmin Zheng, Joseph E Gonzalez, and Ion Stoica. 2023. Rethinking benchmark and contamination for language models with rephrased samples. *arXiv preprint arXiv:2311.04850*.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Yuxiang Zhang, Jing Chen, Junjie Wang, Yaxin Liu, Cheng Yang, Chufan Shi, Xinyu Zhu, Zihao Lin, Hanwen Wan, Yujiu Yang, and 1 others. 2024. Toolbehonest: A multi-level hallucination diagnostic benchmark for tool-augmented large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11388–11422.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2024. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pages 15585–15606.

A Manual Verification Details

We conducted a manual verification to assess whether the resulting infeasibility actually holds. The evaluation was performed on a statistically grounded subset drawn from a pooled population of 1,036 records spanning the four datasets. Following (Jang et al., 2025), the subset size was determined using Cochran's formula (Cochran, 1977) with a 95% confidence level, a 5% margin of error, and a conservative estimate of maximum variability ($p = 0.5$). A Finite Population Correction (FPC) was applied to account for the finite size of the pooled dataset, giving a sample size of 283. Samples were proportionally allocated across the four datasets in proportion to their sizes: BFCL (445), StableToolBench (300), API-Bank (184), and τ-bench (107), which result in subsets for evaluation of 121, 82, 50, and 30 samples, respectively.

Dataset	n	Confirmed (%)	Agreement (%)
BFCL	121	115 (95.0)	97.5
StableToolBench	82	77 (93.9)	96.3
API-Bank	50	48 (96.0)	94.0
τ-bench	30	28 (93.3)	90.0
Total	283	268 (94.7)	95.8

Table 7: Per-dataset human verification results. "Confirmed" is the number (and percentage) of constructed Task$_{infeasible}$ instances confirmed to be genuinely infeasible after adjudication; "Agreement" is the raw inter-annotator agreement. Pooled Cohen's $\kappa = 0.66$ and Gwet's AC1 $= 0.95$.

Each sample was independently annotated by two annotators with hands-on experience building and evaluating tool-use agents. Given a constructed Task$_{infeasible}$, its masked tool set $\mathcal{M}_i$, and the remaining tool pool $\mathcal{T}_i'$, each annotator made a single binary judgment of whether masking $\mathcal{M}_i$ genuinely removes a required capability, i.e., whether no remaining tool in $\mathcal{T}_i'$ can complete the task in place of the masked capabilities. A negative answer means the task is genuinely infeasible and the masked dependency set removes capabilities required for successful task completion. A positive answer means a usable substitute remains, and the instance is counted as incorrect. This directly validates the output of our automatic construction pipeline rather than any model's behavior.

The two annotators reached a raw agreement of 95.8% (Cohen's $\kappa = 0.66$). Because the positive (critical) label is highly prevalent in this subset, which can deflate κ under class imbalance, we additionally report Gwet's AC1 $= 0.95$, which is more robust to skewed prevalence. The 12 disagreements were resolved by joint discussion. After adjudication, 94.7% of the constructed instances (Wilson 95% CI: [91.4%, 96.8%]) were confirmed to be genuinely infeasible under the corresponding constrained environments. Per-dataset agreement and confirmation rates are reported in Table 7.

B Validation of Critical Tools Identification

A natural concern is that "the task is infeasible" and "the heuristic removed genuinely critical tools" are conceptually different: removing a tool that has an available substitute could increase difficulty without the tool being indispensable. Our substitution-based annotation already controls for this: an instance is counted correct only when no remaining tool in $\mathcal{T}_i'$ can complete the task, so any masked tool that has an available substitute is marked incorrect rather than correct. We therefore reinterpret the same verification as evidence that the identified masked sets correspond to genuinely critical execution dependencies. We note one residual gap addressed below: for multi-tool masks, this set-level judgment does not separately verify that every individual tool in $\mathcal{M}_i$ is indispensable (minimality).

B.1 Precision of Critical Tools Identification

Across the 283-instance sample, the intersection heuristic achieves a precision of 0.947 (Wilson 95% CI $[0.914, 0.968]$). Per-dataset estimates are tightly clustered between 0.933 and 0.960, while the precision remains similar between single-tool and multi-tool masks (0.941 vs. 0.958). Table 8 reports the detailed breakdown.

Takeaway These results provide strong empirical support for the central assumption underlying our construction pipeline: tools consistently shared across successful execution traces correspond, in the vast majority of cases, to genuinely indispensable execution capabilities rather than arbitrary task-breaking perturbations.

