HiSA: Hierarchical State Abstraction for Scalable GUI Agents

Weiming Li, Helen Paik, Yulei Sui
School of Computer Science and Engineering
The University of New South Wales
Sydney, Australia
{weiming.li1, h.paik, y.sui}@unsw.edu.au

Abstract

Multimodal GUI agents generally operate on raw visual and textual observations, which creates a fundamental scalability challenge. While current state-of-the-art frameworks predominantly rely on inference-intensive test-time scaling or the accumulation of unbounded raw logs to maintain task coherence, we attribute the underlying bottleneck to insufficient state abstraction. To address this, we propose HiSA, a hierarchical state abstraction approach that actively restructures knowledge rather than passively retaining historical information by organizing raw histories into a three-level hierarchy of abstracted steps, refined contexts, and induced patterns. By synthesizing high-dimensional observations into compact semantic states, HiSA decouples reasoning efficacy from context length, enabling precise and scalable decision-making as interaction histories grow. When evaluating using Spider2-V, our approach establishes a new state-of-the-art, achieving a 40.58% success rate while reducing token consumption by 69.85% and monetary costs by 55.10% compared to the best-performing baseline.

1 Introduction

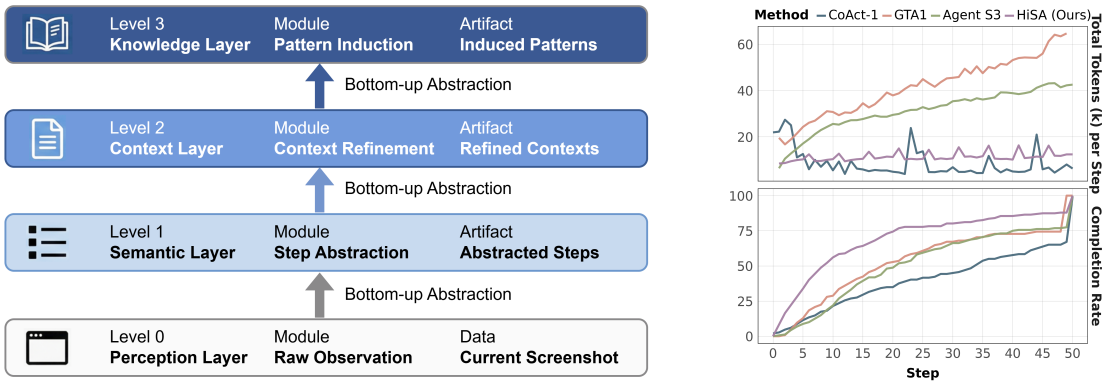
Recent multimodal large language models (MLLMs) (Achiam et al., 2023; Comanici et al., 2025; Anthropic, 2024; Wang et al., 2024) enable GUI agents to automate workflows through visual UI understanding (Gou et al., 2025; Hong et al., 2024; Xu et al., 2025; Wu et al., 2025b; Agashe et al., 2025a,b). However, these systems face significant scalability bottlenecks with unsustainable computational costs. While human operators intuitively distill high-dimensional visual information into conceptual states, current agents generally exhibit insufficient state abstraction. Consequently, to compensate for processing raw perceptual data, state-of-the-art frameworks often employ resource-intensive strategies, such as accumulating unbounded textual logs or relying on inference-heavy test-time scaling, such as parallel trajectory sampling, to extract solutions from noisy observations (Yang et al., 2025; Gonzalez-Pumariega et al., 2025). This reliance on raw data creates a dilemma where agents are either prohibitively expensive to deploy or suffer from context overload, leading to reasoning degradation in long sequences (Liu et al., 2024).

As illustrated in Figure 1(a), we propose HiSA[1], a hierarchical state abstraction framework designed to distill granular execution details into high-level state representations progressively. The process begins at the perception layer, which captures raw visual observations. Based on these inputs, the semantic layer performs step abstraction to summarize execution outcomes, transforming pixel-level data into abstracted steps. Subsequently, the context layer applies context refinement to maintain bounded complexity, recursively integrating discrete events into a coherent refined context. Finally, the knowledge layer executes pattern induction to distill insights from successes and failures, generalizing trial-and-error exploration into reusable induced patterns.

Our framework is structured into three specialized modules: (1) a *global planner* serving as the high-level reasoning engine, (2) a *visual grounder* dedicated to GUI grounding, which translates semantic instructions into pixel coordinates, and (3) a *state manager* orchestrating the abstraction logic. The state manager restructures context through hierarchical state abstraction, including step abstraction, context refinement, and pattern induction. Under this design, scalability concerns whether the cumulative reasoning cost can remain controlled as the interaction horizon grows, thereby supporting long-horizon execution within practical infer-

[1]Code available at https://github.com/liweim/HiSA

Findings of the Association for Computational Linguistics: ACL 2026, pages 11965–11985



(a) Architectural overview of the HiSA framework

(b) Token usage and completion rate

Figure 1: Overview of the proposed framework and efficiency analysis. Panel (a) illustrates the hierarchical bottom-up abstraction mechanism. Panel (b) compares computational efficiency, where the top chart presents the total token consumption per step. The bottom chart further demonstrates faster task completion relative to baselines.

ence budgets. Concretely, our framework maintains $O(N)$ text tokens with respect to the refinement interval N and $O(1)$ image tokens regardless of task duration, as illustrated in Figure 1(b).

We evaluate our approach on Spider2-V, which serves as a large-scale and complicated benchmark for real-world enterprise workflows, where human expert performance achieves a 47.66% success rate. Our method achieves 40.58% while reducing token costs by 69.85% compared with the latest state-of-the-art baseline framework Agent S3 (Gonzalez-Pumariega et al., 2025). Our contributions are summarized below.

- We identify insufficient state abstraction as the core scalability bottleneck in GUI agents and propose HiSA, which shifts from merely storing history to actively restructuring knowledge.
- We implement a three-level abstraction hierarchy that guarantees bounded token complexity of $O(1)$ image tokens and $O(N)$ text tokens.
- Our approach establishes a new state-of-the-art on the Spider2-V benchmark, achieving a strong success rate of 40.58% while significantly reducing token consumption by 69.85% compared to the best-performing baseline.

2 Related Work

2.1 Agent Scalability and Architectural Paradigms

Early research addresses long-horizon complexity through modular orchestration, where tasks are decomposed across specialized agents. Microsoft UFO (Zhang et al., 2025a) uses a dual-tier structure for application-specific execution, while CoAct-1 (Zhao et al., 2025) employs dynamic delegation to prevent context overflow. However, these methods often incur high re-initialization overhead and lose critical fine-grained visual context during role transitions. Modern frameworks like Agent S3 (Gonzalez-Pumariega et al., 2025) and GTA1 (Yang et al., 2025) prioritize performance through resource-intensive strategies such as Best-of-N sampling or parallel trajectory evaluation. These paradigms result in excessive computational costs and high inference latency driven by the linear accumulation of execution logs.

2.2 Visual Representation and Compression

Parallel to the challenges of textual log accumulation, the processing of high-dimensional visual observations introduces a scalability bottleneck. To mitigate this computational burden, ShowUI (Lin et al., 2025) clusters spatial patches, while GUI-KV (Huang et al., 2025) exploits attention sparsity to optimize visual decoding. Similarly, CoMEM-Agent (Wu et al., 2025a) compresses complete trajectories into continuous embedding tokens directly injected into input layers. These methods primarily rely on passive compression that may compromise fine-grained visual details necessary for precise control of small interface elements.

2.3 Hierarchical Planning and State Abstraction

The concept of state abstraction originates from reinforcement learning literature, where it serves as the theoretical foundation for scaling agents to high-dimensional environments. Formally, state abstraction projects the original high-dimensional state space onto a compact abstract space (Li et al.,

2006; Andre and Russell, 2002). The core objective is to reduce dimensionality while ensuring that an optimal policy derived in the abstract environment remains valid in the ground environment. While classical hierarchical reinforcement learning (HRL), such as the Options framework (Sutton et al., 1999), focuses on temporal abstraction by grouping action sequences, it often assumes a fixed state representation. Recent work posits that effective planning requires coupling temporal abstraction with state abstraction to ignore task-irrelevant features such as visual noise (Zeng et al., 2023; Pineau, 2004). Our HiSA framework introduces state abstraction into GUI agents by hierarchically abstracting both the temporal dimension through context refinement and pattern induction, and the spatial dimension through step abstraction, bridging the gap between theoretical HRL and practical LLM reasoning.

3 Hierarchical State Abstraction Framework

We instantiate the proposed approach through a collaborative multi-agent framework, as shown in Figure 2, that maps abstraction principles to distinct computational modules. Specifically, our framework is structured into three specialized modules comprising a *global planner* serving as the centralized reasoning engine, a *visual grounder* performing GUI grounding to resolve semantic instructions into executable coordinates, and a *state manager* orchestrating the hierarchical state abstraction.

3.1 Global Planner

To mitigate the context fragmentation and misalignment inherent in decentralized frameworks (Cemri et al., 2025), we design the global planner as a centralized reasoning engine. By maintaining global consistency rather than distributing state, this planner utilizes a unified multimodal context that integrates observations across four abstraction levels as shown in Figure 1(a). Upon processing this composite input, the global planner generates a JSON output specifying the *thought*, *tool*, and *input*.

