CUA-Skill: Developing Computer Using Agents with a Skill Framework

Tianyi Chen [*1] **Yinheng Li** [*1] **Michael Solodko** [*1] **Sen Wang** [*1]
Nan Jiang [1] **Tingyuan Cui** [1] **Junheng Hao** [1] **Jongwoo Ko** [1] **Sara Abdali** [1] **Qing Xiao** [1]
Justin Wagle [1] **Pashmina Cameron** [1] **Suzhen Zheng** [1] **Leon Xu** [1] **Hao Fan** [1] **Kazuhito Koishida** [1]

Abstract

Computer-Using Agents (CUAs) aim to autonomously operate computer systems to complete real-world tasks. However, existing agentic systems remain difficult to scale and lag behind human performance. A key limitation is the absence of reusable and structured skill abstractions that capture how humans interact with graphical user interfaces and how to leverage these skills. We introduce CUA-Skill, a computer-using agentic skill base that encodes human computer-use knowledge as skills coupled with parameterized execution and composition graphs. CUA-Skill is a large-scale library of carefully engineered skills spanning common Windows applications, serving as a practical infrastructure and tool substrate for scalable, reliable agent development. Built upon this skill base, we construct CUA-Skill Agent, an end-to-end computer-using agent that supports dynamic skill retrieval, argument instantiation, and memory-aware failure recovery. Empirically, CUA-Skill substantially improves the quality and reliability of trajectory generation, achieving a 76.4% success rate, which is multiple times higher than existing baselines. On the challenging end-to-end WindowsAgentArena benchmark, CUA-Skill Agent further attains state-of-the-art performance with a 57.5% best-of-three success rate, while remaining significantly more efficient than prior and concurrent approaches. Together, CUA-Skill serves as a strong and scalable foundation for building future CUA systems.

1. Introduction

Computer-Using Agents (CUAs) aim to autonomously operate graphical user interfaces (GUIs) to complete real-world desktop tasks such as document editing, web navigation, data analysis, and system configuration (Xie et al., 2024; Zhang et al., 2025c; Yang et al., 2025b; Hui et al., 2025). Recent advances in large language models (LLMs) and multimodal perception have substantially improved agents' abilities to interpret user intent and visually ground actions on the screen, making CUAs a promising pathway toward general-purpose digital assistants capable of interacting with complex desktop environments.

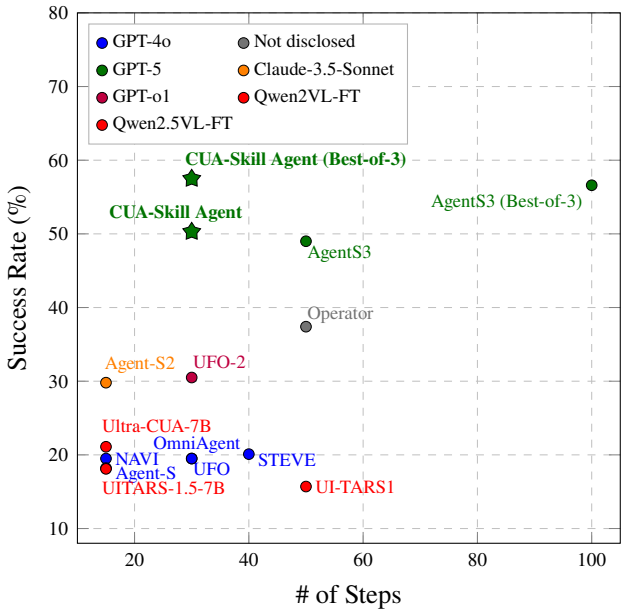


Figure 1. Success rate vs. execution steps on WAA.

Despite this progress, building reliable and scalable CUAs remains challenging. Existing systems often struggle with long-horizon tasks that require executing dozens of interdependent actions across dynamic UI states. Small errors in grounding, planning, or execution can quickly compound, leading to brittle behavior and low end-to-end success rates. More fundamentally, most current approaches lack an explicit representation of *how humans use computers*: desktop interaction is typically modeled as flat sequences of low-level actions, forcing agents to repeatedly rediscover common workflows from scratch.

In contrast to current CUAs, human computer use is in-

[*]Equal contribution [1]Microsoft. Correspondence to: Yinheng Li <yinhengli@microsoft.com>.

Accepted to the 2nd Workshop on Compositional Learning at ICML 2026, Seoul, South Korea. *Copyright 2026 by the author(s).*

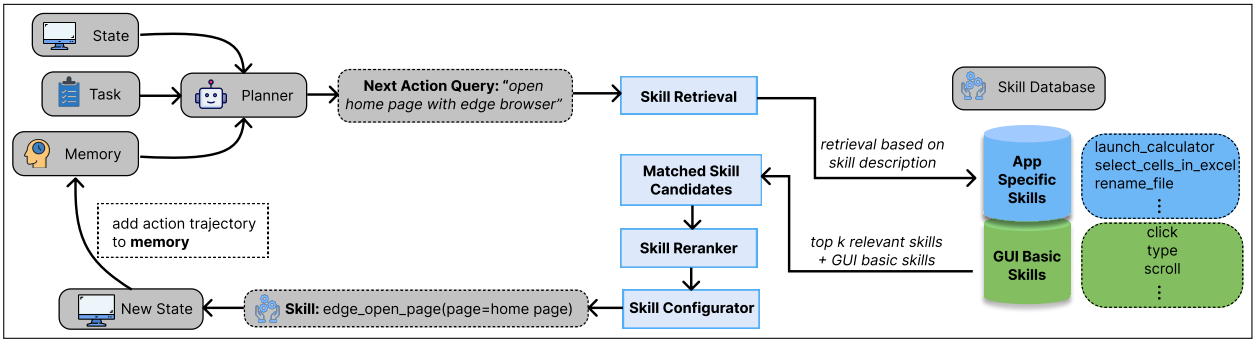


Figure 2. Overview of CUA-Skill and Associated Skill-Agent.

herently structured around reusable procedural knowledge. Users rely on familiar *skills*, such as launching applications, navigating menus, or formatting documents, which are composed into higher-level workflows and adapted to the current UI context. The lack of such reusable and structured skill abstractions remains a key bottleneck for existing CUAs, limiting their scalability, generalization, and robustness on complex real-world tasks.

Concurrently, Anthropic introduced the notion of "agent skills" as reusable, filesystem-based resources that encapsulate domain expertise (Notov, 2025). While effective in code-centric environments (e.g., Linux or API-rich systems), these skills are primarily executed through scripts and tightly integrated with the Model Context Protocol (MCP) (Anthropic, 2024a). As a result, they are less suited for desktop environments such as Windows, where many applications expose limited or inconsistent programmatic APIs and effective task execution fundamentally makes them difficult to leverage across applications. Therefore, a question is naturally raised:

How can we build a scalable and transferable skill base for desktop environments that captures human procedural knowledge and enables reliable and capable CUAs?

In this work, we answer this question by introducing CUA-Skill, the first systematic agentic skill library designed for desktop computer use. CUA-Skill encodes human computer-use knowledge as reusable *skills* coupled with **parameterized execution and composition graphs**, forming a structured intermediate layer between high-level user intent and low-level interaction primitives. While GUI primitives serve as the primary, human-aligned substrate for skill execution, the execution graph abstraction flexibly supports script- and code-based execution paths when they offer improved reliability or efficiency. This unified parameterization makes skills transferable across tasks, UI states, and applications, enabling strong generalization.

Built on top of this skill base, we develop **CUA-Skill Agent**, an end-to-end computer-using agent that performs retrieval-augmented skill selection, configuration, and execution. At each step, an LLM-based planner retrieves relevant skills conditioned on the current UI state and user goal, re-ranks candidates using execution context and memory, instantiates skill arguments, and executes the selected skill via GUI grounding or direct script execution, depending on the instantiated execution path. This design supports scalable skill expansion, memory-aware recovery from failures, and robust long-horizon task completion without hard-coding tools into prompts or relying on monolithic plans (Huang et al., 2023; Schick et al., 2023).

Our main contributions are summarized as follows:

- **CUA-Skill.** We introduce a structured agentic skill library for desktop environments that encodes human computer-use knowledge as reusable, parameterized skills with explicit execution and composition graphs. This design enables strong transferability and generalization across tasks and UI states. The initial release contains hundreds of carefully engineered atomic skills spanning tens of popular applications. Through parameterization and composition, these skills can be instantiated into millions or more executable task variants, supporting a wide range of downstream agent applications.

