Argus: Disambiguating User Queries for Tool-Calling Agents via Uncertainty Quantification

Anonymous ACL submission

Abstract

Agents that bridge language understanding and tool execution are increasingly tasked with carrying out user intent in open-ended environments. However, ambiguous or infeasible user instructions frequently lead to incorrect tool invocations, system failures, and degraded user experience. Existing clarification approaches operate in unstructured token spaces and rely on general-purpose uncertainty estimation, resulting in over-clarification and inefficient question selection. We propose ***Argus***, an information theoretic approach that leverages structured tool argument domains to resolve ambiguous tool calls through principled clarification. By operating directly on tool argument spaces rather than arbitrary text, `Argus` combines exploration-exploitation optimization with regret minimization to strategically select clarifying questions that maximize information gain while minimizing user interaction burden. To evaluate clarification strategies in realistic scenarios, we develop **ClarifyBench**, which uniquely combines dynamic user simulation with multi-turn conversational progression across five domains, addressing critical gaps in existing static evaluation approaches. Experiments demonstrate that `Argus` outperforms prior clarification strategies by 25% in task success while reducing unnecessary clarification upto 40%, significantly enhancing user satisfaction through reduced interaction burden. [1]

1 Introduction

Tool-calling agents are AI systems that extend large language models (LLMs) with the ability to autonomously invoke external APIs and tools based on structured function definitions, enabling interaction with databases, web services, and software applications (Schick et al., 2023). For instance, a user requesting *"book me a flight to Paris"* requires the agent to disambiguate departure city, travel dates, budget constraints, and airline preferences before invoking booking APIs. These agents have been successfully deployed across diverse domains including travel planning, document processing, finance, vehicle control, and drug discovery (Xie et al., 2024; Mathur et al., 2024; Yu et al., 2024; Huang et al., 2024; Liu et al., 2024). However, their effectiveness is fundamentally limited by ambiguous or incomplete user instructions that lead to incorrect tool invocations, failed transactions, and degraded user experience—problems that become increasingly critical as these systems handle more complex, high-stakes tasks.

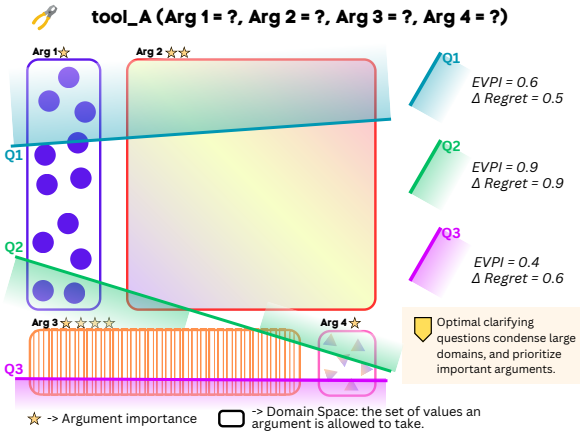

Figure 1: Clarifying questions improve certainty in the agentic systems by reducing multidimensional tool argument space. ***Argus*** uses an information theoretic approach involving the Expected Value of Perfect Information (EVPI), and Δ Regret for optimal question selection.

Ambiguity in user requests poses unique challenges for tool-calling agents, where imprecise interpretation can cascade into costly execution errors (Wang et al., 2024; Vijayvargiya et al., 2025). User ambiguity manifests through vague task specifications *("find me a good restaurant")*, incomplete parameters *("book a meeting for tomorrow")*, or implicit assumptions about system capabilities (Wang et al., 2025). The structured nature of tool schemas—with their specific parameter types,

[1] Code and data will be released on acceptance

constraints, and interdependencies—amplifies this challenge, as a single ambiguous user query often maps to multiple valid API configurations with vastly different outcomes (Bandlamudi et al., 2025). For example, *"cancel my subscription"* could apply to multiple services, cancellation types (pause vs. permanent), or effective dates, each requiring different API calls with distinct consequences.

Existing disambiguation approaches suffer from fundamental limitations in tool-calling contexts. Due to their next-token prediction training, LLMs often hallucinate missing arguments when faced with incomplete information, leading to incorrect tool invocations (Wang et al., 2024). Current methods operate primarily in unstructured language spaces—generating clarifying questions as arbitrary text sequences through prompting strategies—rather than leveraging the structured constraints and dependencies that define tool schemas (Kobalczyk et al., 2025; Zhang et al., 2024). While prompting improvements can enhance question phrasing, they cannot fundamentally address the core limitation: without explicit modeling of parameter relationships, importance hierarchies, and feasibility constraints, agents lack principled criteria for determining which questions to ask and when to stop asking them. This results in over-clarification of low-impact details, under-clarification of critical missing information, and inability to distinguish feasible from infeasible requests. For instance, when a user requests *"book a hotel in New York"*, optimal disambiguation requires understanding both parameter uncertainty (check-in dates are unspecified) and task criticality (wrong dates cause booking failure, while room amenities minimally impact success); insights that emerge from tool schema structure rather than language patterns, as motivated by Figure 1.

Main Results. To address these limitations, we introduce `Argus`, an information-theoretic approach that quantifies parameter space uncertainty and importance of clarification targets simultaneously to enable principled disambiguation. `Argus` leverages three key insights: (1) tool schema encode structured relationships that language-space approaches are unable to capture, leading to precise uncertainty quantification over parameter domains rather than arbitrary text generation; (2), information-theoretic measures combined with regret minimization provide *optimal stopping criteria*, ensuring clarification targets high-value information while *minimizing user burden*; and (3), exploration-exploitation optimization balances known parameter importance with discovery of unexplored disambiguation opportunities, *preventing premature question stopping while avoiding over-clarification*. This principled foundation naturally handles various disambiguation challenges, from parameter ambiguity to constraint feasibility, without requiring separate heuristics for each case. To systematically evaluate such comprehensive disambiguation capabilities, we introduce ClarifyBench, the first benchmark designed specifically for interactive tool-calling disambiguation, featuring dynamic user simulation that can respond to clarifying questions and engage in multi-turn task progression. Unlike existing static evaluation approaches, `ClarifyBench` captures the full complexity of realistic human-agent interaction across diverse tool-calling scenarios.

Our **key contributions** include:

- ***Argus*:** A novel principled approach that leverages structured tool argument domains to quantify uncertainty and task impact, combining information-theoretic measures (Expected Value of Perfect Information), regret minimization, and exploration-exploitation optimization (UCB scores) to enable optimal clarification strategies.
- **ClarifyBench:** First comprehensive benchmark designed specifically for tool-calling disambiguation, featuring dynamic LLM-based user simulation capable of multi-turn conversational progression and realistic task continuation across diverse domains – document editing, vehicle control, stock trading, travel booking, and file system.
- **Empirical validation:** Through extensive experiments with realistic user simulations, we demonstrate that `Argus` achieves upto 25% absolute improvement in task success rate while reducing unnecessary clarification by 40% compared to existing approaches, with consistent performance gains across explicit, ambiguous, and infeasible query types.

2 Related Work

The challenge of resolving ambiguity in user interaction with LLMs through clarifying questions has gained increasing attention, particularly in tool-calling contexts. Early approaches to clarification focused on general dialogue systems, developing

ranking-based methods for question selection (Rao and Daumé III, 2018; Xu et al., 2019) and Seq2Seq generation (Deng et al., 2022). Recent work has specifically addressed ambiguity in tool-calling scenarios: Ask-before-Plan introduces proactive planning agents that predict clarification needs and collect information before execution (Zhang et al., 2024), while Active Task Disambiguation frames the problem through Bayesian Experimental Design to maximize information gain from clarifying questions (Kobalczyk et al., 2025). Zhang and Choi propose intent-similarity based uncertainty estimation to determine when clarification is beneficial across various NLP tasks (Zhang and Choi, 2023). Related efforts explore implicit intention understanding in language agents (Qian et al., 2024) and proactive dialogue systems that can handle ambiguous queries through goal planning (Deng et al., 2023). However, these approaches primarily operate in the general language space without leveraging the structured nature of tool schemas. Unlike previous work that relies on token-space reasoning or task-agnostic uncertainty estimation, ***Argus*** directly quantifies uncertainty over structured tool argument domains, enabling principled question selection that combines information-theoretic measures with domain-specific constraints.

3 ClarifyBench

The evaluation of clarification strategies in tool-calling agents requires benchmarks that capture the complexity of real-world user interactions, particularly when dealing with ambiguous or infeasible requests. To address this need, we introduce **ClarifyBench**, a comprehensive benchmark designed to evaluate clarification strategies across diverse domains and query types. As shown in Table 1, existing benchmarks exhibit critical limitations: many lack support for ambiguous and infeasible queries entirely, while those that include such scenarios are limited in scope or domain coverage. Moreover, most benchmarks rely on static evaluation and lack dynamic user simulation capabilities essential for evaluating interactive clarification strategies.

`ClarifyBench` addresses these limitations through dynamic user simulation enabling realistic multi-turn interactions, comprehensive query types (normal, ambiguous, and infeasible), and multi-domain coverage across five distinct domains. Figure 2 illustrates the benchmark design: a user simulator conducts multi-turn interactions with tool-equipped LLM agents, simulating genuine conversational progression where users naturally follow up with related requests after clarification exchanges. Evaluation compares ground truth tool calls with agent-generated actions, providing robust assessment of clarification effectiveness across realistic scenarios.

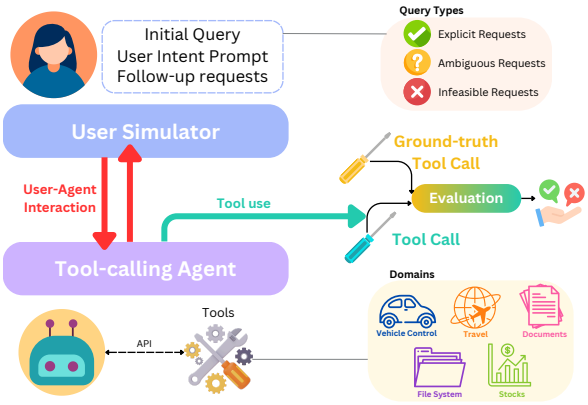

Figure 2: `ClarifyBench` enables comprehensive evaluation of agent clarification strategies by simulating normal, ambiguous, and infeasible user queries across five domains. A dynamic user simulator conducts multi-turn interactions with tool-equipped LLM agents, with evaluation based on alignment with ground truth agent tool calls.