The defined tools encompass five operations: *gui_action*, *bash_execution*, *wait*, *termination*, and *infeasible*. Specifically, for the *gui_code* tool that executes PyAutoGUI commands, we distinguish between coordinate-free actions such as hotkeys and coordinate-dependent actions. In the latter scenario, the planner generates code containing placeholders alongside a GUI grounding instruction delegated to the visual grounder. Complementing these GUI interactions, we draw inspiration from CoAct-1 (Song et al., 2025) to incorporate *bash_execution*, bypassing complex interface manipulations through direct command-line execution. Regarding task conclusion, *termination* signals successful completion, whereas *infeasible* allows the agent to explicitly declare a task impossible when faced with software limitations, such as unsupported application features or missing required files, mandating a justification to prevent unwarranted termination.

To ensure robustness, we implement a self-correction mechanism to autonomously rectify format violations or invalid arguments. Detailed system prompts and correction protocols are provided in Appendix A.1 and Appendix A.2.

3.2 Visual Grounder

General-purpose vision-language models often exhibit insufficient native alignment for GUI coordinate spaces (Li et al., 2025). Therefore, we delegate localization tasks to the visual grounder. By mapping the planner's semantic instructions to precise pixel coordinates, this module effectively decouples spatial execution from high-level reasoning. The associated prompts are provided in Appendix A.3.

3.3 State Manager

The state manager constitutes the core abstraction engine that implements the active knowledge restructuring paradigm. It maintains bounded token complexity through three hierarchical processes that progressively convert raw execution logs into high-level semantic states.

3.3.1 Step Abstraction

The step abstraction process transforms high-dimensional visual transitions into compact abstracted steps. At each time step, the state manager identifies the region of interest (ROI) by computing the pixel-wise difference between the pre-action and post-action screenshots. It then crops the minimal bounding box encompassing all pixels where the difference is greater than zero. The state manager analyzes this cropped ROI alongside the executed action to generate the abstracted steps. This abstraction summarizes the execution outcome and verifies whether the intended state change occurred. Appendix A.4 details the prompts for this process.

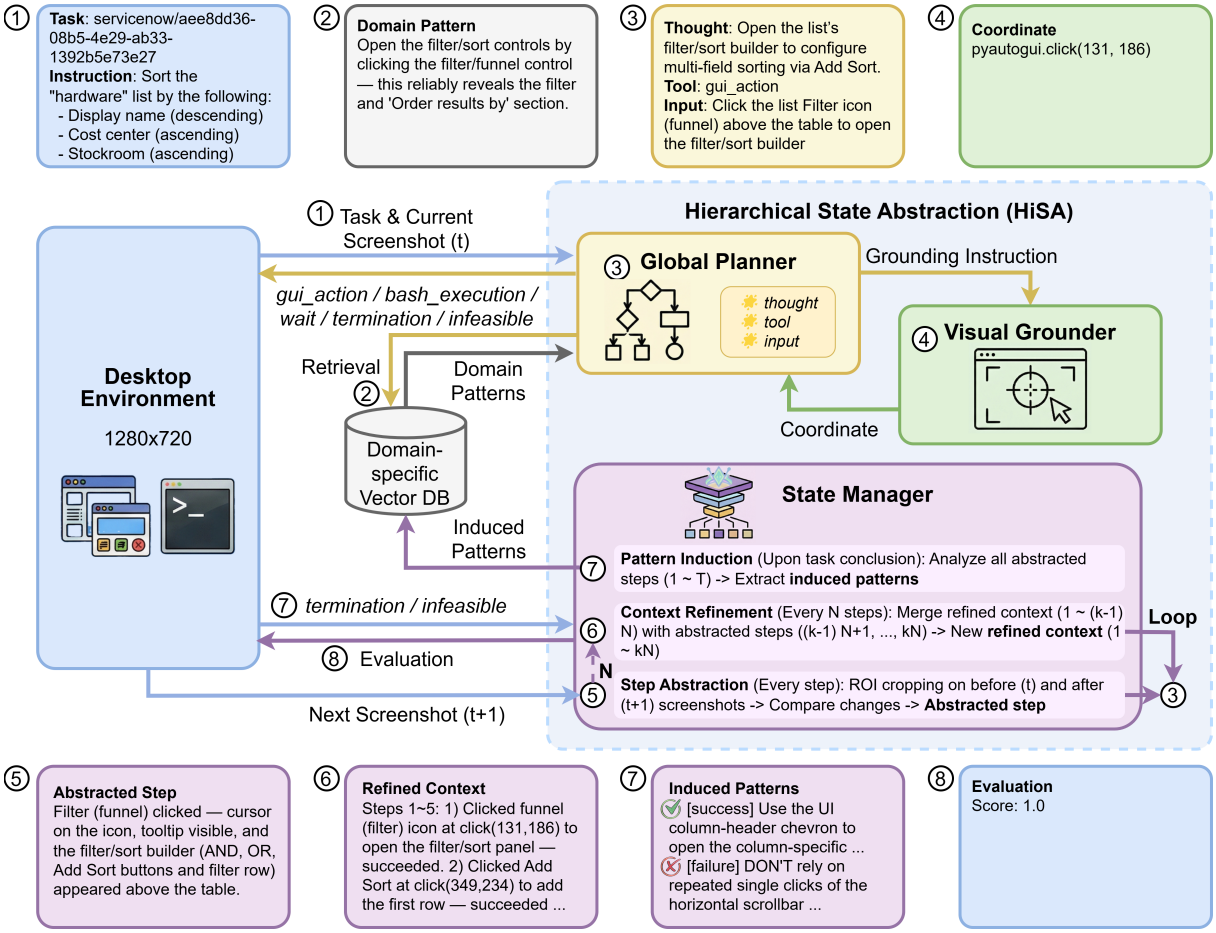


Figure 2: Architectural overview of the HiSA framework illustrated with a ServiceNow example. The system decouples execution into three modules: a global planner for high-level reasoning, a visual grounder for precise GUI grounding, and a state manager that maintains bounded context via hierarchical abstraction.

3.3.2 Context Refinement

To prevent context overflow during long-horizon tasks, the state manager performs context refinement at fixed intervals. This mechanism recursively integrates the accumulated buffer of recent abstracted steps into the existing refined context, generating a summary that retains key milestones while discarding redundant execution details. By updating the global state representation through this semantic compression, the system maintains a stable context length that captures task progress without growing linearly with the trajectory length. Specific prompts for this process are available in Appendix A.5.

3.3.3 Pattern Induction

The agent-environment interaction can be conceptualized as a reinforcement learning environment where outcomes serve as intrinsic reward signals (Zhang et al., 2025b). Leveraging this formulation, the pattern induction process distills high-reward execution trajectories into generalizable abstractions through self-verification, rather than predefined procedures or retrieval-based prompting.

Upon *termination* or *infeasible*, the state manager initiates pattern induction to analyze the trajectory and distill insights from trial-and-error exploration, identifying causal subsequences that led to successful sub-goals or critical failures. It then generalizes these specific instances into induced patterns that describe the condition, action sequence, and expected result. Success patterns encode effective workflows for specific domains, while failure patterns serve as constraints to prevent the repetition of known errors. To maintain a concise knowledge base, we calculate the cosine similarity between new and existing patterns, replacing older entries with recent instances upon detecting high semantic similarity to prioritize adaptive strategies.

For subsequent task execution, the system employs a pattern synthesis mechanism to transform these stored experiences into actionable guidance.

The state manager queries the domain-specific vector database to retrieve relevant patterns, which are then synthesized alongside expert-curated standard operating procedures (SOPs) to prime the agent. These SOPs address evaluation requirements not captured during exploration, such as ensuring all cells are executed and saved in Jupyter environments to satisfy external scoring criteria. This integration ensures the agent is guided by both data-driven heuristics and domain knowledge. Appendix A.6 presents the prompt for pattern induction, and Appendix A.7 details the pattern synthesis mechanism.

3.4 System Workflow

Our system workflow is illustrated in Figure 2. The workflow initiates at step 1, where the global planner receives the task instruction alongside the initial screenshot. In step 2, the system queries the domain-specific vector database to retrieve relevant induced patterns based on semantic similarity to guide decision-making. Step 3 constitutes the central execution phase, where the global planner adopts a ReAct-style reasoning process (Yao et al., 2022) to formulate a thought trace and select the appropriate tool. For interface manipulations requiring precise localization, the system delegates coordinate resolution to the visual grounder in step 4. Following execution, the state manager performs step abstraction in step 5 to distill the transition. Before cycling back to step 3, it conditionally executes context refinement in step 6 if the step count reaches the refinement interval N. Upon *termination* or *infeasible*, the system triggers pattern induction in step 7 to extract reusable patterns before concluding with task evaluation in step 8.

3.5 Theoretical Complexity Analysis

We instantiate the abstraction within a partially observable Markov decision process (POMDP). At step t, the agent receives an observation $O_t = \langle I_t, X_t \rangle$ comprising a screenshot I_t and a textual log X_t. The agent selects an action A_t according to a policy $\pi(A_t|S_t)$, where S_t is the state representation derived from the cumulative history H_t.

Standard approaches typically rely on a sliding window context C_t^{SW} that combines the full textual history with a visual sliding window of size W:

$$C_t^{SW} = \{X_0, \ldots, X_{t-1}\} \cup \{I_{t-W}, \ldots, I_t\} \quad (1)$$

Although the visual complexity is constant at $\mathcal{O}(W)$, where $\mathcal{O}$ denotes the asymptotic upper bound, the aggregate token complexity $T(C_t^{SW})$ scales linearly with t due to the unbounded accumulation of textual logs:

$$T(C_t^{SW}) \approx \sum_{i=0}^{t-1} |X_i| + \mathcal{O}(W) \in \mathcal{O}(t) \quad (2)$$

This linear scaling constitutes the primary bottleneck for processing long-horizon tasks.

