

# AgentStore: Scalable Integration of Heterogeneous Agents As Specialized Generalist Computer Assistant

Chengyou Jia<sup>1,2</sup>, Minnan Luo<sup>1,4</sup>✉, Zhuohang Dang<sup>1,5</sup>, Qiushi Sun<sup>2,3</sup>, Fangzhi Xu<sup>1,2</sup>, Junlin Hu<sup>2</sup>, Tianbao Xie<sup>3</sup>, Zhiyong Wu<sup>2</sup>✉

<sup>1</sup>School of Computer Science and Technology, Xi'an Jiaotong University

<sup>2</sup>Shanghai Artificial Intelligence Laboratory <sup>3</sup>The University of Hong Kong

<sup>4</sup>Ministry of Education Key Laboratory of Intelligent Networks and Network Security, China

<sup>5</sup>Shaanxi Province Key Laboratory of Big Data Knowledge Engineering, China

cp3jia@stu.xjtu.edu.cn minnluo@xjtu.edu.cn wuzhiyong@pjlab.org.cn

## Abstract

Digital agents capable of automating complex computer tasks have attracted considerable attention. However, existing agent methods exhibit deficiencies in their generalization and specialization capabilities, especially in handling open-ended computer tasks in real-world environments. Inspired by the rich functionality of the App store, we present **AgentStore**, a scalable platform designed to dynamically integrate heterogeneous agents for automating computer tasks. AgentStore allows the system to continuously enrich its capabilities and adapt to rapidly evolving operating systems. Additionally, we propose a novel core **MetaAgent** with the **AgentToken** strategy to efficiently manage diverse agents and utilize their specialized and generalist abilities for both domain-specific and system-wide tasks. Extensive experiments on three interactive real-world benchmarks demonstrate that AgentStore significantly expands the capability boundaries of agent systems in both generalization and specialization, underscoring its potential for developing the specialized generalist<sup>1</sup> computer assistant.

## 1 Introduction

The continual evolution of computer Operating Systems (OS), along with proliferating applications, has transformed how people work and live. This transformation goes beyond daily life like shopping and gaming, encompassing professional works such as writing in Office or editing in Photoshop. However, this increased functionality comes with a steep learning curve, often burdening users. As a result, autonomous computer assistants—once limited to fiction like *JARVIS in Iron Man* or *MOSS in Wandering Earth*—have become a concrete pursuit, attracting great interest from researchers.

✉ Equal corresponding author.

<sup>1</sup>The concept of “Specialized Generalist” refers to an AI system that excels in specific tasks while still maintaining broad general capabilities (Zhang et al., 2024).

Advancements in Multimodal Large Language Models (MLLMs) (OpenAI, 2023; Reid et al., 2024), are gradually turning this vision into reality. However, real-world OS environments encompass a diverse array of open-ended computer tasks, each with inherent requirements for capabilities across multi-dimensions (Xie et al., 2024), posing substantial challenges to existing methods. Specifically, “Task\_1” in Figure 1 illustrates that many computer tasks necessitate specific knowledge and operations. In such scenarios, existing generalist agents (Wu et al., 2024; Tan et al., 2024) often underperform due to their lack of these specialized abilities. Conversely, specialized agents, despite excelling at specific tasks within single domains like tabular data processing (Li et al., 2024; Chen et al., 2024a) or web browsing (Zhou et al., 2023; Deng et al., 2024), struggle to perform when confronted with more integrated, system-wide tasks like “Task\_2” in Figure 1. This heterogeneous demand for capabilities presents a challenge for existing single generalist or specialized agents.

We attribute this dilemma to overlooking a key factor behind the success of modern operating systems: App store<sup>2</sup> which continuously expands the range of functionalities beyond the core OS itself. Correspondingly, we argue that *specialized generalist computer agents should possess this characteristic, evolving to grow heterogeneous abilities and autonomously handle an increasingly diverse range of tasks*. To substantiate this, we propose **AgentStore**, a flexible and scalable platform for dynamically integrating various heterogeneous agents to independently or collaboratively automate OS tasks (illustrated on the right in Figure 1).