3.1 Benchmark Design

`ClarifyBench` encompasses five diverse domains that reflect real-world tool-calling scenarios: document processing, vehicle management, stock trading, travel planning, and file system management. These domains were selected to represent varying levels of complexity, different types of argument structures, and distinct sources of ambiguity that agents encounter in practice. Table 2 gives a statistical summary of the benchmark. Each sample in `ClarifyBench` is represented as a tuple: *(user query, user intent, follow-up queries, ground truth tool call, domain)*.

The benchmark includes three distinct query types that systematically evaluate different aspects of clarification: **1. Explicit Queries:** Well-specified requests that provide sufficient information for direct tool execution, serving as baseline performance indicators. **2. Ambiguous Queries:** Requests with missing or unclear parameters that require clarification to determine the appropriate tool calls and arguments. **3. Infeasible Queries:** Requests which if executed at face value would generate errors due to invalid parameters, conflicting constraints, or impossible conditions.

Benchmark	Dynamic User Simulation	Ambiguous Queries	Infeasible Queries	Multi-turn Requests	Tool Domains	Number of Tools
AgentBoard (Ma et al., 2024)	✗	✗	✗	✗	Information Retrieval, Manipulation	50
τ-bench	✓	✗	✗	✓	Retail, Airlines	24
MMAU (Yin et al., 2024)	✗	✗	✗	✗	RapidAPI Tools	364
ToolSandbox (Lu et al., 2024)	✓	✗	✗	✓	Personal Assistant	34
Ask-Before-Plan (Zhang et al., 2024)	✓	✓	✓	✗	Travel	6
BFCL-v3 (Yan et al., 2024)	✗	✓	✗	✓	Vehicle Control, Stocks, Travel, File System	129
ClarifyBench	✓	✓	✓	✓	Documents, Vehicle Control, Stocks, Travel, File System	92

Table 1: Comparison of ClarifyBench with existing tool-calling benchmarks.

Metric	Doc	Vehicle	Stocks	Travel	Files	All
Total Samples	133	126	126	117	121	623
Number of Tools	18	22	19	15	18	92
Avg # of Tool Calls	2.3	3.2	4.1	3.7	4.2	3.5
Explicit Queries	50	50	49	50	43	242
Ambiguous Queries	48	44	49	48	44	233
Infeasible Queries	35	32	28	19	34	148
Avg # of Follow-up	2.2	2.8	2.9	3.1	3.2	2.8

Table 2: Statistical summary of ClarifyBench.

3.2 Benchmark Construction

Data Sources. ClarifyBench draws from two primary sources to ensure diversity and realism. First, we extract successfully executed tool calls from the DocPilot (Mathur et al., 2024), which provides real user interactions in document processing scenarios. Second, we leverage the Berkeley Function Calling Leaderboard (BFCL-v3) (Yan et al., 2024), which offers data across multiple domains: vehicle control, stock trading, travel planning, and file system management.

Data Augmentation. To create the comprehensive set of query types required for clarification evaluation, we employ systematic data augmentation techniques. We process *DocPilot* dataset by anonymizing user metadata, replacing specific file names and domain terms in tool calls with LLM-generated substitutes to ensure generalizability, followed by PII removal. For ambiguous queries, we randomly select upto 3 arguments from successful tool calls and obfuscate them, then prompt GPT-4o to generate five alternative user queries that omit the obfuscated information. We also generate user intent prompts using in-context learning examples to capture the original tool call semantics. For infeasible queries, we design handwritten rules based on common API errors to create tool calls that would generate failures, followed by a similar LLM-based query augmentation process. We process *BFCL-v3* using existing explicit and ambiguous parameter queries from the benchmark, ensuring sample independence by removing cases with secondary API dependencies. We apply rule-based validation and LLM judgment (via in-context learning) to identify and exclude such cases. For retained samples, we strip secondary API utterances and tool calls from ground truth annotations. User intent prompts are generated through LLM processing, and infeasible queries are constructed using domain-specific rules, mirroring the DocPilot data strategy.

Human Validation. To ensure quality and naturalness, a human annotator evaluates all LLM-generated queries using three criteria: (A) naturalness of language, (B) faithfulness to expected tool calls including all required details while excluding obfuscated parameters, and (C) for infeasible queries, presence of error-inducing requirements. The annotator selects one optimal query per sample from the five generated alternatives.

4 *Argus*

Figure 3 illustrates ***Argus***, an information-theoretic disambiguation technique that leverages structured tool argument spaces to resolve ambiguous tool calls through targeted clarification. Argus integrates into the standard Plan-Act-Observe cycle of tool-based agents, operating between planning and action execution to ensure high-confidence tool calls. Tool-based agents operate in sequential steps consisting of Reasoning/Planning, Action, and Observation. ***Argus*** enhances this cycle by introducing a disambiguation phase: following initial reasoning and planning, candidate interpretations are generated, then disambiguated through *Argus* if uncertainty exceeds thresholds, before returning final observations to the user.

Preliminaries: Tool Argument Domains. At the core of ***Argus*** lies the structured nature of tool argument spaces, which we model as domains with explicit constraints and interdependencies. Each tool argument $a_{i,j}$ for tool t_i has an associated domain $D_{t_i}(a_{i,j})$ defining its valid values, analogous to function domains in mathematics. These domains can be finite (e.g., file permissions:

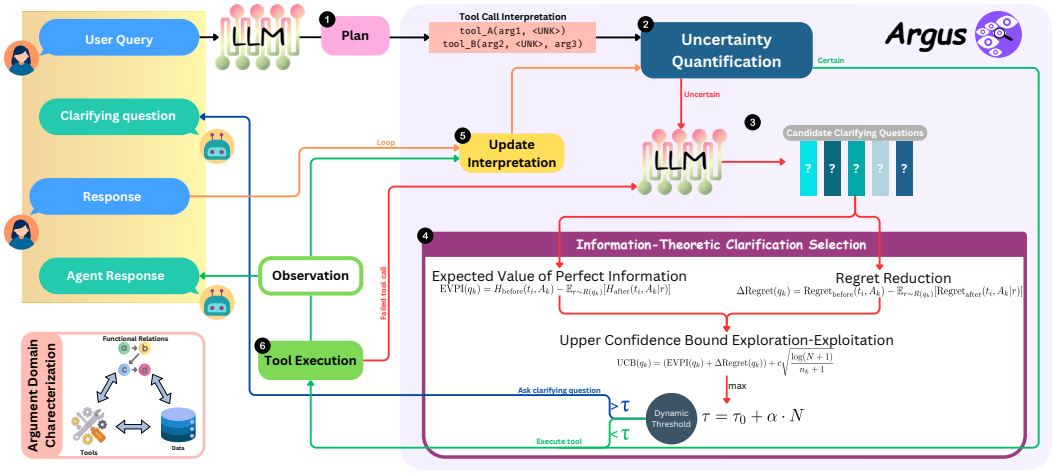

Figure 3: ***Argus***: (❶) Given a user query, an LLM reasons to generate a plan and tool calls with potential uncertain parameters. These tool calls undergo (❷) uncertainty quantification to determine if clarification is needed. When uncertainty exists, the agent uses an LLM to produce (❸) candidate clarifying questions, and scores them using (❹) information-theoretic principles (EVPI and Regret Reduction), selecting the optimal question via UCB based exploration-exploitation. The tool call's interpretation is updated based on user-response to the clarifying question (❺), and given no further uncertainty, is executed ❻.

`read`, `write`, `execute`) or infinite (e.g., file paths), and often exhibit dynamic constraints based on system state or other arguments. Effective clarifying questions strategically reduce this multidimensional argument space by constraining multiple domains simultaneously. This structured approach enables ***Argus*** to quantify uncertainty precisely over argument domains rather than operating in unstructured token space, leading to more principled and efficient disambiguation.

4.1 Uncertainty Quantification Framework

4.1.1 Candidate Interpretation Generation

The agentic system prompts the LLM to generate candidate interpretations of user queries as sequences of tool calls, with specific instructions to use `<UNK>` tokens for arguments where relevant context is absent. Given a query q, the system generates:

$$[(t_i, \mathbf{a}_i)|i = 1, 2, ..., n] \tag{1}$$

where t_i represents the tool name and $\mathbf{a}_i$ is the set of arguments. Generated tool calls are executed sequentially until an ambiguous tool call (containing `<UNK>` tokens) is encountered, as executed tool calls may provide context that resolves subsequent argument specifications.

4.1.2 Domain-Informed Uncertainty Calculation

Tool call uncertainty is assessed based on argument domain constraints. The probability of certainty p_c for each argument is defined as:

$$p_c(a_{i,j}) = \begin{cases} 1.0, & \text{if explicit} \\ \frac{1}{|D_{t_i}(a_{i,j})|}, & \text{if } 1 \leq |D_{t_i}(a)| < \infty \\ \epsilon, & \text{if } |D'_{t_i}(a)| \to \infty \end{cases} \tag{2}$$

where $D_{t_i}(a_{i,j})$ is the domain of argument $a_{i,j}$ for tool t_i, and ϵ is a small positive constant. The overall certainty of tool call t_i is calculated as:

$$H(t_i) = \prod_{j=1}^{m} p_c(a_{i,j}) \tag{3}$$

4.2 Information-Theoretic Clarification Generation

4.2.1 Candidate Question Generation

Argus generates clarification questions for uncertain tool calls by prompting an LLM with conversational context (including observations and original query), tool argument definitions, and domain descriptions. Additionally, it prompts the LLM to identify the set of target arguments the question is expected to resolve. The set of candidate clarification questions is defined as $Q = \{(q_k, t_i, A_k) \mid k = 1, 2, \ldots, l\}$, where q_k is the question text, t_i

is the candidate tool call, and $A_k \subseteq \mathbf{a}_i$ represents the subset of arguments targeted by question q_k.