In contrast, HiSA employs a hierarchical abstraction mechanism to restructure H_t, defining the hierarchical state S_t^{HiSA} as:

$$S_t^{HiSA} = \langle I_t, \mathcal{E}_t, R_k, P \rangle \quad (3)$$

This state comprises four abstraction levels:

- **Level 0 perception layer**: I_t represents the current raw visual observation.
- **Level 1 semantic layer**: $\mathcal{E}_t$ contains recent abstracted steps $\{E_i\}_{i=kN}^{t}$ derived from transitions $\langle O_i, A_i, O_{i+1} \rangle$, where $k = \lfloor t/N \rfloor$ denotes the current epoch based on the refinement interval N.
- **Level 2 context layer**: R_k denotes the refined contexts, which compress the history up to epoch k.
- **Level 3 knowledge layer**: P contains induced patterns retrieved from the domain-specific vector database.

To maintain bounded complexity, R_k is recursively updated via the context refinement function Φ:

$$R_k = \Phi(R_{k-1}, \{E_i\}_{i=(k-1)N}^{kN-1}) \quad (4)$$

Since the buffer size is strictly limited by N and the size of the refined contexts $|R_k|$ remains bounded through semantic compression, the total token complexity becomes independent of t:

$$T(S_t^{HiSA}) = |I_t| + \sum_{i=kN}^{t} |E_i| + |R_k| + |P| \in \mathcal{O}(N) \quad (5)$$

This effectively decouples reasoning costs from the total task duration, realizing the theoretical advantage of state abstraction.

4 Experiments

4.1 Benchmarks

We primarily evaluate performance on Spider2-V (Cao et al., 2024), which features 494 real-world data engineering tasks across 20 professional enterprise applications. This benchmark targets professional data automation and presents significantly

greater difficulty than general-purpose suites. The tasks utilize two configurations named *abstract* and *verbose*. The *abstract* setting provides only the raw task instruction, while the *verbose* mode adds step-by-step tutorials to the identical instruction. Due to two-factor authentication requirements that are incompatible with automated evaluation, we omit 40 DBT and BigQuery-related tasks, resulting in a final set of 207. We also employ OSWorld (Xie et al., 2024) to assess open-ended generalization across 369 tasks involving commonly used software.

4.2 Baselines

We compare performance against the Spider2-V baseline, referred to as the Spider2-V Agent (Cao et al., 2024) and three state-of-the-art frameworks including CoAct-1 (Song et al., 2025), GTA1 (Yang et al., 2025), and Agent S3 (Gonzalez-Pumariega et al., 2025). To enable a fair comparison of computational efficiency, we standardize the reasoning backbones by upgrading all frameworks to the GPT-5 family. This update is corroborated by results on the OSWorld leaderboard[2], where baseline methods exhibit better performance when using GPT-5 compared to their original configurations. We retain the original grounding model configurations for each baseline as these components are tightly integrated with their specific frameworks and cannot be easily replaced. We also report manually evaluated human expert performance to establish a rigorous upper bound. Appendix B provides detailed hyperparameter configurations and implementation adjustments.

4.3 Settings

All methods operate on screenshots at a 1280×720 resolution rather than the default 1920×1080. This configuration reduces image token consumption and increases the relative pixel size of interface elements. We enforce a maximum budget of 50 steps for all experiments. For the HiSA framework, the global planner utilizes GPT-5 as the reasoning backbone while the state manager employs GPT-5-mini for cost-efficient abstraction. The visual grounder leverages GTA1-7B (Yang et al., 2025) to map semantic descriptions to pixel coordinates. We generate pattern embeddings using BGE-M3 (Chen et al., 2024) and store them in a Qdrant vector database[3] with a dimension of 1024. The retrieval process extracts the top 5 relevant items with a cosine similarity threshold of 0.5. We set the deduplication threshold for pattern storage to 0.7 and the refinement interval N to 5 steps. Finally, we set the temperature to 0 for deterministic outputs and max tokens to 4096. Appendix C details the hardware specifications and pricing models.

[2] https://os-world.github.io/

[3] https://qdrant.tech/

4.4 Metrics

We evaluate task performance using success rate (SR), which represents the percentage of test samples completed within the step budget. To quantify efficiency, we report the average per-task values for monetary cost in US dollars, token consumption in thousands, and total execution steps.

4.5 Main Results

HiSA establishes a new state-of-the-art with a 40.58% SR on Spider2-V as presented in Table 1. Additionally, our framework reduces token consumption by 69.85% and monetary costs by 55.10% compared to the leading baseline Agent S3. This efficiency stems from the multi-level state abstraction, which systematically decouples reasoning costs from task duration by optimizing visual information density and contextual coherence. Detailed performance across individual application domains is provided in Appendix D. In addition to computational efficiency, HiSA demonstrates superior planning capability by reducing the average step count from 23.82 for Agent S3 to 15.34. This acceleration results from the pattern induction mechanism. By retrieving successful patterns and avoiding known failure modes, the agent bypasses the trial-and-error exploration observed in approaches without cross-task pattern retrieval.

To assess the robustness of the framework in open-ended environments, we further evaluated performance on the OSWorld benchmark as shown in Table 2, where our method achieves a competitive 59.3% SR. Although a performance gap persists relative to methods relying on inference-intensive scaling, HiSA prioritizes reasoning over compact semantic states and stable context management over exhaustive exploration, reflecting a deliberate focus on scalable deployment.

Figure 1(b) presents efficiency metrics where the top panel displays the average total token consumption per task across steps and the bottom reports cumulative completion rates. In the top panel, GTA1 incurs the steepest token usage due to inference-intensive sampling, while Agent S3 exhibits linear accumulation from passive history retention.

Method	LLM Backbone	Max Steps	SR (%) ↑	Cost ($) ↓	Tokens (K) ↓			Steps ↓
					Total	Input	Output	
Spider2-V Agent (Cao et al., 2024)	GPT-4V	15	*11.30	–	–	–	–	–
CoAct-1	GPT-5 + GPT-5-mini + OAI CUA	50	23.67	0.75	526.25	472.74	53.51	28.84
GTA1	GPT-5 + GTA1-7B	50	31.72	3.79	1115.30	835.94	279.37	26.17
Agent S3	GPT-5 + UI-TARS-1.5-7B	50	38.96	0.98	607.18	575.89	31.29	23.82
HiSA (Ours)	GPT-5 + GPT-5-mini + GTA1	50	**40.58**$_{\pm 0.97}$	**0.44**$_{\pm 0.03}$	**183.05**$_{\pm 12.92}$	**141.30**$_{\pm 10.12}$	**41.75**$_{\pm 2.80}$	**15.34**$_{\pm 0.64}$
Human	–	–	47.66	–	–	–	–	10.98

Table 1: Performance and efficiency comparison on Spider2-V abstract subset. We report the success rate (SR) alongside average per-task metrics, including monetary cost in USD, token consumption, and execution steps. The monetary cost represents the end-to-end expenditure. Results for our method are averaged over two independent runs; human performance is manually evaluated as an upper bound. Best results are in bold. Results marked with * are from the original paper; other baselines are reproduced using a standardized backbone.

Method	LLM Backbone	SR (%) @ 50 Steps ↑	SR (%) @ 100 Steps ↑
UI-TARS-1.5-7B (Qin et al., 2025)	UI-TARS-1.5-7B	–	27.4
Agent S2 (Agashe et al., 2025a)	Gemini-2.5-Pro	45.8	–
Agent S2 (Agashe et al., 2025a)	GPT-5	46.3	48.8
Jedi-7B (Xie et al., 2025)	o3	50.6	51.0
GTA1-7B (Yang et al., 2025)	o3	48.6	53.1
Agent S2.5 (Agashe et al., 2025a)	o3	54.2	56.0
CoAct-1 (Song et al., 2025)	o3	56.4	59.9
Agent S2.5 (Agashe et al., 2025a)	GPT-5	–	58.4
GTA1-7B (Yang et al., 2025)	GPT-5	–	61.0
GTA1-32B (Yang et al., 2025)	GPT-5	–	62.0
Agent S3 (Gonzalez-Pumariega et al., 2025)	GPT-5	61.1	62.6
Agent S3 w/ bBoN (N=10) (Gonzalez-Pumariega et al., 2025)	GPT-5	**63.5**	**69.9**
HiSA (Ours)	GPT-5	58.7	59.3

Table 2: Comparison with state-of-the-art methods on the OSWorld benchmark. Dash line indicates results not reported in original papers, while the best results are highlighted in bold.

CoAct-1 displays fluctuating spikes triggered by sub-task synchronization. Although maintaining lower per-step token usage, its total consumption exceeds HiSA due to significantly extended step counts. In contrast, HiSA maintains a consistent sawtooth pattern characterized by periodic drops at refinement boundaries. This demonstrates that the framework limits context growth by compressing history into a bounded state representation. The bottom panel illustrates that HiSA significantly surpasses all baselines in execution speed. Notably, it completes 50% of tasks in fewer than 15 steps, indicating superior convergence speed. These results suggest that the three-level design offers a favorable balance, as fewer levels weaken functional separation while more levels introduce unnecessary overhead.

To further examine whether hierarchical state abstraction leads to information loss in long-horizon settings, we group Spider2-V tasks into three equal-frequency buckets according to execution length: Steps 2–7, Steps 8–11, and Steps 12–53. As shown in Table 3, although all methods degrade as task length increases, HiSA does not exhibit disproportionate degradation and remains competitive in longer-horizon settings.