We develop a prototype of AgentStore, establishing an integration protocol and implementing over 20 computer OS agents and 10 mobile OS agents

<sup>2</sup>In this paper, App store not only refers to App Store for Apple but all similar platforms. See the concept in App store.

**Task 1:** In a new sheet with 4 headers "Year", "CA changes", "FA changes", and "OA changes", calculate the annual changes for the Current Assets, Fixed Assets, and Other Assets columns.

Year	Current Assets	Fixed Assets	Other Assets	Assets	Current Liabilities	Long-term Liabilities	Owner's Equity
2014	\$ 165,682.00	\$ 45,500.00	\$ 3,580.00	\$ 6,762.00	\$ 50,000.00	\$ 172,474.00	
2015	\$ 204,527.00	\$ 43,243.00	\$ 3,520.00	\$ 7,653.00	\$ 50,000.00	\$ 196,318.00	
2016	\$ 219,289.00	\$ 40,840.00	\$ 3,726.00	\$ 8,258.00	\$ 40,000.00	\$ 220,797.00	
2017	\$ 246,718.00	\$ 38,419.00	\$ 4,011.00	\$ 9,133.00	\$ 40,000.00	\$ 239,576.00	
2018	\$ 264,792.00	\$ 35,854.00	\$ 4,030.00	\$ 9,839.00	\$ 30,000.00	\$ 253,852.00	
2019	\$ 282,148.00	\$ 33,181.00	\$ 4,088.00	\$ 10,585.00	\$ 30,000.00	\$ 282,688.00	



Step 1: Install and locate

```
pip install openpyxl && Isolf | grep '.xlsx'
```

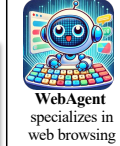
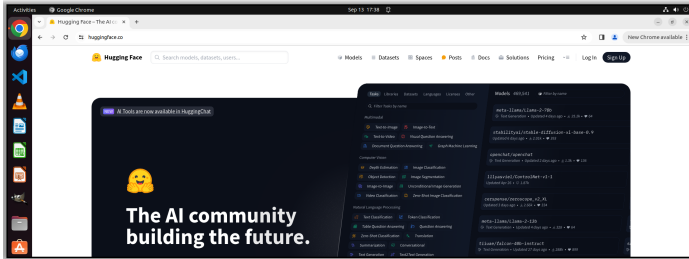
Step 2: Create new sheet and add headers

```
ws_new = wb.create_sheet(title=sheet_name)
ws_new.append(headers), wb.save(file_path)
```

Step 3: Insert table for the required data

```
for row in range(2, ws_original.max_row + 1):
    year = ws_original.cell(row, 1).value, ...
    ws_new.append([year, ...])
```

**Task 2:** Find the daily paper and take down the meta information of papers on 1st March, 2024 in the opened .pptx file. Please conform to the format and complete others.



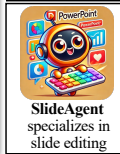
Different specialist agents are required to collaborate on system-wide tasks

SubTask 1: Find papers and extract meta info

Step 1: Click daily papers to browsing  
Step 2: Filter results by choosing 1st March  
Step 3: Extract info for selecting papers

subtask complete → message passing

SubTask 2: write meta info into pptx



Step 1: Install package and locate .pptx file  
Step 2: load content for current .pptx file  
Step 3: Write info into corresponding file  
Step 4: Save and overwrite the original file

Figure 1: Task examples illustrate that diverse open-ended tasks require a combination of generalization and specialization capabilities. The right part provides a simple overview of specific steps.

with diverse functionalities. Based on this foundation, the main challenge is efficiently managing the rapidly growing and increasingly large number of agents, which overwhelms traditional management methods, such as In-Context Learning (ICL; Dong et al., 2022) and full Fine-Tuning (FT; Qin et al., 2023). To this issue, we introduce a novel MLLM-based **MetaAgent** with **AgentToken** strategy, to select the most suitable agent(s) to complete tasks. Each integrated agent in AgentStore is denoted as a learnable token embedding in MetaAgent’s architecture like a word token embedding. During inference, MetaAgent activates the corresponding agent to execute the task when an agent token is predicted. Innovatively, we enhance this approach by shifting from single-token (Hao et al., 2024) to multi-token prediction, allowing MetaAgent to predict and coordinate multiple agents for collaborative task execution. Additionally, we propose an automated process with self-instruct for tuning AgentToken without relying on manual data.