4.2.2 Expected Value of Perfect Information (EVPI)

The disambiguation problem involves decision-making under uncertainty. We adopt the Expected Value of Perfect Information framework (Raiffa and Schlaifer, 2000) to quantify the potential value of acquiring additional information through user clarification. EVPI measures the expected improvement in decision quality from obtaining perfect information before making decisions. For our disambiguation system,

$$\text{EVPI}(q_k, t_i) = H(t_i) - \mathbb{E}_{r \sim R(q_k)}[H(t_i|r)] \quad (4)$$

where $H(t_i)$ represents current uncertainty of tool call t_i, and $H(t_i|r)$ is the expected posterior uncertainty after receiving response $r \in R(q_k)$, the set of responses.

4.2.3 Argument Importance and Regret Minimization

Not all arguments carry equal weight in determining tool call effectiveness. Critical arguments may significantly impact computational efficiency, result quality, or system safety, while others may have acceptable defaults. To account for varying argument importance, we introduce a regret-based formulation, inspired by (Loomes and Sugden, 1982) that models the expected loss from proceeding with uncertain values. Regret associated with a tool call is defined as:

$$\text{Regret}(t_i) = \sum_{j=1}^{m} w_{i,j} \cdot (1 - p_c(a_{i,j})) \quad (5)$$

where $w_{i,j} \in [0, 1]$ represents the importance weight assigned to argument $a_{i,j}$, and $(1-p_c(a_{i,j}))$ captures argument uncertainty. We calculate expected reduction in regret:

$$\Delta\text{Regret}(q_k) = \text{Regret}(t_i) - \mathbb{E}_{r \sim R(q_k)}[\text{Regret}(t_i|r)] \quad (6)$$

This formulation prioritizes questions addressing high-importance arguments with significant uncertainty. Importance can be user-defined or empirically derived from historical performance.

4.3 Exploration-Exploitation Trade-off in Question Selection

The sequential clarification process presents an exploration-exploitation dilemma. We adopt an Upper Confidence Bound approach to balance exploiting known high-value questions with exploring potentially valuable clarifications. The UCB score for candidate question q_k is:

$$\text{UCB}(q_k) = S(q_k) + c\sqrt{\frac{\log(N+1)}{n_k + 1}} \quad (7)$$

where $S(q_k) = \text{EVPI}(q_k) + \Delta\text{Regret}(q_k)$ combines information gain and regret reduction, c controls exploration-exploitation balance, N is total clarifications made, and n_k is the frequency of arguments targeted by q_k.

A dynamic threshold mechanism determines when to terminate clarification:

$$\tau = \tau_0 + \alpha \cdot N \quad (8)$$

where τ_0 is the initial threshold and α controls threshold increase rate. The system selects questions with highest UCB scores exceeding this threshold, naturally encoding diminishing returns in information gathering.

4.4 Response Processing and Belief Update

Upon receiving response r to question q_k, beliefs about the target tool call are updated by refining affected argument domains:

$$D'_{t_i}(a) = \begin{cases} D_{t_i}(a) \cap f_{update}(a, r), & \text{if } a \in A_k \\ D_{t_i}(a), & \text{otherwise} \end{cases} \quad (9)$$

where $f_{update}(a, r)$ extracts domain constraints from the response. Uncertainty values are recalculated based on updated domains:

$$p'_c(a) = \begin{cases} \frac{1}{|D'_{t_i}(a)|}, & \text{if } 1 \leq |D'_{t_i}(a)| < \infty \\ \epsilon, & \text{if } |D'_{t_i}(a)| \to \infty \end{cases} \quad (10)$$

Termination. The clarification process terminates when initial uncertainty falls below a fixed threshold, no questions achieve the UCB threshold, or the maximum number of questions n_s is exceeded.

Error Recovery. When tool execution fails, the system uses the failure context as an observation to either fix the failure, or determine if it can be clarified from the user, generating error-specific clarification questions incorporating both the candidate interpretation and error message: $q_{\text{error}} = f_{\text{error}}(t^*, \mathbf{a}^*, \text{error})$

LLM	Baseline	ClarifyBench - Ambiguous				ClarifyBench - Explicit				ClarifyBench - Infeasible			
		Success ↑	TMR ↑	PMR ↑	Avg #Q ↓	Success ↑	TMR ↑	PMR ↑	Avg #Q ↓	Success ↑	TMR ↑	PMR ↑	Avg #Q ↓
LLaMa 3.1 8B	Control	32.22	37.50	34.20	0.00	38.04	40.39	39.96	0.00	22.67	34.20	26.34	0.00
	ProCOT	36.91	47.30	40.10	3.55	41.19	44.12	42.88	2.05	32.98	33.10	31.45	3.74
	Active Task Disambig.	33.45	44.50	34.11	3.10	44.10	46.10	44.76	2.10	29.72	35.50	33.65	1.09
	Ask before Plan	40.47	49.80	44.77	2.90	48.01	47.13	50.12	2.00	36.89	37.80	35.89	2.20
	***Argus* (Ours)**	44.67	51.25	48.57	2.12	47.85	50.47	49.43	1.70	40.22	40.65	39.33	1.95
GPT-4o	Control	48.50	65.12	52.54	0.00	66.45	67.59	64.07	0.00	37.87	61.33	40.90	0.00
	ProCOT	58.69	68.18	59.80	3.50	68.12	70.30	68.75	2.89	57.82	63.70	60.12	3.73
	Active Task Disambig.	58.73	64.82	60.12	3.90	64.87	67.12	65.33	3.27	49.44	60.50	50.23	1.80
	Ask before Plan	60.22	69.56	61.25	3.20	70.11	74.35	71.00	2.51	54.18	64.22	61.45	2.78
	***Argus* (Ours)**	64.54	73.20	68.05	2.30	70.56	76.50	75.13	1.23	60.52	69.45	67.90	2.17

Table 3: Comparison of ***Argus*** with baselines on ClarifyBench. Best and second best results are highlighted. ***Argus*** maximizes tool call correctness while mitigating potential interaction fatigue.

Ablation	Success Rate	TMR	PMR	Avg #Q
Argus	44.23	47.36	45.71	1.92
✗EVPI	42.01	46.83	42.56	2.02
✗Δ Regret	41.78	45.94	43.44	1.95
✗Exploration Term	42.23	44.52	43.54	1.91
✗Dynamic Threshold	44.19	47.04	45.61	2.23

Table 4: Ablation results showing the impact of removing individual components. ✗indicates ablation.

Figure 4: Comparison of token usage vs. time taken. ***Argus*** achieves high efficiency and performance simultaneously.

5 Experiments

5.1 Baselines

All baselines are built on top of a common ReAct agent for comparability. We evaluate ***Argus*** against the following baselines: Control Baseline is a standard ReAct agent with no specific intervention for tool-calling disambiguation. Control baseline cannot interact with the user outside of receiving queries and delivering results. ProCOT (Deng et al., 2023) uses ProActive Chain-of-Thought reasoning to think through potential ambiguities before tool execution. Active Task Disambiguation (Kobalczyk et al., 2025) generates multiple candidate interpretations and asks questions based on variance in responses. Ask-Before-Plan (Zhang et al., 2024) instructs agents to ask clarifying questions before planning tool execution. We use GPT-4o and LlaMa 3.1 (8B) (Grattafiori et al., 2024) for all baselines.

5.2 Metrics

We evaluate performance based on the following metrics: **Success Rate (%):** The percentage of simulations with complete tool and parameter match with ground truth. **Tool Match Rate (TMR, %):** The average percentage of correctly identified tools across simulations. **Parameter Match Rate (PMR, %):** The average percentage of correctly specified parameters across simulations. **Average Number of Questions (Avg #Q):** The mean number of clarifying questions asked per simulation. For Success Rate, TMR, and PMR, higher values indicate better performance, while lower Avg #Q values are preferable.

6 Results

Main Results. Table 3 compares ***Argus*** with relevant baselines on ClarifyBench, implemented with both `LLaMa-3.1-8B-Instruct` and `GPT-4o` as the base LLMs, with `GPT-4o` serving as the user simulator in both scenarios. `Argus` demonstrates superior performance across all evaluation dimensions, achieving the highest success rates in every scenario. This highlights Argus' ability to understand user intent through principled, proactive disambiguation strategies. The control baseline, which lacks the ability to interact with users through questions, shows the lowest performance across all scenarios despite having an Average Question count of 0.00. This stark performance gap emphasizes the fundamental necessity of clarification in ambiguous user interactions. Notably, when comparing

across model scales, performance with the more powerful GPT-4o shows substantial improvements over LLaMa-3.1, with Argus maintaining its superiority regardless of the underlying model capacity. ClarifyBench Explicit functions as an upper bound for performance since all required details are already present in user queries. Consequently, we observe higher success rates, TMR, and PMR across all systems in this scenario. Despite this inherent advantage in the task setup, Argus still outperforms other approaches while maintaining lower question counts, demonstrating its efficiency even when queries contain relatively complete information. Prompting-based question-answering baselines like ProCOT and Ask-before-Plan exhibit a critical limitation: they ask substantially more questions on average without achieving proportional improvements in success metrics. This inefficiency underscores the necessity for principled uncertainty estimation and strategic disambiguation as proposed in Argus. We observe that Active-task-Disambiguation asks fewer questions for infeasible queries, demonstrating how it is not particularly suitable for dynamic tool execution environments where incorrect parameters may cause errors. This limitation becomes apparent in its significantly lower success rates for infeasible queries compared to Argus. Meanwhile, ProCOT which is a promp based CoT reasoning baseline, exhibits the highest Average Question count for infeasible queries, as it defaults to asking users for clarification rather than attempting to debug errors independently. This behavior increases user interaction burden without necessarily resolving the underlying issues.