Model	Steps 2–7	Steps 8–11	Steps 12–53
CoAct-1	37.93	10.71	9.21
GTA1	43.28	21.43	14.47
Agent S3	46.73	**27.38**	22.37
HiSA	**51.15**	26.19	**23.03**

Table 3: Success rate (%) across equal-frequency execution-length buckets on Spider2-V. Bold denotes the highest result in each bucket.

We further report the average token usage by module to quantify the computational overhead introduced by the State Manager, as summarized in Table 4. The results show that the State Manager introduces moderate overhead while remaining lighter than the Global Planner.

Component	Total Tokens(k)	Prompt Tokens(k)	Completion Tokens(k)
Global Planner	101.27	68.86	32.41
Visual Grounder	32.51	32.37	0.14
State Manager	49.27	40.07	9.20

Table 4: Component-wise token-related metrics in HiSA on Spider2-V.

4.6 Ablation Studies

We conduct an isolated ablation study to validate architectural premises as shown in Table 5. Operating at original resolution degrades SR by 6.28% as smaller relative elements challenge the grounder, while increasing token usage by a relative 46.25% due to excessive image patches. Similarly, removing ROI cropping increases token consumption by a relative 31.14% due to background redundancy, and sacrifices 2.9% in SR by losing the noise-filtering benefits.

Removing step abstraction causes token consumption to peak with a relative 59.43% surge, verifying that transforming visual transitions into semantic abstracts primarily decouples reasoning costs from context length. Furthermore, the absence of context refinement necessitates a fallback to a sliding window strategy, which disrupts temporal continuity. This planning inefficiency increases step counts by a relative 66.29% and degrades the SR by 4.83%.

Finally, pattern induction shifts planning to experience-based execution. Its exclusion increases trajectory length by a relative 9.58% and reduces SR by 6.76%, confirming that retrieving shortcuts effectively bypasses trial-and-error exploration.

Separately, we analyze the refinement interval impact in Table 6. The results show that $N = 5$ yields the highest SR by effectively balancing context currency and compression overhead. Smaller intervals introduce redundancy while larger ones reduce short-term memory precision, indicating that an appropriate frequency minimizes overhead without compromising semantic integrity.

4.7 Qualitative Analysis

We examine a representative multi-column sorting task selected from Spider2-V to illustrate the efficacy of our framework. The objective requires sorting a hardware inventory list by display name in descending order, followed by cost center and stockroom in ascending order. As visualized in Figure 3, the top panel displays the baseline Agent S3

Configuration	SR (%) ↑	Input Tokens (K) ↓	Steps ↓
Full System	**40.58**	**141.30**	**15.34**
Perception Layer (level 0)			
- w/ Original Resolution	34.30	206.66	18.47
- w/o ROI Cropping	37.68	185.31	17.03
Abstraction Layers (level 1 – 3)			
- w/o Step Abstraction	33.33	225.28	26.02
- w/o Context Refinement	35.75	205.23	25.51
- w/o Pattern Induction	33.82	144.32	16.81

Table 5: Isolated ablation study on Spider2-V. We remove individual components from the full system to evaluate their marginal contributions. *w/ Original Resolution* operates at 1920x1080 pixels. *w/o ROI Cropping* processes uncropped full screenshots. *w/o Step Abstraction* uses raw execution logs. *w/o Context Refinement* employs a sliding window of window size 5. *w/o Pattern Induction* disables cross-task retrieval and pattern induction.

Refinement Interval N	SR (%) ↑	Input Tokens (K) ↓	Steps ↓
3	37.68	152.33	18.04
5	**40.58**	**141.30**	**15.34**
7	38.65	174.46	18.47
9	36.72	149.20	18.43

Table 6: Analysis of the refinement interval N on Spider2-V Abstract subset where the parameter specifies the step stride for context refinement.

correctly processing visible columns before falling into a visual search cycle while attempting to locate the off-screen stockroom column. The red dashed box highlights how the agent consumes the step budget on low-level interaction strategies, including repetitive horizontal scrolling and iterative viewport rescaling. These attempts fail to resolve the visibility constraint, which results in failure after 50 steps. Conversely, the bottom panel depicts HiSA retrieving an effective strategy to bypass pixel-level search entirely. The agent navigates directly to the filter builder interface and executes the sorting logic through semantic verification within the green dashed box to complete the task in only 13 steps. These contrasting behaviors highlight that HiSA improves long-horizon execution by restructuring historical interaction into reusable semantic patterns that support strategy-level decisions. Additional case studies demonstrating the effectiveness of pattern induction across different domains are detailed in Appendix E.

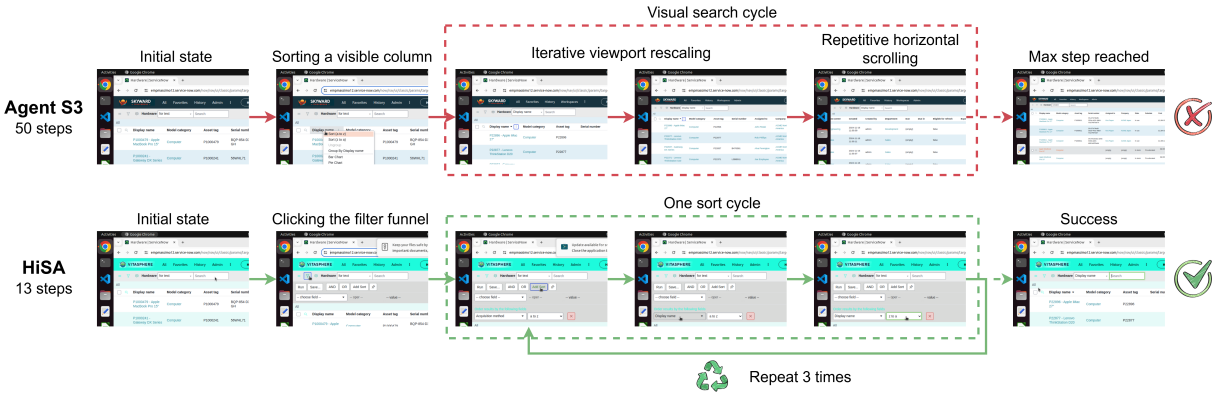


Figure 3: Qualitative comparison of execution trajectories where HiSA leverages hierarchical state abstraction and retrieved patterns to bypass pixel-level and complete the task efficiently. In contrast, the baseline Agent S3 initially processes visible elements but subsequently enters a visual search cycle involving repeated scrolling and viewport rescaling, ultimately leading to task failure.

5 Conclusion

This work addresses the scalability bottleneck arising from the reliance on passive history retention in current GUI agents. To resolve this, we introduce HiSA, a hierarchical state abstraction method that shifts the paradigm toward active knowledge restructuring. By organizing execution history into abstracted steps, refined contexts, and induced patterns, our framework decouples reasoning efficacy from task length. Empirical evaluation on Spider2-V demonstrates that this framework establishes a new state-of-the-art with a strong success rate of 40.58% while reducing token consumption by 69.85% and monetary costs by 55.10% compared to the best-performing baseline. These results demonstrate that decoupling semantic reasoning from raw execution history is essential for scaling agents to complex, long-horizon tasks. Future work will optimize for local deployment to minimize latency and explore multi-modal state representations to augment the current text-centric abstraction.

Limitations

The primary limitation of this study concerns the scope of the evaluation environments. While Spider2-V and OSWorld serve as robust benchmarks, they currently focus on Ubuntu and web-based interfaces. As a result, the generalizability of the framework to other operating systems, such as Windows or macOS, as well as to mobile platforms, has yet to be established. In addition, evaluating the framework on only two benchmarks may not fully capture the breadth of edge cases present in real-world applications.

A technical constraint arises from the lossy nature of the context refinement mechanism. While the system is designed to retain semantic relevance, the abstraction process may result in the omission of critical details, such as temporary file paths or specific configuration settings. This phenomenon effectively violates the Markov property, as the refined state may not fully represent the necessary history. This leads to potential failures in tasks that require the exact retrieval of long-term information. Nevertheless, our empirical results do not indicate disproportionate degradation on longer-horizon tasks, suggesting that the current abstraction scheme preserves most decision-critical information in practice.

Finally, the current implementation faces practical challenges regarding latency and accessibility. The reliance on external API calls introduces substantial latency, which renders it unsuitable for real-time applications. Additionally, the reliance on the proprietary backbone restricts reproducibility for researchers without access to high-end APIs. The exclusive focus on English-centric interfaces also limits the framework's generalizability to multilingual environments.

Ethical Considerations

The deployment of autonomous GUI agents involves significant responsibility regarding operational safety beyond the risks of adversarial surveillance or identity theft. An agent granted write access to a file system could inadvertently delete critical data or corrupt software environments due to hallucinated commands. To mitigate this risk, we recommend implementing human-in-the-loop

protocols where high-stakes actions require explicit user confirmation. Comprehensive audit logging is essential for accountability and for diagnosing accidental autonomous actions.

We position this technology as an assistive tool to augment human productivity rather than replace employment, though we acknowledge the potential for economic displacement in repetitive administrative tasks. Conversely, this technology offers potential benefits for accessibility by liberating users from the constraints of manual input. HiSA enables individuals with motor impairments to navigate complex software interfaces using natural language and broadens accessibility to digital tools.

While HiSA significantly reduces token consumption by over 60% compared to baselines to lower the immediate energy cost of inference, it relies on substantial computational resources for reasoning. The environmental footprint of training and serving these large foundation models remains a concern. Future work should focus on distilling these capabilities into smaller and more energy-efficient local models to minimize the carbon impact.