- **CUA-Skill Agent.** To effectively utilize CUA-Skill, we propose a skill-centric, retrieval-augmented agent that performs dynamic skill retrieval, argument instantiation, and execution. The agent supports scalable skill expansion, memory-aware failure recovery, and robust long-horizon desktop task completion.

- **State-of-the-Art Performance.** Extensive evaluations demonstrate that CUA-Skill substantially improves the performance of multiple agent applications. In the trajectory generation, CUA-Skill achieves a **76.4%** success rate, outperforming existing approaches by $\mathbf{1.7} \times -\mathbf{3.6}\times$. On the end-to-end WindowsAgentArena CUA benchmark, CUA-Skill Agent attains state-of-the-art results, achieving a best-of-three success rate of **57.5%**.

2. Related Works

In this section, we discuss two related topics to CUA-Skill. More related works are provided in Appendix A.

Memory Modules and Knowledge Graph Integration. Memory modules are a core architectural component in many recent computer-use agents (CUAs), enabling agents to track task progress and reuse historical information across long-horizon interactions (Agashe et al., 2025b; Song et al., 2025; Wang et al., 2025a). By retaining execution history and intermediate outcomes, memory supports more informed planning decisions and mitigates the limitations of a single prompt window (Park et al., 2023). Recent systems further structure memory as explicit graphs with retrieval and update operations, allowing agents to store intermediate facts, revise beliefs, and incorporate new evidence over time (Chhikara et al., 2025). In parallel, recent frameworks emphasize iterative query, update interactions with knowledge graphs to improve reasoning consistency and reduce hallucination, with demonstrated benefits in mobile-agent settings (Guan et al., 2025).

Relation to CUA-Skill. These approaches primarily focus on modeling *what the agent knows*, i.e., task state, observations, and historical outcomes, but do not explicitly encode *how humans perform computer interactions* as reusable procedures. In particular, they lack action-level abstractions with parameterized execution semantics, limiting the systematic reuse of interaction knowledge across tasks, applications, and UI contexts. In contrast, CUA-Skill targets this missing procedural layer by encoding human computer-use behavior as reusable skills with **parameterized execution graphs, enabling transferable and reliable procedural knowledge for desktop environments**.

Structured Task Planning and MCPs. Recent work on tool-using agents increasingly frames computer use as a structured planning problem, where success depends on coordinating actions over long horizons rather than selecting isolated tool calls (Zhuang et al., 2023; Chen et al., 2025). Code-based planning further strengthens such compositions by compiling high-level intents into modular units that can be reused across tasks (Singh et al., 2022). Desktop-agent foundations emphasize that explicit workflow structures become increasingly important as task horizons grow, as they support state tracking and recovery under tool and UI ambiguity (Wang et al., 2025c).

In parallel, recent standardization efforts such as the Model Context Protocol (MCP) focus on unifying *how* agents access external software, tools, and data sources through standardized client–server interfaces (Anthropic, 2024b; Model Context Protocol Contributors, 2025). While MCP and related tool abstractions provide a powerful foundation for connecting agents to software systems, they typically require substantial engineering effort to expose application-specific APIs and maintain underlying codebases, especially for complex desktop environments.

Relation to CUA-Skill. CUA-Skill extends both structured planning frameworks and MCP-style tool interfaces, but introduces a new abstraction level to fulfill the gaps. Instead of requiring deep software integration or tool implementations, CUA-Skill encodes human computer-use knowledge as reusable skills with parameterized execution and composition graphs. CUA-Skill significantly lowers the engineering burden for skill authors and agent developers. **This design makes CUA-Skill user-friendly to construct, easier to extend and maintain, and naturally reusable across applications and tasks.** As a result, CUA-Skill provides a practical and scalable substrate for robust desktop CUA.

3. Computer-Using Agentic Skills

This section introduces **CUA-Skill**, a structured and parameterized skill abstraction system designed to encode human computer-use knowledge for desktop environments. Our core premise is that effective computer use is not a flat sequence of primitive GUI actions, but a composition of reusable, intent-aligned skills that humans routinely apply across tasks and applications, each admitting multiple valid realizations under different states.

Formally, CUA-Skill consists of three components: (*i*) a *skill cell* that captures minimal user intent, (*ii*) a *parameterized execution graph* that specifies concrete realizations of the skill through the ways like GUI-grounded interactions and executable scripts, and (*iii*) a *composition graph* that encodes how individual skills are typically chained together.

3.1. Skill

Skill is the primitive behavioral units in CUA-Skill. Each skill is denoted by S and captures a minimal but meaningful user intent. The collection of skills is denoted as $\mathcal{S}$.

$$S := \{\tau, \mathcal{I}, \mathcal{A}, \mathcal{G}_e\}. \tag{1}$$

A skill S is defined by, *(i)* a suitable application τ, *(ii)* a natural language user intent $\mathcal{I}$, *(iii)* an argument pool $\mathcal{A}$, and *(iv)* a parameterized execution graph $\mathcal{G}_e$. The argument schema $\mathcal{A} = \{A_1, \cdots, A_K\}$ specifies a set of type slots that describe the information that the skill needs from the user or the environment. The execution graph $\mathcal{G}_e$ encodes how to realize the intent as a sequence of low-level interactions, such as keystrokes, mouse events, or application-specific API calls, conditional on those arguments. By constraining skills to be small and application-specific, CUA-Skill can reliably reuse them as building blocks when constructing longer multi-step workflows across applications.

Feasible Domain and Generator for Argument For each argument $A \in \mathcal{A}$, we associate a *feasible domain* $\mathcal{D}(A)$ that specifies the set of values for which the skill remains well-defined and executable. These domains are defined as part of the skill specification and reflect both application semantics and desktop environment constraints.

We distinguish between two broad categories of argument domains. **Finite-Domain Arguments** correspond to discrete and enumerable choices, such as menu items, toolbar options, system toggles, or predefined configuration states. For such arguments, $\mathcal{D}(A)$ is a finite set that can be exhaustively enumerated or dynamically queried from the UI state. In contrast, **Open-Domain Arguments** correspond to unbounded or high-cardinality inputs, such as file paths, textual content, and numerical values, etc. These domains are typically infinite or impractically large to enumerate and require structured sampling strategies.

The feasible domain definition enables CUA-Skill to reason about argument validity independently of execution, ensuring that each instantiated skill corresponds to a realizable interaction on the desktop. Moreover, feasible domains allow us to associate each argument type with a specialized argument generator, tailored to the structure of $\mathcal{D}(A)$. For example, finite-domain arguments may be instantiated via enumeration or UI-state grounding, while open-domain arguments may be generated through controlled sampling, or environment-aware heuristics.

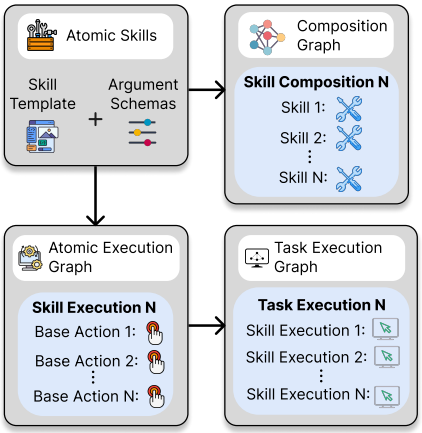


Figure 3. **CUA Skill and Graph Construction.**

3.2. Skill Execution Graph

For each skill $S \in \mathcal{S}$, we construct a *skill execution graph* $\mathcal{G}_e(S) = (\mathcal{V}, \mathcal{E})$ that provides one or more concrete procedures for realizing the user intent. Each execution graph may comprise GUI-grounded interaction primitives or executable script actions, unified within a single representation. Unlike a fixed action sequence, the execution graph encodes a parameterized structured space of valid interaction paths that account for common UI variations, alternative execution realizations, and execution contingencies.

Each node $v \in \mathcal{V}$ represents an internal control state of the skill, including a designated start state and one or more terminal states. Each directed edge $(v, a, v') \in \mathcal{E}$ is labeled by a *base action* a, which may correspond to a GUI interaction primitive or an executable script action, and may be guarded by UI predicates that condition execution on the current screen state. The execution graph is parameterized by a concrete argument instantiation from $\mathcal{D}(A_1) \times \cdots \times \mathcal{D}(A_K)$, which determines concrete interaction targets, such as UI elements, file paths, or input content. Concrete examples are present in Appendix E.