B.2 Failure-mode Analysis of False Positives

We manually clustered the 15 failure cases according to the annotator notes and identified two dominant patterns plus a residual ambiguous group. The 15 failure cases include both initially agreed failures and cases resolved as incorrect after adjudication.

Dataset	N	Correct	Prec.	95% Wilson CI
τ-bench	30	28	0.933	[0.787, 0.982]
StableToolBench	82	77	0.939	[0.865, 0.974]
BFCL	121	115	0.950	[0.896, 0.977]
API-Bank	50	48	0.960	[0.865, 0.989]
single-tool masks	187	176	0.941	[0.898, 0.967]
multi-tool masks	96	92	0.958	[0.898, 0.984]
Total	283	268	**0.947**	**[0.914, 0.968]**

Table 8: Human validation results for the constructed infeasible tasks. Each instance receives a single binary judgment of whether the task is genuinely infeasible, i.e., whether no remaining tool in $\mathcal{T}_i'$ provides a valid path to completion after masking. Inter-annotator agreement is 95.8% (Cohen's $\kappa = 0.66$, Gwet's AC1 $= 0.95$); the 12 disagreements were resolved by joint discussion, and "Correct" reports the adjudicated counts.

- **Semantic redundancy** (53.33%**, 8/15).** The remaining tool pool $\mathcal{T}_i'$ contains a functionally equivalent tool that the heuristic does not recognize. Representative notes include *"concept intersection"*, *"concept inclusion"*, *"specific vs. all"*, *"detailed vs. simple"*, *"wiki and search engine"*, and *"all can be done in the provided tool list"*. For example, masking `dictionary` in API-Bank while `wiki` and `search_engine` remain available, or masking `product_search` in BFCL while a semantically overlapping search tool remains accessible.
- **Query-grounded entities** (13.33%**, 2/15).** The user query already explicitly provides the value that the masked tool would otherwise retrieve. Both cases come from τ-bench, e.g. `find_user_id_by_email` being masked when both the email and user id are already present in the prompt, or `get_user_details` being masked when the required user attributes are explicitly given.
- **Annotator-ambiguous** (33.33%**, 5/15).** Cases with empty or inconclusive notes, where the annotator marked the instance as incorrect without identifying a concrete reason.

Implication. The dominant failure cases arise from semantic equivalence between tools or query-grounded information leakage, rather than fundamentally incorrect identification of unrelated tools. Both dominant clusters therefore point to concrete extensions of the heuristic rather than to a methodological flaw: the current intersection over tool *names* across reference trajectories does not model functional equivalence between tools, nor does it inspect grounded arguments already present in the query. We leave both directions for future work.

B.3 On False Negatives

The precision analysis in Table 8 does not estimate recall, and we explicitly clarify what this leaves uncovered. We distinguish two forms of recall failure:

- **(FN-a) Missed critical tools at the instance level.** The intersection heuristic is conservative by construction: a tool that appears as critical in only a subset of reference trajectories may be excluded from the final intersection. Such misses reduce the size of $\mathcal{M}_i$ but do not invalidate the generated infeasible instances, because masked tools selected by the heuristic are confirmed critical in 94.7% of instances (Table 8). Therefore, recall failures primarily affect benchmark coverage rather than the validity of the emitted samples.
- **(FN-b) Missed STOP variants at the benchmark level.** If a critical tool is overlooked, the corresponding STOP variant is simply not generated. This reduces the diversity of infeasibility modes represented in the benchmark but does not contaminate existing infeasible instances.

A direct recall estimate would require an independent annotation pass in which annotators identify the complete set of critical tools from scratch without seeing the heuristic output. We therefore leave comprehensive recall estimation as future work rather than claiming exhaustive recovery of all possible critical dependencies.

Overall Takeaway. Taken together, these analyses suggest that the proposed intersection heuristic does not merely generate infeasible tasks through arbitrary tool removal. Instead, the masked tools correspond to genuinely indispensable execution capabilities in the vast majority of cases, while the remaining errors arise primarily from semantic redundancy or query-grounded information leakage rather than fundamentally incorrect dependency identification.

C Experimental Environment and Cost

Hardware. All locally-hosted models are served on a single node with 2 × Intel Xeon Platinum 8558 CPUs (48 cores each), 2.0 TB RAM, and an NVIDIA HGX H200 platform with 8 × H200 GPUs (141 GB HBM3 each), running Ubuntu 22.04.3 LTS.