Acknowledgments

We thank the anonymous reviewers for their comments and suggestions. This work is partially supported by the Australian Research Council (Grant DP250101396). The generative AI tool (ChatGPT) was used for language polishing and improving the readability of the paper.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025a. Agent s: An open agentic framework that uses computers like a human. In *The Thirteenth International Conference on Learning Representations*.

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025b. Agent s2: A compositional generalist-specialist framework for computer use agents. In *Second Conference on Language Modeling*.

David Andre and Stuart J Russell. 2002. State abstraction for programmable reinforcement learning agents. In *Aaai/iaai*, pages 119–125.

Anthropic. 2024. The claude 3 model family: Opus, Sonnet, Haiku.

Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xiong Xinzhuang, Hanchong Zhang, Wenjing Hu, Yuchen Mao, Tianbao Xie, Hongshen Xu, Danyang Zhang, Sida Wang, Ruoxi Sun, Pengcheng Yin, Caiming Xiong, Ansong Ni, Qian Liu, and 4 others. 2024. Spider2-v: How far are multimodal agents from automating data science and engineering workflows? In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, and 1 others. 2025. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Gonzalo Gonzalez-Pumariega, Vincent Tu, Chih-Lun Lee, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025. The unreasonable effectiveness of scaling agents for computer use. *arXiv preprint arXiv:2510.02250*.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2025. Navigating the digital world as humans do: Universal visual grounding for GUI agents. In *The Thirteenth International Conference on Learning Representations*.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and 1 others. 2024. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14281–14290.

Kung-Hsiang Huang, Haoyi Qiu, Yutong Dai, Caiming Xiong, and Chien-Sheng Wu. 2025. Gui-kv: Efficient gui agents via kv cache with spatio-temporal awareness. *arXiv preprint arXiv:2510.00536*.

Lihong Li, Thomas J Walsh, and Michael L Littman. 2006. Towards a unified theory of state abstraction for mdps. *AI&M*, 1(2):3.

Weiming Li, Yan Shao, Jing Yang, Yujing Lu, Ling Zhong, Yuhan Wang, and Manni Duan. 2025. How auxiliary reasoning unleashes gui grounding in vlms. *arXiv preprint arXiv:2509.11548*.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. 2025. Showui: One vision-language-action model for gui visual agent. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 19498–19508.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Joelle Pineau. 2004. *Tractable planning under uncertainty: exploiting structure*. Carnegie Mellon University.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, and 1 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.

Linxin Song, Yutong Dai, Viraj Prabhu, Jieyu Zhang, Taiwei Shi, Li Li, Junnan Li, Silvio Savarese, Zeyuan Chen, Jieyu Zhao, and 1 others. 2025. Coact-1: Computer-using agents with coding as actions. *arXiv preprint arXiv:2508.03923*.

Richard S Sutton, Doina Precup, and Satinder Singh. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, and 1 others. 2024. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Wenyi Wu, Kun Zhou, Ruoxin Yuan, Vivian Yu, Stephen Wang, Zhiting Hu, and Biwei Huang. 2025a. Auto-scaling continuous memory for gui agent. *arXiv preprint arXiv:2510.09038*.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. 2025b. OS-ATLAS: Foundation action model for generalist GUI agents. In *The Thirteenth International Conference on Learning Representations*.

Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, Yiheng Xu, Junli Wang, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2025. Scaling computer-use grounding via user interface decomposition and synthesis. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

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.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2025. Aguvis: Unified pure vision agents for autonomous GUI interaction. In *Forty-second International Conference on Machine Learning*.

Yan Yang, Dongxu Li, Yutong Dai, Yuhao Yang, Ziyang Luo, Zirui Zhao, Zhiyuan Hu, Junzhe Huang, Amrita Saha, Zeyuan Chen, and 1 others. 2025. Gta1: Gui test-time scaling agent. *arXiv preprint arXiv:2507.05791*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Xianghua Zeng, Hao Peng, Angsheng Li, Chunyang Liu, Lifang He, and Philip S. Yu. 2023. Hierarchical state abstraction based on structural information principles. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI '23.

Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, and 1 others. 2025a. Ufo: A ui-focused agent for windows os interaction. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 597–622.

Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, and 1 others. 2025b. Agent learning via early experience. *arXiv preprint arXiv:2510.08558*.

Di Zhao, Longhui Ma, Siwei Wang, Miao Wang, and Zhao Lv. 2025. Cola: A scalable multi-agent framework for windows ui task automation. *arXiv preprint arXiv:2503.09263*.

Appendices

A System Prompts and Implementation Details

This section details the prompt designs and input-output pairs for the core architectural modules. We present the system instructions followed by specific instantiation examples derived from a representative ServiceNow task: "Sort the hardware list by display name (descending), cost center (ascending), and stockroom (ascending)." This running example

illustrates how the global planner, visual grounder, and state manager collaborate to execute the complex multi-column sorting workflow.

A.1 Global Planner Prompt

The global planner prompt configures the model to act as a centralized reasoning engine. The structure comprises distinct sections that define the role, operational constraints, available tools, and response protocols. The input consists of the multimodal observation history, including the current screenshot and hierarchical state representations. The output is strictly enforced as a structured JSON object containing the thought, tool, and input.

System Prompt: Global Planner

```
You are an expert in GUIs and bash code
executing tasks step-by-step. Always keep the
 task instruction in mind.

# General Instructions
1. **CRITICAL: Do ONLY what the task asks -
nothing more, nothing less**
2. **CRITICAL: Use as LEAST steps as possible
 to complete the task**
3. **CRITICAL: When all required steps are
done, termination IMMEDIATELY**
4. **CRITICAL: ALWAYS review <
execution_history> before deciding next
action:**
   - Check what actions have been done and
   their results
   - Avoid repeating the same action more
   than 3 times
   - Count completed steps to judge task
   completion
5. You receive: screenshot, execution history
, past patterns
6. Never modify user requirements (file names
, paths, etc.)
7. Each action gets automatic evaluation -
you don't need separate verification steps
8. **You can read text directly from
screenshots** - no need for GUI copy/paste
operations. When you read text, record it in
your `thought` field so it appears in
execution history

# Learning from Past Patterns
When provided:
1. **Review lessons carefully** - Pay
attention to common pitfalls and successful
strategies
2. **Apply relevant advice** - Use domain-
specific tips that match the current task
3. **Avoid repeated mistakes** - If past
attempts failed for specific reasons, use
different approaches
4. **Adapt strategies** - Don't blindly copy
past approaches; adapt them to the current
task

# Tools
## gui_action
Execute pyautogui code with optional
placeholders for visual grounding.
Input: PyAutoGUI code string

Use cases:
- **With placeholders**: `pyautogui.click(
X_COORD, Y_COORD)` with description - the
system will locate the element
  - **CRITICAL**: Use X_COORD and Y_COORD
  placeholders when you need to locate GUI
  elements
  - Only ONE placeholder pair per action

- **Without placeholders**: Direct actions
like `pyautogui.write('text')`, `pyautogui.
press('enter')`, `pyautogui.scroll(5)`

**CRITICAL**: For text input operations,
combine click and type in ONE action: `
pyautogui.click(X_COORD, Y_COORD); pyautogui.
write('text')`

**Note**: Don't use pyperclip. Provide a
clear element description when using
placeholders.

## wait
Wait for async operations to complete and
observe UI changes.
Input: Number of seconds to wait (5-30
recommended)

**When to use**: After triggering async
operations (Submit/Apply/Run buttons, page
loads, etc.), use wait to confirm completion
before termination.

## bash_execution
Execute bash commands and Python scripts.
Input: Code string (bash or Python)

### Available Commands
- **Sudo**: `echo {CLIENT_PASSWORD} | sudo -S
 [COMMAND]`
- **Python**: `python3 -c "code"` or `pip
install package && python3 -c "import package
"`
- **Ignore "sudo: /etc/sudoers.d is world
writable" errors**

## infeasible
Declare that the task is objectively
impossible to complete.
Input: Explanation of why the task is
infeasible

**When to use**: After verifying that:
- Software doesn't support the required
feature
- Required files don't exist and can't be
created
- The environment has fundamental limitations
 preventing task completion

**IMPORTANT**: Try alternative approaches
first - only use this if the task is truly
impossible
```

```
# Core Strategy & Workflow
## Incremental Steps
  - Break into small, self-contained steps (
  one snippet per step)
  - Code doesn't persist - write complete,
  standalone snippets
  - Standard workflow:
    1. Install necessary packages if needed
    2. Locate/find target file
    3. THOROUGHLY inspect file contents (
    values, data types, formats)
    4. Modify the file based on findings
    5. Verify changes

## File Modification
  - Modify existing open files IN PLACE (no
  new files unless required)
  - Use appropriate libraries (python-docx,
  openpyxl, pandas)
  - COMPLETE OVERWRITES, not appends (replace
   all content/sheets/paragraphs)
  - Check screenshot for the currently open
  file
  - **CRITICAL FOR EXCEL AND LIBREOFFICE CALC
  **: Prefer bash_execution with Python
  libraries (openpyxl, pandas, xlrd, xlwt)
  for Excel and LibreOffice Calc operations,
  but use gui_action if necessary

## Preserve Structure
  - Never modify headers, titles, sheet names,
   or structural elements unless requested
  - Maintain fonts, colors, borders,
  formatting, styles, and table positioning
  - Only change content/data, not visual
  presentation

# Action Evaluation
After **EVERY** action, you automatically
receive an evaluation comparing before/after
screenshots:
  - Evaluation reports immediate UI response
  to your action
  - Use to detect errors (wrong element
  clicked, unexpected dialogs)

## How to Use Evaluation
  - **CRITICAL**: Evaluation result does not
  mean Task completion
  - Use evaluation to detect obvious errors,
  not to judge task completion
  - Only retry if evaluation shows clear
  errors (error messages, wrong dialogs)
  - **DO NOT** retry just because evaluation
  says "Failed" - may be slow async
  operations

## When to termination
**Judge task completion by counting required
steps, NOT by evaluation results:**
  - Track which steps the task requires and
  which are done
  - When all required steps are executed,
  termination IMMEDIATELY
  - **Exception**: If unsure whether the
  async operation finished, use the wait tool
   first, then termination
  - Ignore "Failed" evaluations if all
  required steps are done
  - **DO NOT** add verification steps unless
  the task explicitly asks

## Error Recovery Strategy
When operations fail:
1. **Analyze error** - Understand root cause
2. **Retry different approach** - Or fix
underlying issue
3. **Use `hint` field** - If the visual
grounder failed, provide specific
instructions to avoid repeating

# Response Format
## Standard Response
```json
{
    "thought": "Brief reasoning about the
    current action. Check prerequisites and
    verify previous result.",
    "tool": "gui_action|bash_execution|wait|
    termination|infeasible",
    "input": "String - tool-specific content
    (see examples below)",
    "description": "Optional - only for
    gui_action with placeholders, describe
    the element to locate"
}
```

Examples:
- gui_action with placeholder: `{"tool": "
gui_action", "input": "pyautogui.click(
X_COORD, Y_COORD)", "description": "Click the
 Submit button"}`
- gui_action without placeholder: `{"tool": "
gui_action", "input": "pyautogui.write('hello
')"}`
- wait: `{"tool": "wait", "input": "15"}`
- bash_execution: `{"tool": "bash_execution",
 "input": "ls -la"}`
- termination: `{"tool": "termination", "
input": "Task completed. [summary]"}`
- infeasible: `{"tool": "infeasible", "input
": "Chrome doesn't support changing search
results per page - this is a search engine
setting, not a browser feature"}`

Termination (Task Complete)
When **all required actions are done and
succeeded**:
```json
{
    "thought": "All task requirements
    completed successfully.",
    "tool": "termination",
    "input": "Task completed. [brief summary
    of what was done]"
}
```

Infeasible (Task Impossible)
When **task is objectively impossible** after
 verification:
```json
{
```
```

```
    "thought": "Verified that [feature/file/
    capability] doesn't exist and cannot be
    created.",
    "tool": "infeasible",
    "input": "Detailed explanation of why the
     task cannot be completed."
}
```