Ablation Study. We ablate different components from ***Argus*** to understand their impact on overall performance. Table 4 presents results using LLaMa3.1-8B as the base LLM, with metrics averaged across the three query splits. To ablate EVPI, Δ Regret, and the Exploration term, we replace them with a constant denoting their theoretical maximum value. To ablate Dynamic Threshold, we set $\alpha = 0$. Results show that removing EVPI, Δ Regret, or the Exploration term causes noticeable drops in overall performance, confirming their importance in Argus's question selection strategy. Most interestingly, ablating the Dynamic Threshold causes a significant increase in the average number of questions asked without substantially affecting success rates, demonstrating its crucial role in making ***Argus*** more interaction-efficient without compromising effectiveness.

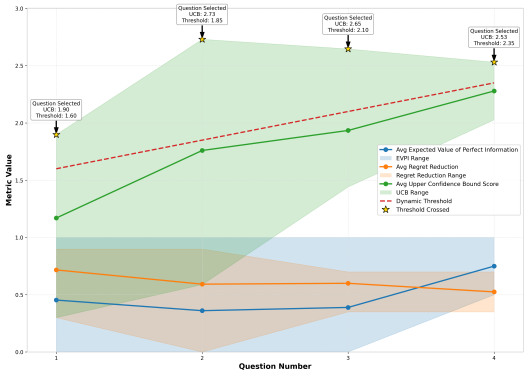

Figure 5: Evolution of question selection metrics used by ***Argis*** during interactive user simulations.

Efficiency. Fig. 4 plots all five methods on a two-dimensional time vs. token-usage plane, with bubble area proportional to end-task performance. ProCOT and Ask-before-Plan—both chaining-based reasoning baselines—incur the highest token counts due to their repeated, lengthy text reasoning steps, placing them in the "Token Intensive" quadrant despite strong success rates. Active Task Disambiguation falls into the "Time Intensive" region: it generates multiple candidate completions per query to estimate uncertainty and issues clarification questions, driving up runtime even though it economizes on tokens. By contrast, Control remains fast but achieves only modest performance. In contrast, **Argus** balances token use and latency, while achieving superior task performance.

Evolution. Fig 5 refects ***Argus***' question selection metrics over steps. The first questions have a lower UCB value, owing to $N = 0$, with UCB value peaking subsequently and then reducing steadily, demonstrating the diminishing value of asking questions.

7 Conclusion

We introduce ***Argus***, a principled information-theoretic approach to disambiguate tool-augmented agentic queries through uncertainty quantification. By operating directly on structured tool argument domains rather than unstructured token spaces, Argus combines information-theoretic measures with regret minimization to optimize question selection. Extensive experiments on ClarifyBench demonstrate that Argus significantly outperforms existing clarification strategies diverse query types and domains, achieving higher task success and reducing unnecessary clarification.

8 Limitations

Despite Argus's strong performance, several limitations should be acknowledged. First, our approach assumes the availability of well-defined tool schemas, and user-defined importance scores with explicitly structured argument domains, which may not always be available for complex or rapidly evolving APIs. Second, while our information-theoretic approach performs well within the tested domains, it may face challenges in extremely high-dimensional argument spaces where calculating expected values becomes computationally intensive. Third, ClarifyBench, while more realistic than previous benchmarks, still represents a simulation of user behavior and may not fully capture the diversity and complexity of real-world human responses. Additionally, our current implementation relies on existing foundation models for natural language processing, inheriting any biases or limitations present in these underlying models.

9 Ethics Statement

Our research does not use any personally identifiable information (PII) and all datasets employed in this work are used in accordance with their respective licenses (Apache 2.0). `Argus` is designed primarily for deployment in collaborative AI assistance contexts where resolving ambiguity enhances productivity and user experience while minimizing unnecessary interaction. The system's core approach of reducing clarification questions through principled uncertainty estimation promotes more equitable access to AI assistance by respecting users' time and cognitive resources. While `Argus` significantly reduces interaction burden, we recommend appropriate transparency about system limitations and human oversight when deploying in sensitive contexts. Furthermore, we encourage ongoing evaluation to ensure that question selection patterns do not reflect or amplify biases present in underlying models or training data.

References

Jayachandu Bandlamudi, Ritwik Chaudhuri, Neelamadhav Gantayat, Kushal Mukherjee, Prerna Agarwal, Renuka Sindhgatta, and Sameep Mehta. 2025. A framework for testing and adapting rest apis as llm tools.

Yang Deng, Wenqiang Lei, Wenxuan Zhang, Wai Lam, and Tat-Seng Chua. 2022. Pacific: towards proactive conversational question answering over tabular and textual data in finance. *arXiv preprint arXiv:2210.08817*.

Yang Deng, Lizi Liao, Liang Chen, Hongru Wang, Wenqiang Lei, and Tat-Seng Chua. 2023. Prompting and evaluating large language models for proactive dialogues: Clarification, target-guided, and non-collaboration. *Preprint*, arXiv:2305.13626.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Kung-Hsiang Huang, Akshara Prabhakar, Sidharth Dhawan, Yixin Mao, Huan Wang, Silvio Savarese, Caiming Xiong, Philippe Laban, and Chien-Sheng Wu. 2024. Crmarena: Understanding the capacity of llm agents to perform professional crm tasks in realistic environments. *arXiv preprint arXiv:2411.02305*.

Katarzyna Kobalczyk, Nicolas Astorga, Tennison Liu, and Mihaela van der Schaar. 2025. Active task disambiguation with llms. *arXiv preprint arXiv:2502.04485*.

Sizhe Liu, Yizhou Lu, Siyu Chen, Xiyang Hu, Jieyu Zhao, Yingzhou Lu, and Yue Zhao. 2024. Drugagent: Automating ai-aided drug discovery programming through llm multi-agent collaboration. *arXiv preprint arXiv:2411.15692*.

Graham Loomes and Robert Sugden. 1982. Regret theory: An alternative theory of rational choice under uncertainty. *The economic journal*, 92(368):805–824.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, and 1 others. 2024. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *arXiv preprint arXiv:2408.04682*.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. Agentboard: An analytical evaluation board of multi-turn llm agents. *arXiv preprint arXiv:2401.13178*.

Puneet Mathur, Alexa Siu, Varun Manjunatha, and Tong Sun. 2024. Docpilot: Copilot for automating pdf edit workflows in documents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 232–246.

Cheng Qian, Bingxiang He, Zhong Zhuang, Jia Deng, Yujia Qin, Xin Cong, Zhong Zhang, Jie Zhou, Yankai Lin, Zhiyuan Liu, and 1 others. 2024. Tell me more! towards implicit user intention understanding of language model driven agents. *arXiv preprint arXiv:2402.09205*.

Howard Raiffa and Robert Schlaifer. 2000. *Applied statistical decision theory*. John Wiley & Sons.

Sudha Rao and Hal Daumé III. 2018. Learning to ask good questions: Ranking clarification questions using neural expected value of perfect information. *arXiv preprint arXiv:1805.04655*.

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.

Sanidhya Vijayvargiya, Xuhui Zhou, Akhila Yerukola, Maarten Sap, and Graham Neubig. 2025. Interactive agents to overcome ambiguity in software engineering.

Chenyu Wang, Weixin Luo, Sixun Dong, Xiaohua Xuan, Zhengxin Li, Lin Ma, and Shenghua Gao. 2025.

Wenxuan Wang, Juluan Shi, Chaozheng Wang, Cheryl Lee, Youliang Yuan, Jen-tse Huang, and Michael R Lyu. 2024. Learning to ask: When llms meet unclear instruction. *arXiv preprint arXiv:2409.00557*.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*.

Jingjing Xu, Yuechen Wang, Duyu Tang, Nan Duan, Pengcheng Yang, Qi Zeng, Ming Zhou, and Xu Sun. 2019. Asking clarification questions in knowledge-based question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1618–1629.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Berkeley function calling leaderboard. https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html.

Guoli Yin, Haoping Bai, Shuang Ma, Feng Nan, Yanchao Sun, Zhaoyang Xu, Shen Ma, Jiarui Lu, Xiang Kong, Aonan Zhang, and 1 others. 2024. Mmau: A holistic benchmark of agent capabilities across diverse domains. *arXiv preprint arXiv:2407.18961*.

Yangyang Yu, Haohang Li, Zhi Chen, Yuechen Jiang, Yang Li, Denghui Zhang, Rong Liu, Jordan W Suchow, and Khaldoun Khashanah. 2024. Finmem: A performance-enhanced llm trading agent with layered memory and character design. In *Proceedings of the AAAI Symposium Series*, volume 3, pages 595–597.

Michael JQ Zhang and Eunsol Choi. 2023. Clarify when necessary: Resolving ambiguity through interaction with lms. *arXiv preprint arXiv:2311.09469*.

Xuan Zhang, Yang Deng, Zifeng Ren, See Kiong Ng, and Tat-Seng Chua. 2024. Ask-before-plan: Proactive language agents for real-world planning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10836–10863.

A Algorithmic Formulation

Algorithms 1–4 represent a formal algorithmic summarization of our method.

B Theoretical Analysis of Argus Question Scoring

B.1 Component Bounds

We derive theoretical bounds for each component of the UCB scoring function to establish the range of possible values and convergence properties.

B.1.1 EVPI and Regret Reduction Bounds

Theorem B.1 *The Expected Value of Perfect Information satisfies* $0 \leq EVPI(q_k) \leq 1$.

Proof B.1.1 *By definition, EVPI measures the difference between expected utility under perfect information and expected utility under current beliefs:*

$$EVPI(q_k) = \mathbb{E}_\theta[\max_a U(a,\theta)] - \max_a \mathbb{E}_\theta[U(a,\theta)]$$

The lower bound follows from Jensen's inequality: since $\max_a \mathbb{E}_\theta[U(a,\theta)] \leq \mathbb{E}_\theta[\max_a U(a,\theta)]$, *we have* $EVPI(q_k) \geq 0$.

For the upper bound, assume utilities are normalized to $[0,1]$. *Then* $\mathbb{E}_\theta[\max_a U(a,\theta)] \leq 1$ *and* $\max_a \mathbb{E}_\theta[U(a,\theta)] \geq 0$, *yielding* $EVPI(q_k) \leq 1$.