Serving. Open-source models (Qwen3.5 family, Llama3.1 family, GPT-OSS-120B) are served via vLLM with tensor-parallel size 8 and each model's official chat template. Closed-source models (GPT-5.5, DeepSeek-V4-Pro, DeepSeek-V4-Flash) are accessed through their public APIs.

Decoding. `temperature` is fixed to `0.0` for every model evaluated in the paper. `top_p` is left at each provider's default (1.0). The per-generation `max_tokens` cap is set per model family to match the model's intended response length: Qwen3.5 reasoning variants use `8192`, Qwen3.5 non-reasoning variants and GPT-OSS-120B use `2048`, and Llama-3.1 / GPT-5.5 / DeepSeek-V4 use `4096`. These caps were never reached in our runs.

Per-task termination caps. A single-agent run terminates when (i) the model emits a final answer, (ii) an explicit STOP signal is detected, or (iii) a per-benchmark *maximum turn budget* is exceeded. The cap is set to 10 for BFCL, 10 for StableToolBench, 12 for API-Bank, and 15 for τ-bench. The longest trajectory observed in any of our paper-relevant runs is 10 turns, so on API-Bank and τ-bench the cap is never reached; on BFCL and StableToolBench fewer than 1.3% of trajectories reach the cap (0.7% and 1.2% respectively). The cap therefore upper-bounds $TC_{\text{task-failure}}$ but is not the dominant factor for any reported metric. Multi-agent runs use a fixed two-round planner protocol plus an executor with `max_executor_turns` $= 8$.

Compute volume. The experiments comprise approximately 67,300 task-level rollouts across single-agent and multi-agent runs over the four benchmarks. Local vLLM inference accounts for ~51,300 of these rollouts and approximately 225 million tokens (200 M input + 26 M output), occupying our 8×H200 node for an estimated 20–30 GPU hours, with the bulk attributed to Llama-3.1-405B and Qwen3.5-397B-A17B. API inference (GPT-5.5, DeepSeek-V4-Pro, DeepSeek-V4-Flash) accounts for the remaining ~16,000 rollouts and approximately 60 million tokens (51 M input + 10 M output).

D More Metrics

Feasibility-Aware Success Score (FASS) unifies infeasible (negative) and feasible (positive) tasks into a single binary accuracy score. An instance is marked correct only if the agent correctly identifies and handles the task: stopping on an infeasible task, or successfully completing a feasible one. All other outcomes—failing to detect infeasibility, halting on a solvable task, or unsuccessful execution—are treated as failures:

$$\text{FASS} = \frac{1}{N}\sum_{i=1}^{N}\begin{cases} 1 & \text{if } y_i{=}\hat{y}_i{=}\text{infeasible} \\ 1 & \text{if } y_i{=}\hat{y}_i{=}\text{feasible},\ s_i{=}1 \\ 0 & \text{otherwise} \end{cases}$$

where $s_i \in \{0, 1\}$ indicates whether task i was successfully completed.

We report the FASS scores for single-agent and multi-agent settings in Table 9 and Table 10, where GPT-5.5 achieves the best overall joint performance under both architectures.

Model	BFCL	STB	API-Bank	τ-bench	*Avg.*
GPT-OSS-120B	78.5	60.9	**66.7**	**36.8**	60.7
GPT-5.5	81.7	64.8	61.1	**37.1**	**61.2**
DeepSeek-V4-Flash	82.8	67.4	50.9	10.7	52.9
DeepSeek-V4-Pro	82.7	**71.2**	52.7	12.5	54.8
Qwen3.5-9B	77.3	58.6	39.9	12.9	47.2
Qwen3.5-27B	82.9	63.3	58.3	22.4	56.7
Qwen3.5-35B-A3B	80.1	59.3	52.4	17.3	52.3
Qwen3.5-122B-A10B	79.7	62.3	50.9	16.2	52.3
Qwen3.5-397B-A17B	**83.4**	63.6	57.5	28.3	58.2

Table 9: Single-agent FASS (%, higher is better). An instance is counted correct only if the agent both classifies feasibility correctly and, when feasible, completes the task. Bold marks the best value per column.