In practice, most execution graphs are compact directed graphs with a dominant execution path and a small number of guarded branches. These branches handle common UI variants, such as alternative menu layouts, dialog prompts, or multiple valid interaction affordances, enabling skills to remain robust to UI changes without requiring redefinition. Moreover, the execution graph supports edge weighting mechanisms that encode execution preferences for different downstream use cases or human preferences.

3.3. Skill Composition Graph

The skill composition graph is a directed graph $\mathcal{G}_c = (\mathcal{V}_c, \mathcal{E}_c)$ that encodes how individual skills can be composed into higher-level user tasks. Each node $v \in \mathcal{V}_c$ corresponds to a skill S_v, and each directed edge $(u, v) \in \mathcal{E}_c$ represents a valid composition from skill S_u to skill S_v.

A path $(v_1, \ldots, v_T)$ in $\mathcal{G}_c$ defines a multi-step task workflow, where nodes represent intermediate sub-goals and edges encode ordering and compatibility constraints between skills. Importantly, the skill composition graph captures reusable procedural knowledge about how skills are typically chained in human computer use, rather than prescribing a fixed execution plan.

We organize $\mathcal{G}_c$ into single-application and cross-application scenarios. Edge transitions may connect skills within the same application or across different applications. This unified representation allows CUA-Skill to model both single-application and multi-application workflows within a shared compositional structure.

4. CUA-Skill Agent

We now design a **CUA-Skill Agent** that supports flexible, long-horizon task completion via dynamic skill selection and execution. Given a natural-language user instruction, the agent incrementally selects, configures, and executes skills from the CUA-Skill library, conditioning each decision on the current UI state, execution history, and ac-

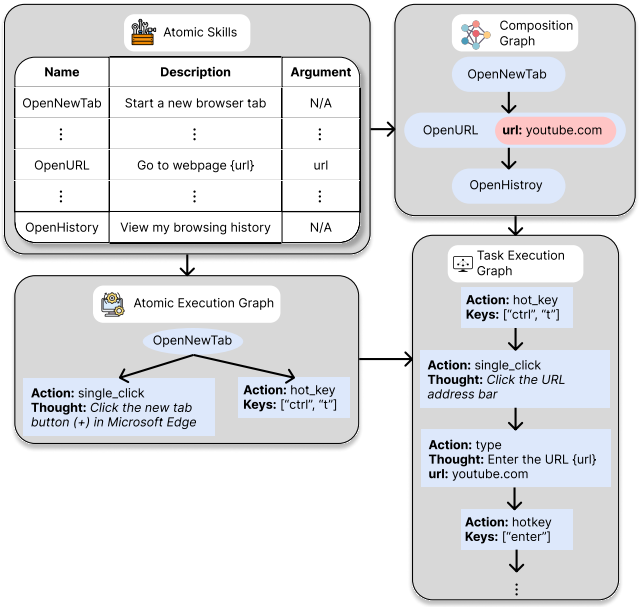


Figure 4. **CUA Skill and Graph Construction Example.**

cumulated memory. At each step, an LLM planner $\mathcal{M}_p$ determines both *which* skill to invoke next and *how* to instantiate its arguments. This design enables adaptive task completion under UI variability and execution uncertainty.

The overall architecture of CUA-Skill Agent is depicted in Figure 2 and stated in Algorithm 1. It consists of five core components: *(i)* a retrieve-augmented skill planner, *(ii)* a skill re-ranker module, *(iii)* a skill argument configuration module, *(iv)* a memory module for to store past action trajectory and execution feedback, and *(v)* an executor.

4.1. Retrieve-Augmented Skill Planner

The planner of CUA-Skill Agent is a *Retrieve-Augmented Skill Planner*. Similar while *different* to the tool invoker in MCP (Anthropic, 2024a), it uses an LLM (e.g., GPT-5) to select an appropriate skill conditioned on the current screen state, execution history, and user goal. Rather than exposing the full skill library to the model context, the planner operates over a structured retrieve-augmented pipeline that narrows the skill space before decision making. The planner $\mathcal{M}_p$ participates throughout the planning process, including skill selection and argument configuration, enabling coherent reasoning across all planning stages.

4.2. Skill Selection

Query Generator. The Query Generator leverages the planner $\mathcal{M}_p$ to produce candidate retrieval queries for skills that can advance the user goal. A central challenge is that the LLM has no prior knowledge of the available skill inventory. Although fine-tuning the model with the full skill list is possible, doing so would require retraining whenever new skills are introduced. Instead, we rely entirely on test-time LLM capabilities and assume that skill names and descriptions follow common natural-language conventions. Under this assumption, the LLM can generate sufficiently general queries that match relevant skills via retrieval.

For better selection, we employ two mechanisms: ensembled query generation, where multiple queries with varying wording granularity are generated to cover semantic interpretations (Appendix B); skill reranker that re-evaluates skill candidates and selects the most appropriate skill.

Skill Retrieval. We adopt a hybrid retrieval strategy that combines lexical matching with semantic retrieval, as such hybrids have shown strong performance in text retrieval tasks (Thakur et al., 2021). In contrast to many MCP architectures that expose the entire tool set directly to model context, CUA-Skill Agent retrieves only a small set of relevant skills, improving scalability and inference efficiency.

Algorithm 1 CUA-Skill Agent

Input: User instruction $\mathcal{U}$, planner $\mathcal{M}_p$, retrieval module $\mathcal{R}$ over skill collection $\mathcal{S}$, basic skill subset $\mathcal{S}_{\text{basic}} \subseteq \mathcal{S}$, memory $\mathcal{M}$, environment $\mathbb{E}$.
Hyperparameters: query budget K, skill budget L.
Initialize memory $\mathcal{M} \leftarrow \{\mathcal{U}\}$ and timestamp $t \leftarrow 0$.
while termination condition is not satisfied **do**
  Obtain observation $o_t \leftarrow \text{GetObservation}(\mathbb{E})$.
  Query generation: LLM produces K queries

$$Q_t \leftarrow \text{QueryGenerator}(\mathcal{M}_p, \mathcal{U}, o_t, \mathcal{M}, K).$$

  Skill retrieval: retrieve top-L candidates from $\mathcal{S}$.

$$C_t \leftarrow \text{RetrieveTopLQuery}(\mathcal{R}, Q_t, S, T).$$

  Skill re-ranking: pick the most promising skill, considering both retrieved and basic skills

$$S_t \leftarrow \text{SkillReranker}(\mathcal{M}_p, C_t \cup \mathcal{S}_{\text{basic}}, o_t, \mathcal{M}).$$

  Skill configuration: configure arguments

$$\hat{S}_t \leftarrow \text{SkillConfigurator}(\mathcal{M}_p, S_t, o_t, \mathcal{M}).$$

  Skill execution (call grounder model if needed)

$$\text{outcome}_t \leftarrow \text{ExecuteSkill}(\hat{S}_t, \mathbb{E}).$$

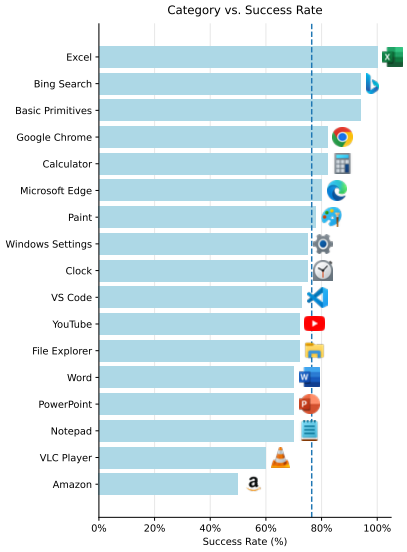
  Update memory: append skill and outcomes

$$\mathcal{M} \leftarrow \mathcal{M} \cup \{\text{Summarize}(\hat{S}_t, \text{outcome}_t)\}.$$

  $t \leftarrow t + 1$.
end while

Table 1. (Left) Statistics of CUA-Skill Execution Graph across applications. The GUI primitive statistics measures per atomic skill, how the quantity of GUI primitives distributes. (Right) Bar plot of success rate across applications.