Theorem B.2 *The regret reduction satisfies* $0 \leq \Delta Regret(q_k) \leq 1$.

Proof B.2.1 *Regret reduction measures the decrease in worst-case regret from asking question* q_k. *Since asking questions cannot increase regret,* $\Delta Regret(q_k) \geq 0$. *The maximum reduction occurs when uncertainty is completely resolved, bounded by the initial regret which is at most 1 under normalized utilities.*

B.1.2 UCB Exploration Term Bounds

The exploration component $c\sqrt{\frac{\log(N+1)}{n_k+1}}$ exhibits different behaviors in extreme cases:

Case 1: Never asked ($n_k = 0$):

$$c\sqrt{\frac{\log(N+1)}{1}} = c\sqrt{\log(N+1)}$$

Case 2: Always asked ($n_k = N$):

$$c\sqrt{\frac{\log(N+1)}{N+1}}$$

Algorithm 1 Multi-Request Simulation Process

procedure EXECUTESIMULATION($\mathcal{S}$) ▷ $\mathcal{S}$ represents the simulation scenario
  Initialize agent $\mathcal{A}$, environment $\mathcal{E}$, user model $\mathcal{U}$
  $\mathcal{R} \leftarrow \{r_0, r_1, \ldots, r_n\}$ ▷ Request sequence
  $\mathcal{C} \leftarrow \emptyset$ ▷ Conversation history
  for each request $r_i \in \mathcal{R}$ **do**
    $\mathcal{T}_i \leftarrow \emptyset$ ▷ Turn sequence for request i
    $q_{current} \leftarrow r_i$ ▷ Current query state
    $clarification\_count \leftarrow 0$
    while $clarification\_count < \tau_{max}$ **and not** terminated **do**
      $response \leftarrow \mathcal{A}(q_{current}, \mathcal{C})$
      if $response \in \Phi_{success}$ **then** ▷ Successful completion
        Record completion in $\mathcal{T}_i$
        break
      else if $response \in \Phi_{clarification}$ **then** ▷ Needs clarification
        $clarification \leftarrow \mathcal{U}(response.question, \mathcal{S})$
        if $clarification = \perp$ **then** ▷ User cannot provide clarification
          Record incomplete in $\mathcal{T}_i$
          break
        end if
        $q_{current} \leftarrow Enrich(r_i, clarification)$
        $clarification\_count \leftarrow clarification\_count + 1$
      else
        Record failure in $\mathcal{T}_i$
        break
      end if
    end while
    $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{T}_i$
  end for
  $evaluation \leftarrow Evaluate(\mathcal{C}, \mathcal{S}.ground\_truth)$
  return $\{\mathcal{C}, evaluation\}$
end procedure

Algorithm 2 ***Argus***

procedure ARGUS($query, context$) ▷ ReAct paradigm: Reason-Act-Observe + ***Argus***
  $\Omega \leftarrow context.observations$ ▷ Observational memory
  $step \leftarrow 0$
  while $step < \sigma_{max}$ **and not** terminated **do** ▷ REASON PHASE: Generate action hypothesis
    $\theta \leftarrow Reason(query, \Omega)$ ▷ Chain of thought
    $\alpha \leftarrow SelectAction(\theta, \mathcal{A}_{available})$ ▷ Proposed action
    ▷ DISAMBIGUATE PHASE: Resolve parameter uncertainty
    $\xi \leftarrow AssessUncertainty(\alpha)$ ▷ Uncertainty measure
    if $\xi < \delta_{threshold}$ **then** ▷ High uncertainty detected
      $q_{clarification} \leftarrow Disambiguation(\alpha, query)$
      return $\{status : clarification, question : q_{clarification}\}$
    end if
    ▷ ACT PHASE: Execute selected action
    $\rho \leftarrow Execute(\alpha, \mathcal{E})$ ▷ Action execution result
    ▷ ERROR RECOVERY PHASE
    if $\rho \in \mathcal{E}_{error}$ **then**
      $recovery \leftarrow RecoveryStrategy(\rho, query, context)$
      if $recovery.type = clarification$ **then**
        return $\{status : error\_clarification, question : recovery.question\}$
      else
        $\Omega \leftarrow \Omega \cup \{recovery.observation\}$
        continue
      end if
    end if
    ▷ COMPLETION CHECK
    if $\alpha = final\_answer$ **then**
      return $\{status : completed, result : \alpha.answer\}$
    end if
    ▷ OBSERVE PHASE: Update memory
    $\Omega \leftarrow \Omega \cup \{\rho.observation\}$
    $step \leftarrow step + 1$
  end while
  return $\{status : completed, result : "max\_steps\_reached"\}$
end procedure
procedure REASON($query, \Omega$) ▷ Deliberative reasoning over available information
  $knowledge \leftarrow \bigcup_i \omega_i$ where $\omega_i \in \Omega$
  $\theta \leftarrow IntegrateInformation(query, knowledge, \mathcal{K}_{background})$
  return θ ▷ Coherent reasoning chain
end procedure

Algorithm 3 Parameter Disambiguation: Top-Level Procedure

procedure DISAMBIGUATION($\alpha, query$)
  $\Xi \leftarrow \{\}$ ▷ Parameter uncertainty mapping
  for each parameter $p_j \in \alpha.parameters$ **do**
    $\xi_j \leftarrow MeasureUncertainty(p_j, \alpha.tool)$
    $\omega_j \leftarrow GetImportance(p_j, \alpha.tool)$
    $\Xi[p_j] \leftarrow \{\xi_j, \omega_j\}$
  end for
  $\xi_{overall} \leftarrow \sum_j \omega_j \cdot \xi_j / \sum_j \omega_j$
  if $\xi_{overall} \geq \delta_{threshold}$ **then**
    return $\{needs\_clarification : false, certainty : \xi_{overall}\}$
  end if
  $\mathcal{Q}_{candidates} \leftarrow GenerateCandidates(query, \alpha, \Xi)$
  $\mathcal{M} \leftarrow \{\}$
  for each question $q_k \in \mathcal{Q}_{candidates}$ **do**
    $\mathcal{M}[q_k].evpi \leftarrow EVPI(q_k, \alpha)$
    $\mathcal{M}[q_k].regret \leftarrow RegretReduction(q_k, \alpha)$
    $\mathcal{M}[q_k].exploration \leftarrow ExplorationBonus(q_k)$
  end for
  $q^* \leftarrow \arg\max_{q_k} CompositeScore(\mathcal{M}[q_k])$
  if $q^* \neq \perp$ **then**
    return $\{needs\_clarification : true, question : q^*, certainty : \xi_{overall}\}$
  else
    return $\{needs\_clarification : false, certainty : \xi_{overall}\}$
  end if
end procedure

Algorithm 4 MeasureUncertainty

procedure MEASUREUNCERTAINTY($parameter, tool$)
  $value \leftarrow parameter.value$
  $domain \leftarrow GetDomain(parameter, tool)$
  if $value = \perp$ **or** $value = unknown$ **then**
    return 1.0
  end if
  switch ($domain.type$)
    case finite_set:
  return $DomainUncertainty(value, domain.values)$
    case continuous_range:
  return $RangeUncertainty(value, domain.bounds)$
    case context_dependent:
  return $ContextualUncertainty(value, domain, \mathcal{C}_{current})$
    default:
  return $BaselineUncertainty(value)$
  end switch
end procedure

As $N \to \infty$, the second case approaches 0, while the first grows as $c\sqrt{\log N}$.

B.1.3 Combined UCB Score Bounds

Theorem B.3 *The total UCB score satisfies:*

$$0 \leq UCB(q_k) \leq 2 + c\sqrt{\log(N+1)}$$

Proof B.3.1 *The lower bound follows from non-negativity of all components. For the upper bound:*

$$UCB(q_k) = EVPI(q_k) + \Delta Regret(q_k) + c\sqrt{\frac{\log(N+1)}{n_k+1}}$$

Using the bounds from previous theorems and the maximum exploration term when $n_k = 0$:

$$\begin{aligned} UCB(q_k) &\leq 1 + 1 + c\sqrt{\log(N+1)} \\ &= 2 + c\sqrt{\log(N+1)} \end{aligned}$$

B.2 Dynamic Threshold Justification

The linear threshold form $\tau = \tau_0 + \alpha \cdot N$ captures the intuition of diminishing returns in information gathering:

Early stages (N small): $\tau \approx \tau_0$, providing a low threshold that encourages exploration when little is known about question values.

Later stages (N large): τ grows linearly, creating a higher bar for questions as the system accumulates knowledge and marginal information value decreases.

Relationship to score bounds: Given that UCB scores are bounded by $2+c\sqrt{\log(N+1)}$, a reasonable choice of α should ensure the threshold grows slower than the maximum possible score, maintaining a non-zero probability of asking questions. This suggests $\alpha = O(\sqrt{\log N}/N)$ to balance between being too permissive (asking too many questions) and too restrictive (missing valuable clarifications).

Optimality intuition: Under a simple cost model where each question costs c_1 and each unit of remaining regret costs c_2, the optimal threshold approximately satisfies:

$$\tau^*(N) \approx \frac{c_1}{c_2} + \mu_S(N)$$

where $\mu_S(N)$ is the expected score of a random question. Since $\mu_S(N)$ typically decreases with N (as high-value questions are asked first), a linear increase in τ can approximate this optimal policy.