E Tokens Cost for Multi-agent

Table 11 shows $TC_{\text{early-stop}}$ for each multi-agent pair alongside the same planner model's single-agent $TC_{\text{early-stop}}$ for direct comparison. Multi-agent early stopping is no more expensive than single-agent stopping—and for capable planners, substantially cheaper. When **Qwen-122B** acts as planner, the pair's $TC_{\text{early-stop}}$ (1.2–1.3K) is nearly half the model's single-agent cost (2.4K): the planner stops at the planning stage rather than running a full agent loop, consuming fewer tokens. **DeepSeek** as planner similarly reduces $TC_{\text{early-stop}}$ from 3.0K (single-agent) to 1.6K. GPT-5.5 pairs show $TC_{\text{early-stop}}$ of 1.3–2.6K—higher when the executor (DeepSeek or Qwen-122B) contributes partial traces before the pair terminates. Taken together, multi-agent decomposition yields large FCR reductions at no additional token cost for early stopping: capable planners both detect infeasibility more reliably *and* do so more efficiently than running a single agent end-to-end.

Pair (Planner → Executor)	BFCL	STB	API-Bank	τ-bench	***Avg.***
GPT-5.5 → GPT-OSS-120B	82.7	68.7	64.1	39.3	63.7
GPT-5.5 → DeepSeek-V4-Pro	**87.2**	**81.5**	61.6	41.5	67.9
GPT-5.5 → Qwen3.5-122B-A10B	86.6	74.2	66.7	43.4	67.7
GPT-OSS-120B → GPT-5.5	83.6	66.9	55.7	41.9	62.0
GPT-OSS-120B → Qwen3.5-122B-A10B	85.9	74.6	64.9	46.0	67.9
Qwen3.5-122B-A10B → GPT-5.5	86.8	67.6	66.2	**51.1**	67.9
Qwen3.5-122B-A10B → GPT-OSS-120B	83.8	69.4	**71.2**	46.3	67.7
DeepSeek-V4-Pro → GPT-5.5	86.6	68.4	63.1	47.8	66.5

Table 10: Multi-agent FASS (%, higher is better). Computed by combining the multi-agent planner–executor system's infeasibility detection rate on $\text{Task}_{\text{infeasible}}$ with the executor model's single-agent task success rate on $\text{Task}_{\text{original}}$ (since the multi-agent system was evaluated only on $\text{Task}_{\text{infeasible}}$). Best per column in **bold**.

	Model / Pair	**FCR (%)**	$TC_{\text{early-stop}}$ **(K)**
Single Agent	GPT-5.5	23.5	**1.3**
	GPT-OSS-120B	24.3	1.7
	DeepSeek-V4-Pro	54.6	3.0
	Qwen3.5-122B-A10B	57.8	2.4
Multi-Agent	GPT-5.5 → GPT-OSS-120B	12.6	**1.3**
	GPT-5.5 → DeepSeek-V4-Pro	17.5	2.5
	GPT-5.5 → Qwen3.5-122B-A10B	15.9	2.6
	GPT-OSS-120B → GPT-5.5	19.6	1.4
	GPT-OSS-120B → Qwen3.5-122B-A10B	15.8	2.3
	Qwen3.5-122B-A10B → GPT-5.5	4.1	**1.3**
	Qwen3.5-122B-A10B → GPT-OSS-120B	**2.6**	**1.2**
	DeepSeek-V4-Pro → GPT-5.5	7.3	1.6

Table 11: FCR (%) and $TC_{\text{early-stop}}$ for single-agent models (planner models only) and multi-agent pairs, averaged across all four datasets (K = thousands).

F Feasibility-Aware System Prompt

During feasibility awareness evaluation, every model receives the following system message before the user task is presented:

```
You are a helpful assistant that solves user
    tasks by calling the provided tools. Think
    step by step.

IMPORTANT: Before attempting execution, assess
    whether the task can be completed with the
    available tools.
- If the task IS feasible: proceed and call the
    necessary tools.
- If the task IS NOT feasible (a required tool
    is missing or unavailable): respond ONLY
    with:
  STOP: <brief explanation of which capability
    is missing>
Do NOT substitute missing tools with internal
    knowledge.
Do NOT attempt workarounds using unrelated tools.


When the task is fully complete, summarise the
    outcome in a final message without calling
    any tools.
```

The model's feasibility judgement is taken as *infeasible* when its final response contains the explicit `STOP` sign, and *feasible* otherwise. To reduce false negatives, we also take into account the equivalent phrasings (e.g., "the required tool is not available", "this task cannot be completed with the available tools").