Input: Bounded Context

```
Task: Sort the "hardware" list by the
following fields:
 - Display name (descending)
 - Cost center (ascending)
 - Stockroom (ascending)

<past_pattern>
- Open the filter/funnel panel to access the
"Order results by" controls.
- Configure multi-sort there: 1) Display name
 - Descending, 2) Cost center - Ascending, 3)
 Stockroom - Ascending; then apply.
- Do not use Shift+click on headers to add
secondary/tertiary sorts-it's unreliable.
- For single-field tweaks, use each column
header's menu to set explicit Asc/Desc; it
reliably applies and shows the sort indicator.

- After applying, verify the active sort via
the filter breadcrumb/indicators to confirm
everything is committed.
</past_pattern>

Based on the execution_history and the
current screenshot, decide the next action.
Avoid repeating failed actions. You should
strictly follow the JSON format below:
```json
{
    "thought": "Brief reasoning about the
    current action. Check prerequisites and
    verify previous result.",
    "tool": "gui_action|bash_execution|wait|
    termination|infeasible",
    "input": "String - tool-specific content
    (see examples below)",
    "description": "Optional - only for
    gui_action with placeholders, describe
    the element to locate"
}
```
```

Output: Reasoning and Action

```
{
    "thought": "Open the filter panel to
    configure multi-sort as per pattern.",
    "tool": "gui_action",
    "input": "pyautogui.click(X_COORD,
    Y_COORD)",
    "description": "Click the funnel/filter
    icon on the Hardware list toolbar to open
     the filter panel"
}
```

A.2 Format Correction Prompt

The format correction prompt operates as a repair mechanism triggered by parsing failures. It accepts the specific error message, the malformed raw response, and the required schema definition as inputs. The instruction directs the model to identify the syntax violation and regenerate the response. The output is a valid JSON object that strictly adheres to the format required by the downstream execution controllers.

System Prompt: Format Correction

```
Error: Failed to parse your response.
Error message: {error_message}

Your response was:
{response}

Please provide a valid JSON response in the
exact format:
```json
{{
    "thought": "Brief reasoning (check
    prerequisites, count operations)",
    "tool": "gui_action|bash_execution|wait|
    termination|infeasible",
    "input": "tool input here"
}}
```
```

A.3 Visual Grounder Prompt

The visual grounder prompt directs the model to function as a precise UI element locator. The input includes the current screenshot, the semantic description of the target element, and the dynamic resolution parameters. The output is restricted to a single coordinate pair formatted as (x, y).

System Prompt: Visual Grounder

```
You are an expert UI element locator. Given a
 GUI image and a user's element description,
provide the coordinates of the specified
element as a single (x,y) point. The image
resolution is height {resized_height} and
width {resized_width}. For elements with area
, return the center point.

Output the coordinate pair exactly:
(x,y)
```

Input: Grounding Instruction

```
Click the funnel/filter icon on the Hardware
list toolbar to open the filter panel
```

Output: Coordinate

```
(131, 187)
```

A.4 Step Abstraction Prompt

The step abstraction prompt manages the conversion of raw execution data into abstracted steps. The input comprises the description of the executed action and the visual difference between the pre-action and post-action states. The output consists of a brief textual description of one or two sentences that validates whether the intended state change occurred.

System Prompt: Step Abstraction

```
Compare before/after screenshots and describe
 the UI response in 1-2 sentences:

Action: {action_description}

Be concise:
- Loading/waiting states = action triggered
successfully
- Check cursor position for confirmation
- Only report what changed

Example: "Succeeded. Button clicked, loading
state appeared."
Example: "Succeeded. Cursor at target, no
immediate change."
Example: "Failed. Error dialog: [text]."
```

Input: Action Description

```
Click the funnel/filter icon on the Hardware
list toolbar to open the filter panel
```

Output: Abstracted Step

```
Result: Succeeded. Click registered - cursor
on the funnel icon and the filter panel
expanded (grey filter bar under the toolbar)
with the "Show / hide filter" tooltip visible.
```

A.5 Context Refinement Prompt

The context refinement prompt guides the compression of the execution history. It receives the original task instruction, the previous refined context, and a buffer of recent abstracted steps as inputs. The instruction requires the model to consolidate these inputs by retaining critical strategic milestones while discarding redundant operational details. The output is a formatted summary that updates the global refined context.

System Prompt: Context Refinement

```
Analyze task execution progress and provide
guidance.

Task instruction: {task_instruction}

Execution history (Steps {start_step}~{
end_step}):
{history_text}

Instructions:
- If history contains <previous_summary>,
combine it with <new_steps> to create a
comprehensive summary
- If no <previous_summary>, directly
summarize the provided steps
- List what was done in order (successes and
failures)
- **IMPORTANT**: Preserve coordinates in
click actions (e.g., "click(500,300)") -
these can be reused later
- Identify if we're stuck in loops, making
progress, or blocked
- Provide actionable suggestions for the next
 step if there are issues

Return a concise summary string in this
format:
"Steps {start_step}~{end_step}: [ordered list
 of what was done, keeping coordinates].
Suggestion: [actionable advice, or 'Continue'
 if progressing well]"

Examples:
- "Steps 1~5: Opened file, tried to edit (
failed 3 times with permission error),
attempted sudo (failed). Suggestion: Try a
different approach - copy file to temp
location first."
- "Steps 1~5: Clicked Submit button at click
(850,620), typed text, clicked Save at click
(920,580). Suggestion: Continue - forms being
 filled correctly."
- "Steps 1~10: Previously installed package
and ran script (steps 1~5). Then verified
output, tested functionality (steps 6~10).
Suggestion: Continue - good progress."
- "Steps 1~15: Clicked the same button 5
times with no response, tried alternative
buttons (failed). Suggestion: This approach
isn't working - try an alternative method or
termination as infeasible."
```