B.3 Parameter Selection Guidelines

B.3.1 Exploration Coefficient c

From multi-armed bandit theory, the optimal exploration coefficient scales as $c^* = O(\sqrt{\log K})$ where K is the number of potential questions. In practice:

- **Small c (< 0.5)**: Heavy exploitation, may miss valuable unexplored questions
- **Large c (> 2.0)**: Heavy exploration, asks many low-value questions
- **Recommended range**: $c \in [0.5, 1.5]$ balances exploration-exploitation effectively

B.3.2 Threshold Parameters

Initial threshold τ_0: Should be set relative to typical EVPI/regret scales:

- Too low: Asks questions with minimal information value
- Too high: Misses valuable early clarifications
- **Recommended**: $\tau_0 \in [1.0, 2.0]$ for normalized utilities

Growth rate α: Controls how aggressively the system reduces question-asking over time:

- **Empirically effective range**: $\alpha \in [0.1, 0.3]$
- Lower values: More questions, higher information gain, increased user burden
- Higher values: Fewer questions, faster termination, potential information loss

Trade-off considerations: The choice of (τ_0, α) should reflect the relative costs of asking questions versus making decisions under uncertainty. Applications with high decision stakes should use lower α values to gather more information, while time-sensitive applications should increase α to reduce clarification overhead.

C Implementation Details

C.1 Computational Resources

Table 5 summarizes the computational budget for our experiments.

GPU	VRAM (GB)	GPU Hours	Purpose
RTX A6000	48	24	LLaMa3.1-8B Inference

Table 5: Compute budget

C.2 Hyperparameters

For the Argus framework, we set exploration-exploitation parameter $c = 0.8$, base threshold $\tau = 0.6$, and decay rate $\alpha = 0.15$. The system was configured with

C.2.1 ReAct Agent Prompts

Reasoning Prompt This prompt is used in the main reasoning phase of the ReAct agent to decide which tool to use next based on the current state of the conversation.

```
You are an AI assistant helping with a
    user request.
SYSTEM CONTEXT:
You have access to the following tool
    domain:
{plugin_descriptions}
Request: {request}
Previous observations:
{obs_text}
Available tools:
{tool_registry.get_tool_descriptions()}
Think step by step about what tool to
    use next. Consider the plugin
    context above to understand the
    capabilities available to you. If
    you have enough information to
    provide a final answer, use the
    final_answer tool.
Respond in JSON format:
{
"reasoning": "Your step-by-step thinking
    ",
"tool_call": {
"tool_name": "name_of_tool",
"arguments": {
"arg1": "value1",
"arg2": "value2"
}
}
}
```

Error Recovery Prompt Used when a tool execution fails to determine if the error can be resolved automatically.

```
You are helping fix a failed tool call.
Original Request: {request}
Tool Information:
{tool_info or f"Tool: {tool_name}"}
Error Details:
{error_result.message}
Based on the error and tool information,
     can you suggest how to fix this?
Respond in JSON format:
{
"can_fix": true/false,
"reasoning": "explanation of what went
    wrong and how to fix it",
"suggested_action": "retry_with_changes"
     or "different_tool" or "
    need_clarification",
"observation": "observation to add to
    context for next reasoning step"
}
If you cannot determine a fix from the
    available information, set can_fix
    to false.
```

C.2.2 Question Generation Prompts for Argus

Used to generate clarification questions when there is uncertainty about tool arguments.

```
You are an AI assistant that helps users
     by understanding their queries and
    executing tool calls.
{conversation_history}Original user
    query:
"{user_query}"
Based on the query, I've determined that
     the following tool calls are needed
    , but some arguments are uncertain:
Tool Calls:
{tool_calls}
Detailed Tool Documentation:
{tool_documentation}
Uncertain Arguments:
{uncertain_args}
Your task is to generate clarification
    questions that would help resolve
    the uncertainty about specific
    arguments.
Instructions:

Generate questions that are clear,
    specific, and directly address the
    uncertain arguments
Each question should target one or more
    specific arguments
Questions should be conversational and
    easy for a user to understand
For each question, specify which tool
    and argument(s) it aims to clarify.
Generate 5 diverse questions.
Keep in mind the the arguments you wish
    to clarify, their domains etc.

Return your response as a JSON object
    with the following structure:
{
"questions": [
{
"question": "A clear question to ask the
     user",
"target_args": [["tool_name", "arg_name
    "], ["tool_name", "other_arg_name"]]
}
// ... 5 total questions
]
}
Ensure that each question targets at
    least one uncertain argument.
```

C.2.3 User Simulator

The simulator takes a language model provider, ground truth data, and user intent as inputs. It maintains the conversation state and ensures responses are consistent with the user's information. The core of the simulation lies in two prompt templates that instruct a language model to act as a user:

```
You are simulating a user who is
    interacting with an AI assistant.
Original query: "{self.original_query}"
User's intent for the CURRENT request: {
    self.user_intent}
Information needed for the CURRENT
    request (do not reveal future
    intentions):
{current_turn_ground_truth}
Additional context:
{self.context}
The AI assistant has asked the following
     specific question:
"{question}"
Generate a realistic user response to
    this SPECIFIC question. The response
     should:

Be natural and conversational
ONLY provide information that directly
    answers the specific question asked
NOT mention any future requests or
    intentions the user might have
ONLY focus on the current task, not on
    future tasks
Be concise and to the point

IMPORTANT: Never reveal future
    intentions. Respond ONLY to the
    specific question asked.
NEVER BREAK CHARACTER. DO NOT THINK OUT
    LOUD. Respond directly as the user
    would:
```

This template ensures the simulator provides natural, conversational responses that only address the specific question without revealing future intentions. For generating follow-up requests, the simulator uses this template:

```
You are simulating a user who is
    interacting with an AI assistant.
Original query: "{self.original_query}"
User's intent: {self.user_intent}
Previous conversation:
{formatted_history}
Based on the conversation so far and the
     user's intent, decide if the user
    would have a follow-up request.
Consider:

Has everything the user wanted been
    accomplished?
Is there a logical next step the user
    might want to take?
Has the agent clearly indicated that
    they've completed all necessary
    tasks?

If you believe the user would have a
    follow-up request, provide it in a
    natural, conversational way.
If you believe the conversation is
    complete, respond with "
    CONVERSATION_COMPLETE".
NEVER BREAK CHARACTER, DO NOT THINK!
Decision:
```

This template helps the simulator determine whether to generate a follow-up request based on the conversation context and predefined potential follow-ups. The User Simulator isolates ground truth information for each conversation turn, ensuring only relevant information is revealed at appropriate times. It tracks the original query, user intent, ground truth for tool calls, completed tool calls, potential follow-up queries, and the current conversation turn. By providing consistent, realistic user responses, the simulator allows for reproducible evaluation of clarification strategies across multiple scenarios.

D Benchmark Details

D.1 Benchmark Domains

This appendix describes the key characteristics of each API domain used in our experiments, detailing their initialization parameters, state management, and tool specifications.

Gorilla File System Plugin (GFS). The Gorilla File System API simulates a UNIX-like file system with a hierarchical directory structure. It maintains state through:

- Directory structure with nested files and subdirectories
- Current working directory pointer
- Each file contains content as strings

The plugin provides 18 tools implementing common file system operations such as navigation, file creation, modification, and content manipulation. Each tool supports parameters relevant to file system operations, such as file names, directory paths, and content strings. Table 10 provides detailed information about these tools and their parameter domains.

The GFS plugin's domains depend heavily on the current state of the file system. Domain updates revolve primarily around available files and directories in the current working directory, as outlined in Table 11.

Document Processing. The Document API simulates operations for PDF document manipulation. Its state consists of:

- Number of pages in the current document
- PDF filename metadata

- Operation-specific context for page-based operations

The plugin provides 18 document manipulation tools including conversion, annotation, redaction, and page manipulation functions. Parameters include page numbers, text content, formatting options, and file paths. Table 7 details the tools and their parameter domains.

Domain updates in the Document Plugin focus on page numbers and ranges, adapting dynamically to changes in document length when pages are added or deleted, as shown in Table 11.

Vehicle Control. The Vehicle Control API simulates an automotive control system with:

- Engine state (running or stopped)
- Door lock status for each door
- Fuel level (ranging from 0 to 50 gallons)
- Battery voltage
- Climate control settings
- Brake systems (pedal position and parking brake)
- Lighting systems
- Navigation state

This plugin implements 24 vehicle control tools that manipulate different aspects of the vehicle, including engine operations, door management, climate control, lighting, braking systems, and navigation. Table 9 details the specific tools and their parameter domains.

Vehicle Control domain updates primarily concern contextual constraints such as brake pedal position for engine start, door states, and fuel level requirements, as referenced in Table 11.

Travel. The Travel API simulates a travel booking and management system with:

- Credit card registry and balances
- Flight booking records
- User information (first name, last name)
- Budget limits
- Available routes with pricing data

The plugin provides 15 tools for travel-related operations, including flight bookings, credit card management, budget settings, and travel information queries. Table 6 details these tools and their parameter domains.

Domain updates in the Travel Plugin focus on available credit cards, booking IDs, and airport codes for valid routes, as detailed in Table 11.

Trading Bot. The Trading Bot simulates a stock trading platform with:

- Account information and balance
- Order records (pending, completed, cancelled)
- Stock data with prices and metrics
- Watchlist of stocks
- Transaction history
- Market status (open/closed)

This plugin provides 19 trading tools for account management, order placement, stock information retrieval, and market analysis. Table 8 lists the specific tools and their parameter domains.

Trading Plugin domain updates primarily involve available stocks, watchlist items, and order IDs, adapting to user actions like placing orders or modifying watchlists, as referenced in Table 11.

All plugins follow a consistent pattern for state initialization through configuration objects, domain updates based on state changes, and parameter validation. The dynamic nature of these domains presents particular challenges for language model interactions, as valid parameter values continuously evolve during conversations based on system state changes.

D.2 Human Annotation

We employed two graduate student annotators, aged 22-25. The annotators were proficient in English, and have proficiency in Python (relevant to test tool calls). The annotators were fairly compensated at the standard Graduate Assistant hourly rate, following their respective graduate school policies. Fig 6 shows a summary of the annotator guidelines.

D.3 Tool Call Corruption Heuristics

We handcrafted rues to corrupt validated tool calls in the ground truth data, to construct ClarifyBench-Infeasible.