Category	# Atomic Skills	Action Primitive Statistics		Success Rate (%)
		# Mean ± Std	Range	
Basic & Common GUI Primitives				
Basic Primitives	29	1.00 ± 0.00	[1–1]	94%
Application-Level Atomic Skill Distribution				
Amazon	20	2.40 ± 2.22	[1–9]	50%
Bing Search	19	3.20 ± 1.10	[1–4]	94%
Calculator	33	1.90 ± 0.69	[1–3]	82%
Clock	20	3.70 ± 3.38	[1–20]	75%
Excel	18	4.40 ± 5.38	[1–9]	100%
File Explorer	50	2.10 ± 2.40	[1–6]	72%
Google Chrome	31	4.10 ± 1.17	[1–12]	82%
Microsoft Edge	38	5.20 ± 3.19	[1–16]	80%
Notepad	33	5.10 ± 4.38	[1–20]	70%
Paint	7	6.70 ± 1.80	[3–9]	78%
PowerPoint	45	3.80 ± 2.02	[1–9]	70%
VLC Player	26	3.70 ± 3.02	[1–13]	60%
VS Code	20	2.90 ± 1.29	[1–7]	73%
Windows Settings	21	2.00 ± 0.97	[1–4]	75%
Word	42	3.60 ± 2.09	[1–9]	70%
YouTube	26	3.70 ± 1.21	[1–7]	72%
Total	**478**	3.75 ± 2.91	[1-20]	**76.4%**



During indexing, each skill S is embedded using its name and functional description $\mathcal{I}$. We use Qwen3-Embedding-0.6B (Yang et al., 2025a) to construct the semantic index, while maintaining an inverted index over skill text for lexical retrieval. Our architecture is also compatible with other semantic retrieval models. For each generated query, we retrieve by default top-5 most relevant skills from both channels and merge them into a consolidated candidate set, which is then passed to the subsequent skill reranker.

Skill Re-ranker. The reranker evaluates the retrieved candidate skills and selects the one to make meaningful progress toward the user goal. This evaluation relies on the current UI state, execution history, and the compatibility between candidate skills and their required arguments.

Skill Fallback. Note that in addition to retrieved skills, **the reranker also considers a small set of basic low-level actions** (e.g., mouse clicks and keyboard actions). This allows the CUA-Skill Agent to fall back to fine-grained control when high-level skills are insufficient, enhancing robustness beyond the predefined skill library.

4.3. Skill Configurator

Once a skill S is selected, the planner $\mathcal{M}_p$ instantiates its arguments $\mathcal{A}$ by conditioning on the current UI state, execution history, and user goal. Each argument is generated within its feasible domain, see Section 3.1. This domain-aware argument instantiation ensures that each configured skill corresponds to a realizable execution on the desktop. After argument configuration, the skill is fully specified, then ready for execution over the environment.

4.4. Executor

After a skill is selected and configured, the agent executes it by invoking the corresponding executable actions defined in the skill execution graph $\mathcal{G}_e$. Each execution graph specifies a parameterized realization composed of GUI interaction primitives and/or executable script actions, depending on the instantiated execution path. For execution steps that require spatial interaction with the user interface, we employ a GUI grounding model to predict interaction coordinates on the current screen. By decoupling high-level planning from low-level spatial grounding, the agent can leverage specialized perception models for accurate UI localization while allowing non-UI steps to be executed directly, improving overall execution reliability and efficiency.

Execution proceeds by traversing $\mathcal{G}_e$ in a depth-first manner to identify the next executable primitive, conditioned on the current UI state and execution context. When multiple valid successor nodes are available, the executor selects one uniformly at random by default. When edge weights are provided, the traversal policy can instead incorporate execution preferences by prioritizing successors according to their associated weights.

4.5. Memory and Reflection.

Our architecture incorporates a memory buffer that records previously executed skills and their observed outcomes, serving as a persistent substrate for agent reflection. For each executed skill, we generate a concise summary that captures both the skill's intent and its resulting effect, including whether the skill succeeded, achieved the expected outcome, or failed to execute as intended.

All such summaries are stored in the memory buffer and exposed to the planner, providing an up-to-date and reflective view of the agent's state, progress, and past decisions. Importantly, the memory module explicitly records failed skills and their contexts, enabling the planner to reason about prior mistakes, avoid repeatedly selecting ineffective actions, and adapt its strategy accordingly. Through this reflective feedback loop, the agent is encouraged to explore alternative execution paths, improving robustness and reducing unnecessary loops in long-horizon task execution.

5. Numerical Experiment

We present a comprehensive evaluation of CUA-Skill and the CUA-Skill Agent. We first assess the standalone reliability of the constructed skills and their associated execution graphs across a diverse set of desktop applications. We then evaluate the effectiveness of integrating CUA-Skill into an end-to-end computer-using agent on challenging real-time desktop benchmarks. Finally, we conduct ablation and robustness studies to isolate the contribution of individual components within the CUA-Skill framework.

5.1. Evaluation of Skills and Execution Graphs

Setup. We curate a library of 452 atomic skills spanning 17 commonly used applications on Windows OS, including File Explorer, Excel, Word, Chrome, VS Code, and system utilities, etc. To evaluate skill executability in realistic settings, we synthesize user tasks by composing atomic skills according to the skill composition graph, with arguments instantiated using the domain-aware generators described in Section 3.1. Each instantiated task is executed in isolation on a virtual machine using its corresponding parameterized execution graph. In total, we generate approximately 200K executable tasks, see examples in Appendix C. We randomly sample around 1,000 tasks for evaluation. Task outcomes are assessed using GPT-5 as an LLM-based judge, with additional human screening to ensure evaluation reliability.

Metrics. We report two metrics: *(i)* Execution success rate: measures whether the synthesized tasks covering the skills successfully completes its intent, *(ii)* Average number of primitives: counts the number of low-level actions required per skill, serving as a proxy for skill and execution complexity. In general, a higher execution success rate indicates greater reliability of skills, while a larger average number of primitives reflects higher execution complexity

Results. Table 1 summarizes the execution statistics across applications. Overall, the constructed skills achieve an average success rate of **76.4%**, with execution graphs requiring 3.75 GUI primitives per skill on average and at most 20 basic actions. This indicates strong executability while covering a broad range of interaction complexity.

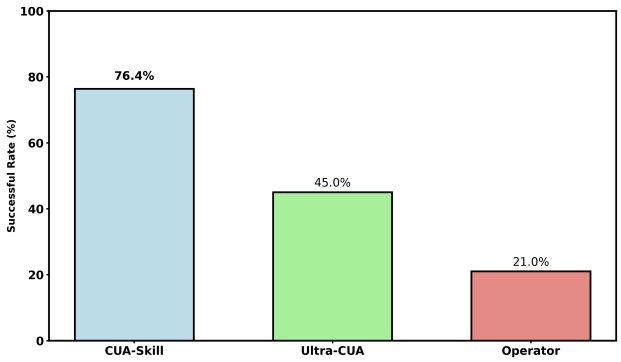


Figure 5. Synthesized User Task Successful Rate. CUA-Skill is noticeablly higher than Ultra-CUA by 1.7x, and Operator by 3.6x.

Applications with stable UI layouts and strong keyboard affordances (e.g., Excel, Settings, and Bing Search) exhibit higher success rates, whereas visually complex or media-heavy applications (e.g., VLC and PowerPoint) remain more challenging. These results demonstrate that CUA-Skill is sufficiently reliable to serve as reusable building blocks.

Direct Application: CUA Trajectory Generation. As a direct application of skill composition, CUA-Skill naturally induces executable trajectories by composing skills through their parameterized execution and composition graphs. This process yields complete, low-cost, and high-success trajectories that can serve a variety of downstream purposes. We compare CUA-Skill with existing trajectory generation systems, including UltraCUA (Yang et al., 2025b), which reports a success rate of 45%, as well as OpenAI Operator (OpenAI, 2025) evaluated on the same synthesized user tasks. As shown in Figure 5, trajectories generated by CUA-Skill achieve substantially higher success rates **1.7×**-**3.6×** higher than UltraCUA and Operator, respectively. These results suggest that CUA-Skill can alleviate the training data scarcity bottleneck of CUA.

5.2. End-to-End Performance of the CUA-Skill Agent

We next evaluate the effectiveness of CUA-Skill Agent in an end-to-end computer-using agent setting, where the agent operates directly from natural-language user instructions without skill composition available. Unlike the synthesized skill compositions in Section 5.1, CUA-Skill Agent must autonomously decide which skill to invoke, when to invoke it, and how to configure its arguments based on the current UI state and execution history. Consequently, failures may stem not only from execution errors, but also from imperfect skill retrieval, mis-ranking, or incorrect argument instantiation. This evaluation therefore provides a stringent test of whether structured and parameterized skill abstractions can support robust decision making in realistic user tasks.

Table 2. Success Rate by Application Category of CUA-Skill Agent on WindowsAgentArena (Bonatti et al., 2024).