Input: Execution History

```
Execution history (Steps 1~5):
Step 1: gui_action | Thought: Open the filter
 panel to configure multi-sort as per pattern.
 | Description: Click the funnel/filter icon
on the Hardware list toolbar to open the
filter panel | Code: pyautogui.click(131,
187) | Result: Succeeded. Click registered -
cursor on the funnel icon and the filter
panel expanded (grey filter bar under the
toolbar) with the "Show / hide filter"
tooltip visible.

Step 2: gui_action | Thought: The filter
panel is already open. Next, I need to add
sort criteria via the Add Sort control per
the pattern. | Description: Click the \'Add
Sort\' button in the filter panel | Code:
pyautogui.click(349, 234) | Result: Succeeded.
 Cursor on the Add Sort button and a new "
```

```
Order results by the following fields" sort
row appeared (Acquisition method = a to z)
with a red remove (X) button.

Step 3: gui_action | Thought: The sort panel
shows one sort row: field = \'Acquisition
method\', order = \'a to z\'. First, change
the field to \'Display name\' as required. |
Description: Open the field dropdown in the
first sort row (currently showing \'
Acquisition method\') and select \'Display
name\' | Code: pyautogui.click(182, 356);
pyautogui.write(\'Display name\'); pyautogui.
press(\'enter\') | Result: Succeeded. The
sort field changed from "Acquisition method"
to "Display name"; cursor is at the dropdown
(hover/click) and no loading/waiting state is
 visible.

Step 4: gui_action | Thought: The filter
panel shows one sort row: Field = \'Display
name\', Order = \'a to z\'. First set it to
descending (\'z to a\'). | Description: Open
the order dropdown (currently \'a to z\') in
the first sort row and select \'z to a\' |
Code: pyautogui.click(376, 356); pyautogui.
write(\'z to a\'); pyautogui.press(\'enter\')
 | Result: Succeeded. The sort operator
changed from "a to z" to "z to a", and the
cursor is positioned over the operator
dropdown.

Step 5: gui_action | Thought: We set the
first sort: Display name - z to a. Now add
the second sort criterion via Add Sort. |
Description: Click the Add Sort button in the
 filter panel | Code: pyautogui.click(349,
234) | Result: Succeeded. Cursor on the Add
Sort button, and a new sort row ("Acquisition
 method - a to z") was added beneath the
existing "Display name - z to a"; no loading/
waiting state shown.
```

Output: Refined Context

```
Steps 1~5:
1) Opened filter panel by clicking funnel (
click(131,187)) - succeeded.
2) Clicked Add Sort (click(349,234)) to add a
 sort row - succeeded.
3) Opened field dropdown (click(182,356)),
typed "Display name" and pressed Enter -
field set to Display name - succeeded.
4) Opened order dropdown (click(376,356)),
typed "z to a" and pressed Enter - order set
to z to a (descending) - succeeded.
5) Clicked Add Sort again (click(349,234)) to
 add a second sort row (shows Acquisition
method - a to z) - succeeded.

Suggestion:
Continue - change the second sort row to "
Cost center" with ascending ("a to z"), then
click Add Sort (click(349,234)) once more to
add a third row and set it to "Stockroom"
ascending. Use the same field and order
dropdown controls you used before (field
dropdown at the row\'s field control -
previously click(182,356) - and order control
 at the row\'s order dropdown - previously
click(376,356)) to make those selections.
```

A.6 Pattern Induction Prompt

The pattern induction prompt orchestrates the extraction of generalizable patterns from completed episodes. The input consists of the initial task instruction and the full sequence of abstracted steps. The output is structured as a JSON list containing verified insights categorized as success or failure patterns to be stored in the vector database.

System Prompt: Pattern Induction

```
Analyze this task execution and extract ONLY
the most important, reusable lessons.

Task: {task_instruction}

Execution history:
{step_abstracts}

Extract ONLY verified lessons (maximum 3)
that would help with similar tasks.

IMPORTANT Guidelines:
- **Data leakage prevention**: You do NOT
know final success/failure - focus on
execution process only
- **Only VERIFIED lessons**: If stuck on the
same step for multiple attempts, record as
failed approach (e.g., "DON'T use X for Y")
- **Clear evidence required**: Only include
what clearly worked or clearly failed after
attempts
- **No speculation**: Omit uncertain/
unverified observations - if unsure, don't
include
- **Focus on**: Failed methods (tried
multiple times), successful strategies,
critical pitfalls
- **Avoid**: Vague suggestions, unverified
hypotheses, trivial details
- **Generalize lessons**: Do NOT include
specific values (text content, file names,
field values, etc.) - describe patterns and
methods instead
- Each lesson must be specific and actionable
- Return an empty list if no significant
verified lessons

Format as a JSON list of objects with type
and lesson (maximum 3 items):
[
  {{"type": "success", "lesson": "Method X
  worked: ..."}},
  {{"type": "failure", "lesson": "DON'T use
  method Y: tried 3 times, doesn't work"}}
]

Type values (ONLY these two):
- "success": A method/strategy that clearly
worked during execution
```

```
- "failure": A method/strategy that clearly
failed after multiple attempts
```

Input: All Abstracted Steps

```
Step 1: gui_action | Thought: Open the filter
 panel to configure multi-sort as per pattern.
 | Description: Click the funnel/filter icon
on the Hardware list toolbar to open the
filter panel | Code: pyautogui.click(131,
187) | Result: Succeeded. Click registered -
cursor on the funnel icon and the filter
panel expanded (grey filter bar under the
toolbar) with the "Show / hide filter"
tooltip visible.

Step 2: gui_action | Thought: The filter
panel is already open. Next, I need to add
sort criteria via the Add Sort control per
the pattern. | Description: Click the \'Add
Sort\' button in the filter panel | Code:
pyautogui.click(349, 234) | Result: Succeeded.
 Cursor on the Add Sort button and a new "
Order results by the following fields" sort
row appeared (Acquisition method = a to z)
with a red remove (X) button.

Step 3: gui_action | Thought: The sort panel
shows one sort row: field = \'Acquisition
method\', order = \'a to z\'. First, change
the field to \'Display name\' as required. |
Description: Open the field dropdown in the
first sort row (currently showing \'
Acquisition method\') and select \'Display
name\' | Code: pyautogui.click(182, 356);
pyautogui.write(\'Display name\'); pyautogui.
press(\'enter\') | Result: Succeeded. The
sort field changed from "Acquisition method"
to "Display name"; cursor is at the dropdown
(hover/click) and no loading/waiting state is
 visible.

Step 4: gui_action | Thought: The filter
panel shows one sort row: Field = \'Display
name\', Order = \'a to z\'. First set it to
descending (\'z to a\'). | Description: Open
the order dropdown (currently \'a to z\') in
the first sort row and select \'z to a\' |
Code: pyautogui.click(376, 356); pyautogui.
write(\'z to a\'); pyautogui.press(\'enter\')
 | Result: Succeeded. The sort operator
changed from "a to z" to "z to a", and the
cursor is positioned over the operator
dropdown.

Step 5: gui_action | Thought: We set the
first sort: Display name - z to a. Now add
the second sort criterion via Add Sort. |
Description: Click the Add Sort button in the
 filter panel | Code: pyautogui.click(349,
234) | Result: Succeeded. Cursor on the Add
Sort button, and a new sort row ("Acquisition
 method - a to z") was added beneath the
existing "Display name - z to a"; no loading/
waiting state shown.

(We present the initial five steps for
illustrative purposes)
```

Output: Induced Patterns

```
[
    {
        "type": "success",
        "lesson": "Selecting dropdown values
        by typing the desired option and
        pressing Enter worked consistently for
         both field selection and sort
        direction, avoiding scrolling and
        speeding configuration."
    },
    {
        "type": "success",
        "lesson": "Constructing multi-criteria
         sorting by repeatedly adding sort
        rows worked: add a row, set its field
        and direction, then add the next until
         all criteria are configured."
    },
    {
        "type": "success",
        "lesson": "Applying the configured
        sorts via the panel's run/apply
        control successfully submitted the
        changes, as indicated by a loading
        state."
    }
]
```

A.7 Pattern Synthesis Prompt

The pattern synthesis prompt facilitates the transfer of retrieved patterns to the current context. It takes the current task instruction and a set of retrieved induced patterns as inputs. The output is a concise list of strategic advice that is injected into the context to prime the planning process.

System Prompt: Pattern Synthesis

```
Given the current task and past lessons from
the same domain, provide a concise, refined
summary of actionable advice.

Current task: {current_task}

Past lessons:
{pattern_summary}

IMPORTANT: Items marked as "REQUIREMENTS (
MUST FOLLOW)" are mandatory rules that MUST
be followed.

Your task:
1. **Filter** - Select ONLY the most relevant
 lessons for this specific task
2. **Synthesize** - Combine similar lessons
into unified advice
3. **Refine** - Express advice concisely and
actionably (5 bullet points maximum)
4. **Prioritize** - Focus on: mandatory
requirements first, then critical pitfalls,
then helpful strategies
5. **Conflict Resolution** - If success/fail
lessons conflict with required lessons,
prioritize and follow the required lessons.
```

```
Return empty string if no relevant lessons
exist.
```

Input: Domain Relevant Patterns

```
--- SUCCESS patterns ---
[234] Use the column header menu to sort:
opening the header's menu and choosing
explicit ascending/descending reliably
applied the sort and showed the indicator (
confirmed on multiple columns). (similarity:
0.66)

[252] Open the filter/sort controls by
clicking the filter/funnel control - this
reliably reveals the filter and 'Order
results by' section. (similarity: 0.65)

[237] Use the catalog's built-in search input
 to locate the target item, then open its
result entry to access configuration; this
reliably surfaced the correct item without
manual browsing. (similarity: 0.57)

[151] Confirm applied criteria via the filter
 breadcrumb after running; it can reveal
uncommitted selections even when the builder
displays a value. (similarity: 0.51)

--- FAILURE patterns ---
[235] DON'T rely on Shift+click to add
secondary/tertiary sorts: tried many times
and it rarely produced a sort indicator or
visible change. (similarity: 0.53)
```

Output: Synthesized Patterns

```
- Open the filter/funnel panel to access the
"Order results by" controls.
- Configure multi-sort there: 1) Display name
 - Descending, 2) Cost center - Ascending, 3)
 Stockroom - Ascending; then apply.
- Do not use Shift+click on headers to add
secondary/tertiary sorts-it's unreliable.
- For single-field tweaks, use each column
header's menu to set explicit Asc/Desc; it
reliably applies and shows the sort indicator.

- After applying, verify the active sort via
the filter breadcrumb/indicators to confirm
everything committed.
```

B Baseline Implementation Details

Regarding specific implementations, we enforce a strict hard-stop mechanism for CoAct-1 to terminate execution immediately upon reaching the step budget limit. This modification overrides the default protocol, which permits subtask completion beyond the step limit, thereby ensuring that all evaluated frameworks operate under identical and rigorous step constraints. For Agent S3, we employ a hybrid memory mechanism pairing an 8-frame visual sliding window with full textual history. For GTA1, we utilize a step-wise sampling parameter of 5 to generate candidate actions during reasoning.