Tool Name	Argument	Description	Domain Type	Domain Values	Data Dep.	Required	Importance
get_budget_fiscal_year	lastModifiedAfter	Date filter for fiscal years	string	Any date string	N	N	0.5
	includeRemoved	Include removed fiscal years	string	Any string	N	N	0.5
register_credit_card	card_number	Credit card number	string	Any card number	N	Y	0.9
	expiration_date	Card expiration (MM/YYYY)	string	MM/YYYY format	N	Y	0.8
	cardholder_name	Name on card	string	Any name string	N	Y	0.8
	card_verification_number	CVV code	numeric_range	[100, 999]	N	Y	0.8
get_flight_cost	travel_from	Departure airport code	string*	3-letter codes	Y	Y	0.9
	travel_to	Arrival airport code	string*	3-letter codes	Y	Y	0.9
	travel_date	Travel date	string	YYYY-MM-DD	N	Y	0.9
	travel_class	Seat class	finite	[economy, business, first]	N	Y	0.8
get_credit_card_balance	card_id	Credit card identifier	string*	Card ID list	Y	Y	0.9
book_flight	card_id	Payment card ID	string*	Card ID list	Y	Y	0.9
	travel_date	Travel date	string	YYYY-MM-DD	N	Y	0.9
	travel_from	Departure airport	string*	Airport codes	Y	Y	0.9
	travel_to	Arrival airport	string*	Airport codes	Y	Y	0.9
	travel_class	Seat class	finite	[economy, business, first]	N	Y	0.8
	travel_cost	Flight cost	numeric_range	[0, 10000]	N	Y	0.9
retrieve_invoice	booking_id	Booking identifier	string*	Booking ID list	Y	N	0.9
	insurance_id	Insurance identifier	string*	Insurance ID list	Y	N	0.7
list_all_airports	*No arguments*						
cancel_booking	booking_id	Booking to cancel	string*	Booking ID list	Y	Y	0.9
compute_exchange_rate	base_currency	Source currency	finite	[USD, RMB, EUR, JPY, GBP, CAD, AUD, INR, RUB, BRL, MXN]	N	Y	0.9
	target_currency	Target currency	finite	[USD, RMB, EUR, JPY, GBP, CAD, AUD, INR, RUB, BRL, MXN]	N	Y	0.9
	value	Amount to convert	numeric_range	[0, 1000000]	N	Y	0.9
verify_traveler_information	first_name	Traveler's first name	string	Any name	N	Y	0.9
	last_name	Traveler's last name	string	Any name	N	Y	0.9
	date_of_birth	Birth date	string	YYYY-MM-DD	N	Y	0.9
	passport_number	Passport number	string	Any passport ID	N	Y	0.9
set_budget_limit	budget_limit	Budget limit in USD	numeric_range	[0, 10000]	N	Y	0.9
get_nearest_airport_by_city	location	City name	finite	[Rivermist, Stonebrook, ...]	N	Y	0.9
purchase_insurance	insurance_type	Type of insurance	finite	[basic, premium, deluxe]	N	Y	0.8
	booking_id	Booking identifier	string*	Booking ID list	Y	Y	0.9
	insurance_cost	Insurance cost	numeric_range	[0, 1000]	N	Y	0.9
	card_id	Payment card ID	string*	Card ID list	Y	Y	0.9
contact_customer_support	booking_id	Booking reference	string*	Booking ID list	Y	Y	0.9
	message	Support message	string	Any message text	N	Y	0.9
get_all_credit_cards	*No arguments*						

Table 6: Travel Plugin API: Complete Tool and Argument Specification with Domain Dependencies

Tool Name	Argument	Description	Domain Type	Domain Values	Data Dep.	Required	Importance
duplicate	output_filename	Name of duplicate file	string	Any filename	N	Y	0.8
rename	output_filename	New filename	string	Any filename	N	Y	0.8
search	object_name	Search term/object	string	Any search term	N	Y	0.9
count_pages	*No arguments*						
compress_file	output_filename	Compressed output name	string	Any filename	N	N	0.6
convert	format	Target format	finite	[pptx, doc, png, jpeg, tiff]	N	Y	0.9
	output_filename	Output filename	string	Any filename	N	Y	0.7
	zip	Zip output files	boolean	[true, false]	N	N	0.4
add_comment	page_num	Page number	numeric_range*	[1, num_pages]	Y	Y	0.9
	coordinates	Comment position [x,y]	list	[x, y] coordinates	N	Y	0.6
	font_size	Font size (points)	numeric_range	[8, 72]	N	Y	0.5
redact_page_range	start	Start page (inclusive)	numeric_range*	[1, num_pages]	Y	Y	0.9
	end	End page (inclusive)	numeric_range*	[1, num_pages]	Y	Y	0.9
redact_text	start	Start page	numeric_range*	[1, num_pages]	Y	Y	0.9
	end	End page	numeric_range*	[1, num_pages]	Y	Y	0.9
	object_name	Text to redact (list)	list	List of text strings	N	Y	0.9
	overwrite	Overwrite original	boolean	[true, false]	N	Y	0.7
	output_pathname	Output filename	string	Any filename	N	N	0.7
highlight_text	start	Start page	numeric_range*	[1, num_pages]	Y	Y	0.9
	end	End page	numeric_range*	[1, num_pages]	Y	Y	0.9
	object_name	Text to highlight (list)	list	List of text strings	N	Y	0.9
	overwrite	Overwrite original	boolean	[true, false]	N	Y	0.7
	output_pathname	Output filename	string	Any filename	N	N	0.7
underline_text	start	Start page	numeric_range*	[1, num_pages]	Y	Y	0.9
	end	End page	numeric_range*	[1, num_pages]	Y	Y	0.9
	object_name	Text to underline (list)	list	List of text strings	N	Y	0.9
	overwrite	Overwrite original	boolean	[true, false]	N	Y	0.7
	output_pathname	Output filename	string	Any filename	N	N	0.7
extract_pages	start	Start page	numeric_range*	[1, num_pages]	Y	Y	0.9
	end	End page	numeric_range*	[1, num_pages]	Y	Y	0.9
	overwrite	Overwrite original	boolean	[true, false]	N	Y	0.7
	output_pathname	Output filename	string	Any filename	N	N	0.7
delete_page	page_num	Page to delete	numeric_range*	[1, num_pages]	Y	Y	0.9
	overwrite	Overwrite original	boolean	[true, false]	N	Y	0.7
	output_pathname	Output filename	string	Any filename	N	N	0.7
delete_page_range	start	Start page	numeric_range*	[1, num_pages]	Y	Y	0.9
	end	End page	numeric_range*	[1, num_pages]	Y	Y	0.9
	overwrite	Overwrite original	boolean	[true, false]	N	Y	0.7
	output_pathname	Output filename	string	Any filename	N	N	0.7
add_signature	page_num	Page for signature	numeric_range*	[1, num_pages]	Y	Y	0.9
	position	Signature position	finite	[top-left, top-middle, ...]	N	Y	0.7
	overwrite	Overwrite original	boolean	[true, false]	N	Y	0.7
	output_pathname	Output filename	string	Any filename	N	N	0.7
add_page_with_text	text_content	Page text content	string	Any text content	N	Y	0.9
	font_size	Text font size	numeric_range	[8, 72]	N	Y	0.6
	page_num	Insert position	numeric_range*	[1, num_pages+1]	Y	Y	0.8
add_watermark	watermark_text	Watermark text	string	Any text	N	Y	0.9
	transparency	Transparency level	numeric_range	[0.0, 1.0]	N	Y	0.6
add_password	password	PDF password	string	Any password string	N	Y	0.9

Table 7: Document Plugin API: Complete Tool and Argument Specification with Domain Dependencies

Tool Name	Argument	Description	Domain Type	Domain Values	Data Dep.	Required	Importance
get_current_time			*No arguments*				
update_market_status	current_time_str	Time in HH:MM AM/PM	string	HH:MM AM/PM format	N	Y	0.8
get_symbol_by_name	name	Company name	string	Any company name	N	Y	0.9
get_stock_info	symbol	Stock symbol	string*	Available stock symbols	Y	Y	0.9
get_order_details	order_id	Order identifier	numeric_range*	Existing order IDs	Y	Y	0.9
cancel_order	order_id	Order to cancel	numeric_range*	Existing order IDs	Y	Y	0.9
place_order	order_type	Buy or Sell	finite	[Buy, Sell]	N	Y	0.9
	symbol	Stock symbol	string*	Available stocks	Y	Y	0.9
	price	Price per share	numeric_range	[0.01, 10000.0]	N	Y	0.8
	amount	Number of shares	numeric_range	[1, 10000]	N	Y	0.8
make_transaction	xact_type	Transaction type	finite	[deposit, withdrawal]	N	Y	0.9
	amount	Transaction amount	numeric_range	[0.01, 1000000.0]	N	Y	0.9
get_account_info			*No arguments*				
fund_account	amount	Funding amount	numeric_range	[0.01, 1000000.0]	N	Y	0.9
remove_stock_from_watchlist	symbol	Stock to remove	string*	Watchlist stocks	Y	Y	0.9
get_watchlist			*No arguments*				
get_order_history			*No arguments*				
get_transaction_history	start_date	Start date filter	string	YYYY-MM-DD format	N	N	0.7
	end_date	End date filter	string	YYYY-MM-DD format	N	N	0.7
update_stock_price	symbol	Stock symbol	string*	Available stocks	Y	Y	0.9
	new_price	New stock price	numeric_range	[0.01, 10000.0]	N	Y	0.9
get_available_stocks	sector	Market sector	finite	[Technology, Automobile, Healthcare, Finance, Energy]	N	Y	0.8
filter_stocks_by_price	stocks	Stock list to filter	list	List of stock symbols	N	Y	0.9
	min_price	Minimum price	numeric_range	[0.01, 10000.0]	N	Y	0.8
	max_price	Maximum price	numeric_range	[0.01, 10000.0]	N	Y	0.8
add_to_watchlist	stock	Stock to add	string*	Available stocks	Y	Y	0.9
notify_price_change	stocks	Stocks to monitor	list	List of stock symbols	N	Y	0.9
	threshold	Change threshold (%)	numeric_range	[0.01, 100.0]	N	Y	0.8