Category	# Overall	# of Success	# of Skills Used	# of Distinct Skills Used	Avg Distinct Skills Per Task	Success Rate (%)
Chrome	17	10.9	67	19	1.82	64.11
Clock	4	4	47	5	2.50	100.00
File Explorer	19	12	112	21	2.47	63.16
Microsoft Paint	3	1	20	3	1.33	33.33
Microsoft Edge	13	7	29	20	1.69	53.85
Microsoft Excel	24	6	163	9	2.21	25.00
Microsoft Word	18	7	23	10	0.94	38.89
Notepad	2	1	19	8	4.00	50.00
Settings	5	5	11	5	2.00	100.00
VLC	21	11	56	16	1.38	52.38
VS Code	24	10	71	11	1.25	41.67
Windows Calculator	3	2	30	10	6.67	66.67
Overall	153	**87.9**	648	**117**	–	**50.26**

Table 3. Performance on the WAA Benchmark.

System	Success Rate (%)	# of Steps
Human Performance (Bonatti et al., 2024)	74.5	–
NAVI (Bonatti et al., 2024) (GPT4o)	19.5	15
UI-TARS1 (Qin et al., 2025) (Qwen2VL-FT)	15.7	50
UITARS-1.5-7B (Qin et al., 2025) (Qwen2.5VL-FT)	18.1	15
STEVE (Lu et al., 2025) (GPT4o)	20.1	40
Agent-S (Agashe et al., 2024) (GPT-4o)	18.2	15
Agent-S2 (Agashe et al., 2025a) (Claude-3.5-Sonnet)	29.8	15
UFO-2 (Zhang et al., 2025a) (GPT-o1)	30.5	30
Ultra-CUA-7B (Yang et al., 2025b) (QWen2.5VL-FT)	21.1	15
AgentS3 (Gonzalez-Pumariega et al., 2025) (GPT-5)	49.0	50
AgentS3 (Gonzalez-Pumariega et al., 2025) (GPT-5) (Best-of-3)	56.6	100
Operator (OpenAI, 2025)	37.4	50
CUA-Skill Agent (GPT-5)	**50.3**	30
CUA-Skill Agent (GPT-5, Best of 3)	**57.5**	30

Benchmarks and Metrics. Since CUA-Skill primarily focus on Windows OS, we naturally evaluate CUA-Skill Agent on the popular WindowsAgentArena benchmark (Bonatti et al., 2024). We report success rate and the number of (distinct) skills used per task. The former one indicates the overall performance of the agent that we built. The later two indicates the coverage of the CUA-Skill.

Results. Table 2 reports per-application success rates of the CUA-Skill Agent. Averaged across all evaluated tasks, the agent achieves a 50.26% success rate while using an average of 2.22 distinct skills per task. Performance varies across applications: system utilities and configuration tasks are solved reliably, whereas document editing and spreadsheet workflows remain more challenging due to dense UI interactions. These results demonstrate the effectiveness of structured skill reuse, while highlighting remaining challenges in complex application workflows. We further compare CUA-Skill Agent with existing CUA systems in Table 3. With GPT-5 as the planner, CUA-Skill Agent achieves a state-of-the-art best-of-three success rate of 57.5%, significantly outperforming existing approaches by a large margin. Moreover, in addition to its strong performance, CUA-Skill Agent completes tasks efficiently, requiring at most 30 execution steps. Notably, across all WAA evaluations, the agent invokes only 117 distinct skills out of the 478 available in CUA-Skill, indicating that the performance gains arise from general-purpose skill abstractions rather than benchmark-specific engineering.

5.3. Ablation Study

We studied the impact of different planners and the gain of CUA-Skill to improve computer use performance.

LLM Planner. Skills are designed to be model-agnostic and compatible with a wide range of LLM backbones. As shown in Table 4, we evaluate CUA-Skill Agent using Qwen3-VL-32B-Instruct, GPT-4o, and GPT-5. The results show a clear positive correlation between agent performance and the capability of the underlying language model, with GPT-5 achieving higher successful rate than less capable backbones. We further ablate the effect of reasoning depth within GPT-5. Increasing the reasoning level consistently improves task success, rising from 33.31% under minimal reasoning to 50.26% under low reasoning. This trend indicates that stronger reasoning benefits the usage of CUA-Skill for computer use.

Table 4. LLM Backbones.

Model Configuration	SR (%)
Qwen3-VL-32B-Instruct	11.77
GPT-4o	28.10
GPT-5 (Minimal Reasoning)	33.31
GPT-5 (Low Reasoning)	50.26

Table 5. Skill Integration Across Different Backbones on WAA.

Model Backbone	Baseline (No Skill)	With Skills	Improvement (Δ)
Qwen3-VL-32B-Instruct	6.54%	11.77%	**+5.23% (↑)**
GPT-4o	19.60%	28.10	**+8.50% (↑)**
GPT-5	34.64%	50.26%	**+15.62% (↑)**

Saliency of Skills. Table 5 demonstrates that skill augmentation consistently improves agent performance across all evaluated LLM backbones, with gains scaling alongside model capability. For Qwen3-VL-32B-Instruct, skills deliver a substantial improvement (**+5.23%**). GPT-4o exhibits a larger gain (**+8.50%**), reflecting improved reliability in skill selection and configuration. For GPT-5, skill integration yields the largest improvement (**+15.62%**).

6. Conclusion

We present CUA-Skill and CUA-Skill Agent, a skill-centric framework that encodes human computer-use knowledge

as reusable, parameterized skills with execution and composition graphs. CUA-Skill is highly transferable across tasks and applications, and directly enables high-success executable trajectory generation. Evaluations on WindowsAgentArena show consistent performance gains across LLM backbones, establishing CUA-Skill as a practical, model-agnostic foundation for scalable desktop agents.

Impact Statement

This work introduces CUA-Skill, a structured, skill-centric framework that encodes human computer-use knowledge as reusable, parameterized skills with execution and composition graphs, together with a retrieval-augmented CUA-Skill Agent for end-to-end desktop task execution. The framework yields two primary impacts. First, by enabling systematic skill parameterization and composition, CUA-Skill can generate large-scale, diverse, and high-success executable trajectories, which can be suggested to alleviate the data scarcity bottleneck for training computer-using agents for the future work. Second, when integrated into an online agent, these structured skills significantly improve the agent's capability, robustness, and reliability in solving real-world computer-use tasks, achieving state-of-the-art performance on WindowsAgentArena while remaining efficient and model-agnostic. Together, these results establish CUA-Skill as a practical foundation for both scalable CUA training and reliable deployment.

References

S. Agashe, J. Han, S. Gan, J. Yang, A. Li, and X. E. Wang. Agent s: An open agentic framework that uses computers like a human, 2024.

S. Agashe, K. Wong, V. Tu, J. Yang, A. Li, and X. E. Wang. Agent s2: A compositional generalist-specialist framework for computer use agents, 2025a.

S. Agashe, K. Wong, V. Tu, J. Yang, A. Li, and X. E. Wang. Agent s2: A compositional generalist-specialist framework for computer use agents, 2025b.

Anthropic. Introducing the Model Context Protocol, November 2024a. Accessed: 2026-01-24.

Anthropic. Introducing the model context protocol, Nov. 2024b. Accessed: 2026-01-20.

R. Bonatti, D. Zhao, F. Bonacci, D. Dupont, S. Abdali, Y. Li, Y. Lu, J. Wagle, K. Koishida, A. Bucker, L. Jang, and Z. Hui. Windows agent arena: Evaluating multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*, 2024.

N. Braunschweiler, R. Doddipatla, and T.-c. Zorila. Toolreagt: Tool retrieval for llm-based complex task solution via retrieval augmented generation. In *Proceedings of the 3rd Workshop on Towards Knowledgeable Foundation Models (KnowFM)*, pages 75–83, Vienna, Austria, aug 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.knowllm-1.7.

T. Chen, M. Solodko, S. Wang, J. Ko, J. Hao, C. Banbury, S. Abdali, S. Amizadeh, Q. Xiao, Y. Li, et al. Appselectbench: Application-level tool selection benchmark. *arXiv preprint arXiv:2511.19957*, 2025.

P. Chhikara, D. Khant, S. Aryan, T. Singh, and D. Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025. doi: 10.48550/arXiv.2504.19413.

G. Gonzalez-Pumariega, V. Tu, C.-L. Lee, J. Yang, A. Li, and X. E. Wang. The unreasonable effectiveness of scaling agents for computer use. *arXiv preprint arXiv:2510.02250*, 2025.