C Computational Resources

C.1 API Specifications

We utilize the official API provided by OpenAI. Table 7 details the specific model snapshots and their corresponding pricing.

Model Identifier	Cost ($/M)	
	Input	Output
GPT-5-2025-08-07	1.25	10.00
GPT-5-mini-2025-08-07	0.25	2.00
Computer-Use-Preview-2025-03-11	3.00	12.00

Table 7: List of API model identifiers and corresponding pricing used in experiments. Prices are denoted in USD per million tokens.

C.2 Hardware Environment

We utilize a Dell Pro Max Tower T2 desktop model FCT2250 to host local inference for open-source models, including UI-TARS-1.5-7B and GTA1-7B. This workstation relies on an Intel Core Ultra 7 265 processor and features a high-performance NVIDIA RTX Pro 6000 Blackwell workstation edition GPU with 96GB of VRAM. To conduct the benchmarks, we employ two standard Windows laptops to host the execution environments within VMware virtual machines configured in single-environment mode.

C.3 Assets and Licenses

We evaluate our framework using the Spider2-V and OSWorld benchmarks. We utilize these assets consistently with their intended usage for research evaluation and adhere to their respective Apache-2.0 licenses. Our implementation of the HiSA framework is open-sourced under the MIT license.

D Domain-Specific Performance Analysis

Table 8 details the performance and efficiency metrics across various application domains within the Spider2-V abstract subset. The results demonstrate that the resource consumption varies significantly depending on the interaction modality required by each software environment. For instance, the Dagster domain exhibits the lowest average monetary

Domain	SR (%) ↑	Cost ($) ↓	Image Count ↓	Tokens (K) ↓			Steps ↓		
				Total	Input	Output	Total	GUI	Bash
Excel	45.45	0.40	36.00	120.41	80.33	40.07	9.75	8.50	1.25
Snowflake	40.00	0.81	102.42	302.72	222.61	80.12	26.50	24.42	2.08
Dagster	43.33	0.25	31.63	123.92	103.34	20.59	9.31	4.63	4.69
Airflow	87.50	0.71	89.25	299.29	230.57	68.72	23.50	17.00	6.50
Airbyte	65.00	0.39	62.50	202.72	167.31	35.41	15.95	13.80	2.15
Superset	85.71	0.56	67.25	229.11	175.98	53.13	18.75	13.75	5.00
Metabase	52.38	0.46	67.71	193.82	148.72	45.10	17.36	16.36	1.00
ServiceNow	58.00	0.20	37.33	93.92	74.46	19.47	9.50	9.50	0.00
Jupyter	50.00	0.39	49.60	143.37	105.77	37.61	13.10	12.10	1.00
Overall	**40.58**	**0.44**	**58.28**	**183.05**	**141.30**	**41.75**	**15.34**	**13.01**	**2.33**

Table 8: Performance and efficiency metrics across different application domains on Spider2-V abstract subset. All metrics are averaged per task.

cost of $0.25, which correlates with a balanced distribution between GUI and bash steps where the agent effectively leverages command-line operations to minimize expensive visual processing. Conversely, the Snowflake domain incurs the highest token consumption and monetary costs, driven by a significantly larger volume of image inputs and extended trajectory lengths necessary for complex data tasks. Despite these variations in task complexity and duration, the framework maintains a consistent success rate across diverse platforms by dynamically adapting its action space and context management strategies to the specific constraints of each application.

E Qualitative Analysis of Pattern Induction

We present qualitative case studies to demonstrate the effectiveness of pattern induction. These examples illustrate how retrieved patterns serve as explicit operational guidelines, providing the agent with critical domain knowledge to guide its decision-making and facilitate accurate task execution.

Case 1: Excel Chart Creation

```
Instruction:
Work out the monthly total sales in a new row
 called "Total" and then create a line chart
to show the calculated results. The x-axis
should be Months. I want a Lines Only chart
with "Total" as the title. Do not display
Legend and leave other settings untouched.

Retrieved Pattern:
- Mandatory: If any pivot is involved perform
 it via GUI only. If you create a new sheet
at any point name it Sheet2.
- Do not use a pivot for this. Add a new "
Total" row under the month columns and
compute monthly totals with SUM.
- Enter the formula in the first month's
Total cell then drag fill across all month
columns.
- Insert the chart via the Insert Chart
button. Choose Line -> Lines Only.

Analysis:
The baseline model failed by attempting to
aggregate data using a pivot table. This
action impermissibly altered the sheet
structure. The retrieved pattern acted as a
strict constraint mechanism and explicitly
forbade the use of pivot tables for this
specific summation task. It further provided
the exact navigation path to the chart sub-
type.
```

Case 2: Airflow DAG Triggering

```
Instruction:
I am using Astronomer to deploy Airflow. I
have just implemented several test functions.
 Please check these logs and adjust my DAG
file accordingly then trigger the dag.

Retrieved Pattern:
- Open the DAG detail page and switch to the
Graph tab.
- Trigger via the header Trigger/Play button
as this path is the most reliable.
- The Graph-view trigger will auto-unpause a
paused DAG. Confirm the UI shows the trigger
and status change succeeded.

Analysis:
Triggering a DAG from the main list view in
the Airflow interface is often unreliable due
 to state update latencies. The pattern
provided a robust navigation heuristic which
directed the agent to switch to the Graph tab.
 This interaction path ensured the workflow
execution was confirmed even if the DAG was
initially paused.
```

Case 3: Airbyte Data Synchronization

Instruction:
I hope to synchronize data regularly from a MySQL database to a local json file "/json_data". Set the schedule to every 12 hours. The detailed configuration is provided in the opened file on desktop.

Retrieved Pattern:
- If API-based scheduling or triggering does not return the expected connection then open the connection page in the UI and start the sync there to reliably initiate it.
- After setting the 12-hour schedule run a manual sync from the connection page once to validate the configuration immediately.

Analysis:
The baseline model typically concluded the task after setting the schedule and assumed the system would handle the subsequent execution. This behavior often left connections in a pending state. The pattern introduced a mandatory validation step that forced the agent to perform an active check before marking the task as complete.

Case 4: Airbyte CDC Configuration

Instruction:
Please help me synchronize data. Read changes directly from a transaction log using CDC and capture every change from a source database.

Retrieved Pattern:
- Complete the upstream ingestion sync then use force refresh table list and search again.

- After refresh confirm the new tables appear and select them for CDC replication before starting the sync.

Analysis:
A common failure in Airbyte CDC tasks occurs when the UI cache does not display newly added source tables. The baseline model abandoned the task when tables were not found. The pattern anticipated this system state and provided the specific remediation to force refresh the table list. This allowed the agent to actively correct the UI state.

Case 5: Airbyte Scheduling Syntax

Instruction:
I have established a connection from Faker to local .csv file. I hope it can be replicated at 18:00 pm every day.

Retrieved Pattern:
- In schedule settings set it to run daily at 18:00. If cron is used enter 0 18 * * *.
- Run once manually now to validate path and permissions before relying on the schedule.

Analysis:
Constructing Cron expressions is error-prone for language models. The baseline model frequently generated incorrect syntax. The pattern provided the exact string literal and reiterated the importance of a manual test run. This intervention eliminated syntax-related errors.

Case 6: ServiceNow List Sorting

Instruction:
Sort the "hardware" list by the following fields: Display name (descending), Cost center (ascending), and Stockroom (ascending).

Retrieved Pattern:
- In the list filter/sort builder use the Sort (A/Z) control to add three sort rows in this order.
- Do not try to turn a filter row into a sort via an "Order by" option. It will not create a proper sort row or apply sorting.

Analysis:
The baseline model attempted to sort by clicking column headers sequentially. This action overwrites previous sorts rather than stacking them. The pattern explicitly warned against this approach and directed the agent to use the specialized sort control. This enabled the correct application of complex multi-level sorting.

Case 7: Jupyter Notebook Execution

Instruction:
Help me build a Decision Tree model. Add codes only to the existing cells and in the end run all the cells and save the jupyter notebook.

Retrieved Pattern:
- Check cell types before execution. If a cell containing code is marked as Raw convert it to Code format.
- Ensure all cells are executed in sequence and check for execution numbers to confirm they ran.

Analysis:
The environment contained a trap where the training data cell was set to a raw format. The baseline model ignored this setting and failed when running the notebook. The pattern provided critical environmental awareness and prompted the agent to inspect and change the cell type.

Case 8: ServiceNow User Creation

Instruction:
Create a new user with a value of "Christophel" for field "Last name".

Retrieved Pattern:

- Use the catalog page search field to quickly filter to the "Users" item. Open it from the results to configure and create the new user with the provided fields.

Analysis:
The menu structure in ServiceNow is deep and complex. The baseline model often failed to navigate the menu hierarchy. The pattern recommended a search-based strategy which bypassed the menu hierarchy entirely. This shortcut reliably led the agent to the standard user creation form and minimized interaction steps.