Table 8: Trading Plugin API: Complete Tool and Argument Specification with Domain Dependencies

Tool Name	Argument	Description	Domain Type	Domain Values	Data Dep.	Required	Importance
startEngine	ignitionMode	Engine ignition mode	finite	[START, STOP]	N	Y	0.9
fillFuelTank	fuelAmount	Fuel to add (gallons)	numeric_range*	[0, 50-current_fuel]	Y	Y	0.8
lockDoors	unlock	Lock or unlock	boolean	[true, false]	N	Y	0.8
	door	Doors to operate	list*	[driver, passenger, rear_left, rear_right]	Y	Y	0.9
adjustClimateControl	temperature	Target temperature	numeric_range	[-10, 50]	N	Y	0.8
	unit	Temperature unit	finite	[celsius, fahrenheit]	N	N	0.6
	fanSpeed	Fan speed (0-100)	numeric_range	[0, 100]	N	N	0.6
	mode	Climate mode	finite	[auto, cool, heat, defrost]	N	N	0.7
get_outside_temperature_from_google			*No arguments*				
get_outside_temperature_from_weather_com			*No arguments*				
setHeadlights	mode	Headlight mode	finite	[on, off, auto]	N	Y	0.8
displayCarStatus	option	Status display option	finite	[fuel, battery, doors, climate, headlights, parkingBrake, brakePedal, engine]	N	Y	0.8
activateParkingBrake	mode	Brake mode	finite	[engage, release]	N	Y	0.8
pressBrakePedal	pedalPosition	Pedal position (0-1)	numeric_range	[0, 1]	N	Y	0.8
releaseBrakePedal			*No arguments*				
setCruiseControl	speed	Cruise speed (mph)	finite*	[0, 5, 10, ..., 120]	Y	Y	0.8
	activate	Activate cruise	boolean*	[true, false]	Y	Y	0.8
	distanceToNextVehicle	Following distance (m)	numeric_range	[0, 1000]	N	Y	0.7
get_current_speed			*No arguments*				
display_log	messages	Log messages	list	List of strings	N	Y	0.7
estimate_drive_feasibility_by_mileage	distance	Distance in miles	numeric_range	[0, 10000]	N	Y	0.8
liter_to_gallon	liter	Liters to convert	numeric_range	[0, 1000]	N	Y	0.6
gallon_to_liter	gallon	Gallons to convert	numeric_range	[0, 1000]	N	Y	0.6
estimate_distance	cityA	First city zipcode	finite	[83214, 74532, 56108, ...]	N	Y	0.8
	cityB	Second city zipcode	finite	[83214, 74532, 56108, ...]	N	Y	0.8
get_zipcode_based_on_city	city	City name	finite	[Rivermist, Stonebrook, ...]	N	Y	0.8
set_navigation	destination	Destination address	string	Street, city, state format	N	Y	0.8
check_tire_pressure			*No arguments*				
find_nearest_tire_shop			*No arguments*				

Table 9: Vehicle Control Plugin API: Complete Tool and Argument Specification with Domain Dependencies

Tool Name	Argument	Description	Domain Type	Domain Values	Data Dep.	Required	Importance
pwd	*No arguments*						
ls	a	Show hidden files	boolean	[true, false]	N	N	0.3
cd	folder	Directory to change to	string*	Available directories + [.., /]	Y	Y	0.9
mkdir	dir_name	New directory name	string	Any valid directory name	N	Y	0.8
touch	file_name	New file name	string	Any valid filename	N	Y	0.8
echo	content	Text content	string	Any text string	N	Y	0.9
	file_name	Output file (optional)	string	Any filename	N	N	0.7
cat	file_name	File to display	string*	Available files	Y	Y	0.9
find	path	Search starting point	string	Any path	N	N	0.6
	name	Search pattern	string	Any search pattern	N	N	0.8
wc	file_name	File to count	string*	Available files	Y	Y	0.9
	mode	Count mode	finite	[l, w, c]	N	N	0.6
sort	file_name	File to sort	string*	Available files	Y	Y	0.9
grep	file_name	File to search	string*	Available files	Y	Y	0.9
	pattern	Search pattern	string	Any text pattern	N	Y	0.9
du	human_readable	Human readable format	boolean	[true, false]	N	N	0.4
tail	file_name	File to display	string*	Available files	Y	Y	0.9
	lines	Number of lines	numeric_range	[1, 100]	N	N	0.5
diff	file_name1	First file	string*	Available files	Y	Y	0.9
	file_name2	Second file	string*	Available files	Y	Y	0.9
mv	source	Source file/directory	string*	Available items	Y	Y	0.9
	destination	Destination name	string*	Available items + new names	Y	Y	0.9
rm	file_name	File/directory to remove	string*	Available items	Y	Y	0.9
rmdir	dir_name	Directory to remove	string*	Available directories	Y	Y	0.9
cp	source	Source file/directory	string*	Available items	Y	Y	0.9
	destination	Destination name	string*	Available items + new names	Y	Y	0.9

Table 10: File System Plugin API: Complete Tool and Argument Specification with Domain Dependencies

Plugin	Update Trigger	Dynamic Domain Updates	Affected Operations
Travel			
	Credit card registration	Card IDs → available payment methods	book_flight, get_credit_card_balance, purchase_insurance
	Flight booking	Booking IDs → cancellable/retrievable bookings	cancel_booking, retrieve_invoice, contact_customer_support
	Budget setting	Budget limits → financial constraints	All cost-related operations
	Route updates	Airport codes → valid travel routes	get_flight_cost, book_flight
Document			
	Page operations	Page count → valid page numbers	All page-specific operations
	Document loading	Total pages → range constraints	add_comment, delete_page, etc.
	Cache invalidation	State changes → domain refresh	Page-changing operations
Trading			
	Order placement	Order IDs → manageable orders	get_order_details, cancel_order
	Stock updates	Available stocks → tradeable symbols	place_order, get_stock_info
	Watchlist changes	Watchlist → removable stocks	remove_stock_from_watchlist
Vehicle			
	Fuel level changes	Current fuel → addable amount	fillFuelTank
	Door state changes	Door status → operable doors	lockDoors
	Engine state	Running/stopped → cruise control availability	setCruiseControl
File System			
	Directory navigation	Current contents → available items	cd, cat, mv, cp, rm
	File operations	File list → operable files	File-specific operations
	Directory changes	Directory list → navigable paths	cd, rmdir
	State synchronization	FS changes → domain cache invalidation	All state-changing operations

Table 11: Dynamic Domain Update Rules and Triggers Across Plugin System

GorillaFileSystem For the file system API, we implemented four primary corruption strategies:

- *Invalid File Name Corruption* targeting functions like `mkdir`, `touch`, and `cat` by inserting forbidden characters (e.g., `|`, `/`, `\`, `?`);
- *Path Traversal Corruption* for `cd`, `mv`, `cp`, and `find` operations by inserting relative paths (`../`) or absolute paths (`/root/`);
- *Non-existent Files Corruption* for file operation functions by generating random names or modifying existing names;
- *Duplicate Creation Corruption* for `mkdir` and `touch` operations by using existing file/directory names.

DocumentPlugin For the document manipulation API, we implemented three corruption strategies:

- *Invalid Page Range Corruption* for functions like `add_comment` and `delete_page` by setting zero/negative values or exceeding total pages;
- *Invalid Formats Corruption* for `convert` operations by using unsupported formats or partial strings;
- *Out of Range Values Corruption* for parameters like `font_size` and `transparency` by exceeding min/max bounds or using negative values.

VehicleControlAPI For the vehicle control API, we focused on two corruption categories:

- *Invalid Ranges Corruption* for functions like `fillFuelTank` and `adjustClimateControl` by exceeding capacity or using negative values;
- *Invalid Enums Corruption* for operations like `startEngine` and `setHeadlights` by supplying wrong enum values or case mismatches.

TravelAPI For the travel booking API, we implemented three corruption strategies:

- *Financial Constraints Corruption* for functions like `book_flight` by exceeding available balance or using negative values;
- *Invalid Routes Corruption* for route parameters by using non-existent airport codes or identical from/to locations;
- *Non-existent Booking Corruption* for functions like `cancel_booking` by generating random non-existent IDs.

TradingBot For the stock trading API, we implemented three corruption strategies:

- *Invalid Symbols Corruption* for functions like `get_stock_info` by using non-existent symbols or malformed formats;
- *Financial Validation Corruption* for `place_order` and related functions by using negative values or amounts exceeding account balance;
- *Order State Conflicts Corruption* for `cancel_order` operations by referencing completed orders or using malformed order IDs.

Human Annotation Guidelines

Objective:
Annotators must evaluate five LLM-generated queries per sample. Each query is scored on three dimensions: (A) Naturalness of language, (B) Faithfulness to the expected tool call, and (C) Executability/Validity. Additionally, annotators must check for removal of Personally Identifiable Information (PII), assess tool call feasibility, and select one optimal query per sample.

Evaluation Rubric

Criterion	Score 5	Score 4	Score 3	Score 2	Score 1
A. Naturalness	Fully fluent, natural, human-like	Minor awkwardness or stiffness	Understandable but robotic	Clearly awkward or difficult to read	Unintelligible or nonsensical
B. Faithfulness	Perfect match to expected tool call; all required arguments present	Mostly aligned; minor phrasing or parameter issues	Some omissions or hallucinations; core logic intact	Major deviations from expected tool behavior	Entirely incorrect or misleading tool structure
C. Executability	Fully executable; properly structured and valid	Executes with minor issues or missing defaults	Partially executable with moderate corrections needed	Major issues preventing execution	Unexecutable or contradicts tool logic/API

Required Checks

- **PII Removal:** Ensure no personal identifiers (names, emails, phone numbers, IDs) are present Flag these queries for further processing.
- **Tool Call Validation:** If feasible, simulate or run tool calls to confirm validity and argument correctness.
- **Error Identification:** Mark and annotate any queries with logical inconsistencies, invalid parameters, or unsupported constraints.

Figure 6: Summary of instructions given to human annotators.