Z. Guan, J. C. L. Li, Z. Hou, P. Zhang, D. Xu, Y. Zhao, M. Wu, J. Chen, T.-T. Nguyen, P. Xian, W. Ma, S. Qin, G. Chesi, and N. Wong. Kg-rag: Enhancing gui agent decision-making via knowledge graph-driven retrieval-augmented generation. *arXiv preprint arXiv:2509.00366*, 2025. doi: 10.48550/arXiv.2509.00366.

Y. Huang, J. Shi, Y. Li, C. Fan, S. Wu, Q. Zhang, Y. Liu, P. Zhou, Y. Wan, N. Z. Gong, et al. Metatool benchmark for large language models: Deciding whether to use tools and which to use. *arXiv preprint arXiv:2310.03128*, 2023.

Z. Hui, Y. Li, D. Zhao, C. Banbury, T. Chen, and K. Koishida. WinSpot: GUI grounding benchmark with multimodal large language models. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1086–1096, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-252-7. doi: 10.18653/v1/2025.acl-short.85.

F. Lu, Z. Zhong, Z. Wei, S. Liu, C.-W. Fu, and J. Jia. Steve: A step verification pipeline for computer-use agent training, 2025.

Model Context Protocol Contributors. Model context protocol (mcp) specification (protocol revision: 2025-11-25), Nov. 2025. Accessed: 2026-01-20.

A. Notov. Introduction to Claude Skills, October 2025. Accessed: 2026-01-24.

OpenAI. Introducing operator, 2025. Accessed 2026-01-25.

J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*, 2023. doi: 10.48550/arXiv.2304.03442.

Y. Qin, Y. Ye, J. Fang, H. Wang, S. Liang, S. Tian, J. Zhang, J. Li, Y. Li, S. Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

C. Qu, S. Dai, X. Wei, H. Cai, S. Wang, D. Yin, J. Xu, and J.-R. Wen. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343, 2025.

T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022. doi: 10.48550/arXiv.2209.11302.

L. Song, Y. Dai, V. Prabhu, J. Zhang, T. Shi, L. Li, J. Li, S. Savarese, Z. Chen, J. Zhao, R. Xu, and C. Xiong. Coact-1: Computer-using agents with coding as actions, 2025.

N. Thakur, N. Reimers, A. Rücklé, A. Srivastava, and I. Gurevych. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models, 2021.

H. Wang, H. Zou, H. Song, J. Feng, J. Fang, J. Lu, L. Liu, Q. Luo, S. Liang, S. Huang, W. Zhong, Y. Ye, Y. Qin, Y. Xiong, Y. Song, Z. Wu, A. Li, B. Li, C. Dun, C. Liu, D. Zan, F. Leng, H. Wang, H. Yu, H. Chen, H. Guo, J. Su, J. Huang, K. Shen, K. Shi, L. Yan, P. Zhao, P. Liu, Q. Ye, R. Zheng, S. Xin, W. X. Zhao, W. Heng, W. Huang, W. Wang, X. Qin, Y. Lin, Y. Wu, Z. Chen, Z. Wang, B. Zhong, X. Zhang, X. Li, Y. Li, Z. Zhao, C. Jiang, F. Wu, H. Zhou, J. Pang, L. Han, Q. Liu, Q. Ma, S. Liu, S. Cai, W. Fu, X. Liu, Y. Wang, Z. Zhang, B. Zhou, G. Li, J. Shi, J. Yang, J. Tang, L. Li, Q. Han, T. Lu, W. Lin, X. Tong, X. Li, Y. Zhang, Y. Miao, Z. Jiang, Z. Li, Z. Zhao, C. Li, D. Ma, F. Lin, G. Zhang, H. Yang, H. Guo, H. Zhu, J. Liu, J. Du, K. Cai, K. Li, L. Yuan, M. Han, M. Wang, S. Guo, T. Cheng, X. Ma, X. Xiao, X. Huang, X. Chen, Y. Du, Y. Chen, Y. Wang, Z. Li, Z. Yang, Z. Zeng, C. Jin, C. Li, H. Chen, H. Chen, J. Chen, Q. Zhao, and G. Shi. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning, 2025a.

H. Wang, H. Zou, H. Song, J. Feng, J. Fang, J. Lu, L. Liu, Q. Luo, S. Liang, S. Huang, et al. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*, 2025b. doi: 10.48550/arXiv.2509.02544.

X. Wang, B. Wang, D. Lu, J. Yang, T. Xie, J. Wang, J. Deng, X. Guo, Y. Xu, C. H. Wu, Z. Shen, Z. Li, R. Li, X. Li, J. Chen, B. Zheng, P. Li, F. Lei, R. Cao, Y. Fu, D. Shin, M. Shin, J. Hu, Y. Wang, J. Chen,

Y. Ye, D. Zhang, D. Du, H. Hu, H. Chen, Z. Zhou, H. Yao, Z. Chen, Q. Gu, Y. Wang, H. Wang, D. Yang, V. Zhong, F. Sung, Y. Charles, Z. Yang, and T. Yu. Opencua: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*, 2025c. doi: 10.48550/arXiv.2508.09123.

T. Xie, D. Zhang, J. Chen, X. Li, S. Zhao, R. Cao, T. J. Hua, Z. Cheng, D. Shin, F. Lei, Y. Liu, Y. Xu, S. Zhou, S. Savarese, C. Xiong, V. Zhong, and T. Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.

A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Y. Yang, Z. Yang, Z.-Y. Dou, A. Nguyen, K. You, O. Attia, A. Szot, M. Feng, R. Ramrakhya, A. Toshev, et al. Ultracua: A foundation model for computer use agents with hybrid action. *arXiv preprint arXiv:2510.17790*, 2025b.

S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. 2023. Publisher Copyright: © 2023 11th International Conference on Learning Representations, ICLR 2023. All rights reserved.; 11th International Conference on Learning Representations, ICLR 2023 ; Conference date: 01-05-2023 Through 05-05-2023.

C. Zhang et al. Ufo2: The desktop agentos, 2025a.

M. Zhang, Z. Xu, J. Zhu, Q. Dai, K. Qiu, Y. Yang, C. Luo, T. Chen, J. Wagle, T. Franklin, and B. Guo. Phi-ground tech report: Advancing perception in gui grounding. *arXiv preprint arXiv:2507.23779*, 2025b. doi: 10.48550/arXiv.2507.23779.

M. Zhang, Z. Xu, J. Zhu, Q. Dai, K. Qiu, Y. Yang, C. Luo, T. Chen, J. Wagle, T. Franklin, et al. Phi-ground tech report: Advancing perception in gui grounding. *arXiv preprint arXiv:2507.23779*, 2025c.

H. Zhao, T. Chen, and Z. Wang. On the robustness of gui grounding models against image attacks. *arXiv preprint arXiv:2504.04716*, 2025.

Y. Zhuang, Y. Yu, K. Wang, H. Sun, and C. Zhang. Toolqa: A dataset for llm question answering with external tools. *arXiv preprint arXiv:2306.13304*, 2023. doi: 10.48550/arXiv.2306.13304.

A. More Related Works

GUI Grounding in Desktop Agents GUI grounding is central to dekstop-agent planning since each natural-language step must be bound to a specific on-screen target before an action can be executed (Hui et al., 2025). In desktop environments, grounding must be repeated after every state change (Xie et al., 2024). OS-agent evaluations show that minor grounding errors quickly compound into multi-step failures (Bonatti et al., 2024). Recent work improves this by strengthening UI perception and reference resolution (Zhang et al., 2025b; Wang et al., 2025b; Zhao et al., 2025). CUA-Skill uses grounding as a planning constraint, aligning each candidate action with spatial and contextual cues from the current interface state.

Retrieval-Augmented Planning Retrieval-augmented planning interleaves planning and action, letting an agent revise its next step using environment feedback instead of committing to a full plan up front (Yao et al., 2023). A challenge is tool orchestration. Agents must decide whether an external tool is needed and select an appropriate tool given the current objective and context (Huang et al., 2023). Retrieval addresses scalability by narrowing a large action/tool space to a small set of relevant candidates. These candidates can then be ranked and invoked within the reasoning loop rather than treated as static menus (Braunschweiler et al., 2025; Qu et al., 2025; Schick et al., 2023). CUA-Skill follows this paradigm by retrieving and ranking atomic template conditions on the current goal, enabling more targeted planning and tool selection.

B. Example of Ensembled Query Generation

Ensembled Query Generation Example

Instruction: Next: Open Edge Home Page.
Query 1: Open home page in Edge.
Query 2: Double-click Microsoft Edge icon to open it and navigate to the home page.
Query 3: Use Windows menu to launch Edge.

C. Example of Synthesized Tasks by Skill Composition Graph

Synthesized User Task Example 1 upon Skill Composition Graph

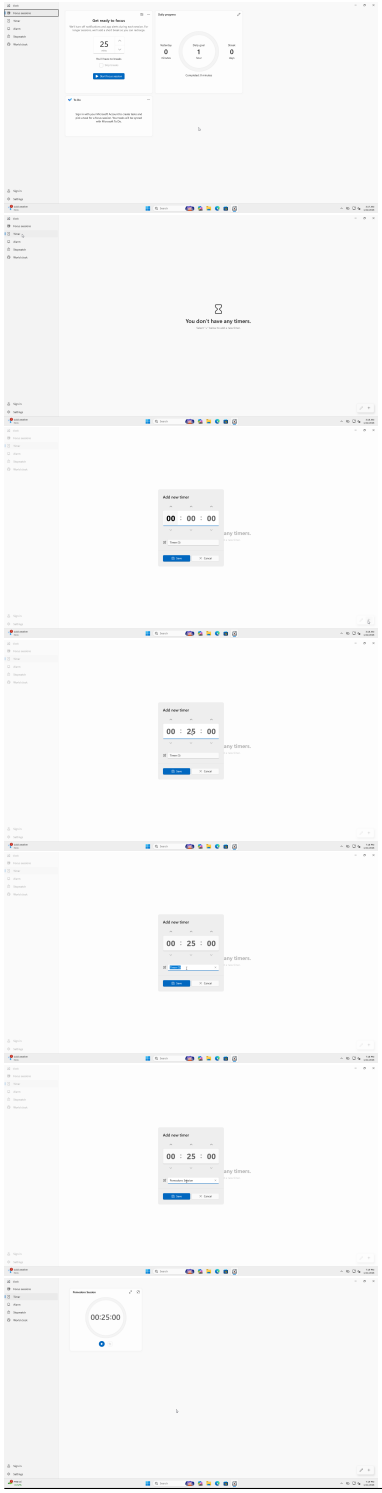
ID: excel_005
Domain: Excel
Instruction: Open the 'betawacc.xlsx' file, rename the sheet1 as company analysis and fulfill the average column.
Steps: (A sequence of skills to complete the instruction)
ExcelOpenExistingWorkbook, `file_path=betawacc.xlsx`.
ExcelRenameSheet, `target_sheet_name=sheet1`, `new_sheet_name=company analysis`.
ExcelInsertFunctionCall, `target_cell=F7`, `function_call_command=AVERAGE(C7:E7)`.
ExcelAutoFillDown, `start_cell=F7`, `end_cell=F10`.

Synthesized User Task Example 2 upon Skill Composition Graph

ID: calculator_01
Domain: Calculator
Instruction: Calculate $398 - 174 \times \sqrt{505}$
Steps: (A sequence of skills to complete the instruction)
CalculatorLaunch.
CalculatorSwitchMode, `mode_name=scientific`.
CalculatorEnterNumber, `number=398`.
CalculatorSubtract.
CalculatorEnterNumber, `number=174`.
CalculatorMultiply.
CalculatorSquareRoot, `number=505`.
CalculatorEquals.

D. Case Study

Case Study: Skill: `ClockCreateTimer`. **Task:** Create a 25 minute timer called Pomodoro Session.



Step 1: ClockSwitchTab
Reasoning. Click on the 'Timer' tab to switch view.
Argument Instantiation.

- `tab`: Timer.

Step 2: `SingleClickAction`
Reasoning. Click add timer button.
Argument Instantiation.

- `Coordinate:` Call grounding model.
- `Button:` Left

Step 3: `SingleClickAction`
Reasoning. Focus on minutes input.
Argument Instantiation.

- `Coordinate:` Call grounding model.
- `Button:` Left

Step 4: `TypeAction`
Reasoning. Enter minutes '25'.
Argument Instantiation.

- `Input_mode:` 'keyboard'
- `Text:` 25

Step 5: `SingleClickAction`
Reasoning. Locate and click the timer name input field (e.g., placeholder 'Name' or 'Timer name').
Argument Instantiation.

- `Coordinate:` Call grounding model.
- `Button:` Left.

Step 6: `TypeAction`
Reasoning. Type timer label 'Pomodoro Session'.
Argument Instantiation.

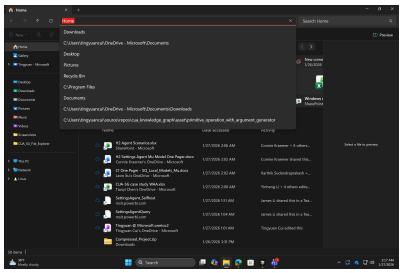
- `Input_mode:` 'copy_paste'
- `Text:` 'Pomodoro Session'.

Step 7: `SingleClickAction`
Reasoning. Save timer..
Argument Instantiation.

- `Coordinate:` Call grounding model.
- `Button:` Left.

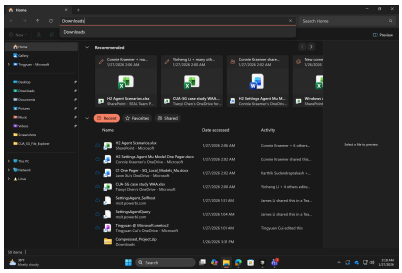
Outcome. The agent successfully completed the skill.
Key Insight. Skill arguments either need Planner to configure or call grounding model to predict, e.g., the coordinate.

Case Study: Skill: `FileExplorerCreateNewFolder`. **Task:** Create a new folder named Logs inside Downloads.



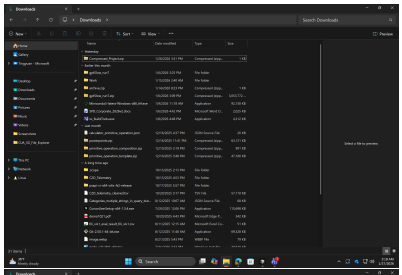
Step 1: `HotKeyAction`
Reasoning. Focus the address bar.
Argument Instantiation.

- `keys`: [‘ctrl’, ‘l’].



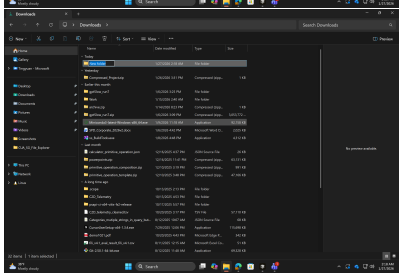
Step 2: `TypeAction`
Reasoning. Type path ‘Downloads‘ to search for it.
Argument Instantiation.

- `input_mode`: ‘keyboard’
- `text`: ‘Downloads’



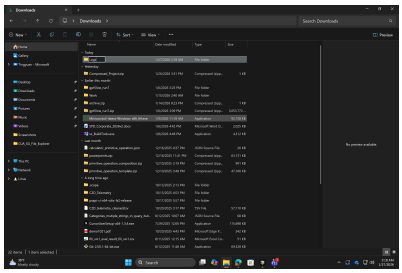
Step 3: `HotKeyAction`
Reasoning. Open folder.
Argument Instantiation.

- `key`: ‘Enter’



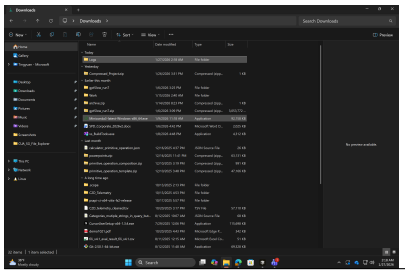
Step 4: `HotKeyAction`
Reasoning. Create new folder.
Argument Instantiation.

- `keys`: [‘ctrl’, ‘shift’, ‘n’]



Step 5: `TypeAction`
Reasoning. Set folder name.
Argument Instantiation.

- `input_mode`: ‘copy_paste’
- `text`: ‘Logs’

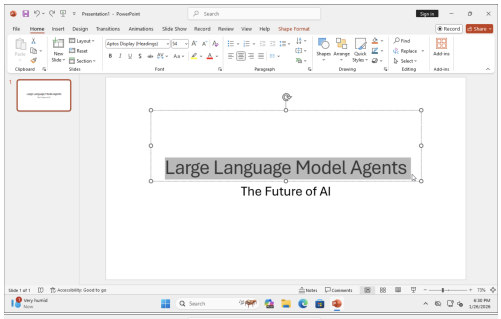


Step 6: `HotKeyAction`
Reasoning. Confirm new folder.
Argument Instantiation.

- `key`: ‘Enter’

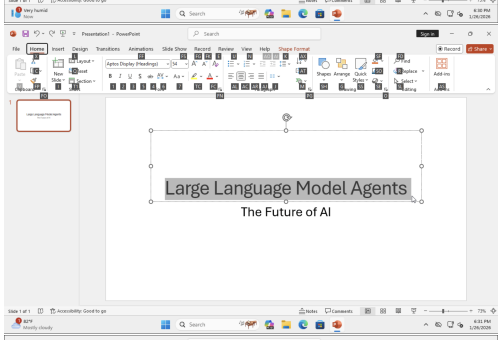
Outcome. The agent successfully completed the skill.
Key Insight. The skill uses hotkey shortcuts in File Explorer to create and rename a folder, reducing failures from visual grounding, and only calling the grounding model to predict coordinates when needed.

Case Study: Skill: `PowerPointSetTextFontColor`. **Task:** Set the font color of selected text box to Light Blue.



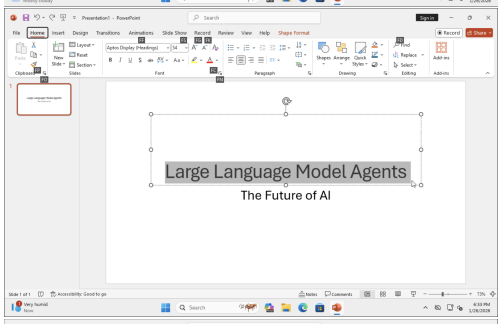
Step 1: HotKeyAction
Reasoning. Press Alt + H to switch to the Home tab.
Argument Instantiation.

- `keys`: ['alt', 'h'].



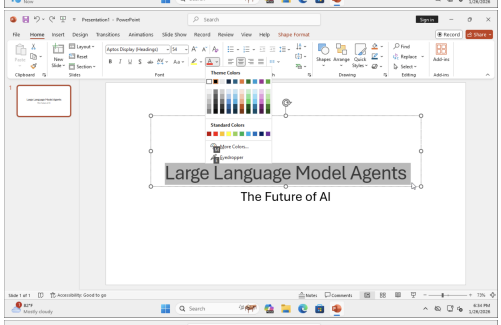
Step 2: `PressKeyAction`
Reasoning. Press F and C to open the font color dropdown menu. First press F.
Argument Instantiation.

- `key`: 'f'



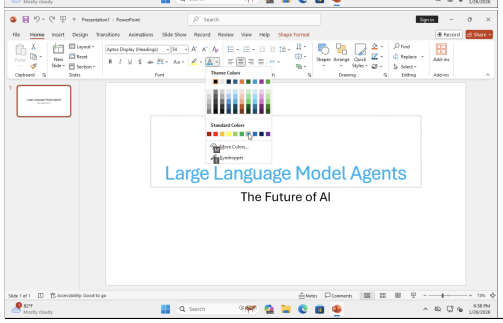
Step 3: `PressKeyAction`
Reasoning. Then press C to complete the opening of the font color dropdown menu.
Argument Instantiation.

- `key`: 'c'



Step 4: `SingleClickAction`
Reasoning. Click on the 'Light Blue' color to set it for the selected text.
Argument Instantiation.

- `Coordinate`: Call grounding model.
- `Button`: Left.



Step 5: `HotKeyAction`
Reasoning. Press Escape to close the font color dropdown menu.
Argument Instantiation.

- `keys`: ['esc'].

Outcome. The agent successfully completed the skill.
Key Insight. The skill uses hotkey shortcuts in PowerPoint to navigate in the ribbon, which reduces the possible failures from visual grounding, and only call grounding model to predict coordinates when needed.

E. CUA Execution Graph

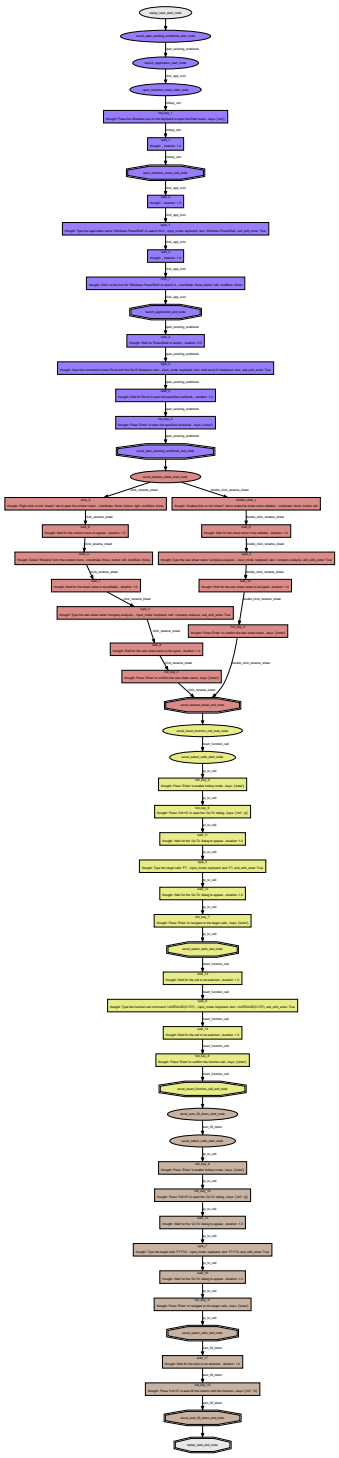


Figure 6. **CUA Task Execution Graph Example for Excel.** Open the file `betawacc.xlsx`, rename `Sheet1` to `company analysis`, and compute the `Average` column.

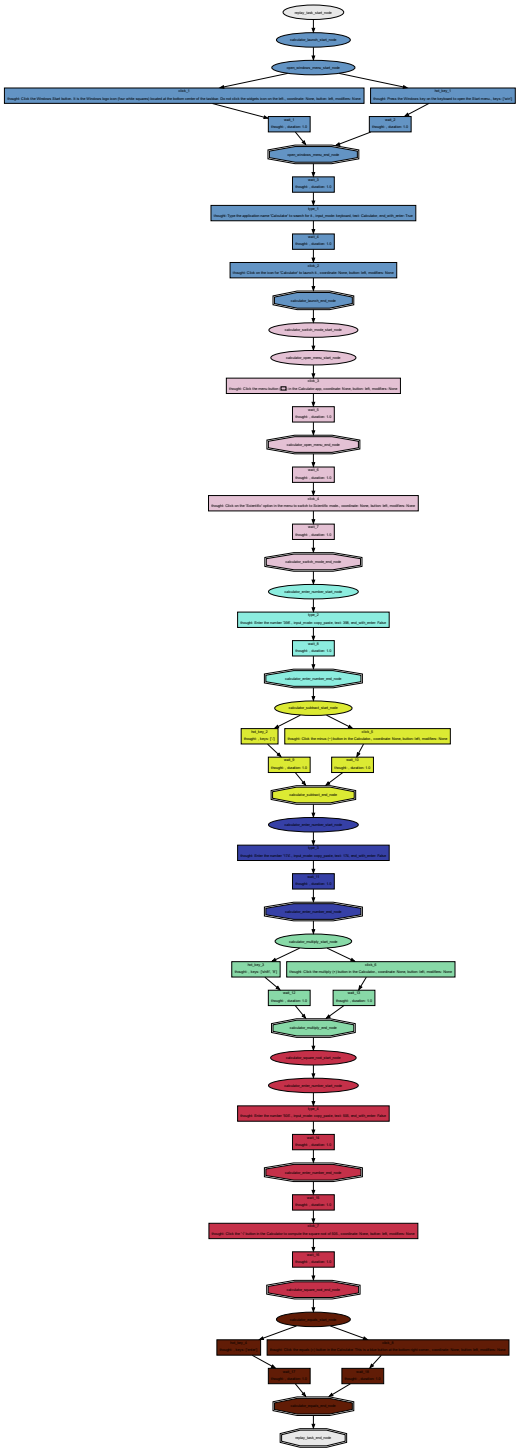


Figure 7. **CUA Task Execution Graph Example for Calculator.** Instruction: Calculate $398 - 174 \times \sqrt{505}$.