We validate the effectiveness of AgentStore across three real-world dynamic benchmarks involving both computer and mobile OS platforms. On the highly challenging OSWorld benchmark, AgentStore achieves a success rate of 23.85%, more than doubling the performance of the previous generalist agent (11.21%) (Xie et al., 2024). On a novel benchmark OSWorld-Multi for evaluating agent collaboration, AgentStore significantly improves the execution success rate from 6.93% to 22.77%, and achieves strong performance across all metrics. These results highlight the importance of scalable agent integration in expanding the system’s capabilities. Similar outcomes are ob-

served when evaluating AgentStore in mobile environments. Additionally, we also demonstrate the advantage of AgentToken in comparison to other strategies, highlighting its efficiency in training and its effectiveness in dynamic management.

## 2 Related Work

**LLM-based Agents.** Recent advancements in (M)LLMs (OpenAI, 2023; Reid et al., 2024) have led to the development of highly capable AI agents, applied across various domains, including robotics (Driess et al., 2023), software development (Wang et al., 2024), and beyond. A rapidly growing research field among these is automating interactions with computer environments to solve complex tasks. Early work primarily focused on specific scenarios, such as web manipulation (Yao et al., 2022; Deng et al., 2024; Cheng et al., 2024), command-line coding (Sun et al., 2024), and gaming (Wang et al., 2023a). Following this, recent methods (Wu et al., 2024; Tan et al., 2024) have started exploring general-purpose computer agents capable of interacting with diverse components in OS. Unfortunately, both of these struggle with open-ended tasks in real environments, exposing limitations in their generalization and specialization capabilities.

**Multi-Agent Systems.** Recently, various approaches (Park et al., 2023; Sun et al., 2023; Hong et al., 2023) have been proposed to facilitate effective collaboration among multi-agents to overcome hallucinations. Advanced frameworks like AutoGen (Wu et al., 2023) and Dylan (Liu et al., 2023) have demonstrated flexible agent orchestration capabilities, primarily focusing on tool-based

task execution with structured workflows. Similarly, Camel and the concurrent IoA explore agent integration for specialized tasks. While these approaches have shown promising results in domains such as automating coding, their application to OS environments presents unique challenges. First, OS tasks demand dynamic integration across heterogeneous applications with varying operational constraints and system-level interactions. Second, OS automation requires specialized capabilities across diverse system operations, from low-level file management to application-specific automation. These unique challenges motivate AgentStore’s design to expand multi-agent capabilities specifically for real-world OS applications.

### 3 AgentStore

As illustrated in Figure 2, AgentStore consists of two main components: AgentPool and MetaAgent. The AgentPool (in Section 3.1) stores feature-specific agents with distinct functionalities and defines the integration protocol for adding new agents. The MetaAgent (in Section 3.2 and 3.3) selects the most suitable agent(s) from AgentPool to independently or collaboratively complete tasks.

#### 3.1 AgentPool

The AgentPool is a collection of all available agents within AgentStore. When the developer creates a new agent and seeks to integrate it into AgentStore, it is essential to register the agent’s information in a standardized format. To ensure consistency in the integration process, we establish an **integration protocol**. During enrolling, the developer completes a predefined form outlining the agent’s capabilities, limitations, applications, and demonstrations of its functionality (in Figure 2). Formally, the set of all enrolled agents is represented as  $\mathcal{A} = \{(a_1, d_1), (a_2, d_2), \dots, (a_n, d_n)\}$ , where the completed form for each agent  $a_i$  constitutes a document  $d_i$ . To build the prototype of AgentStore, we create 20 computer agents and 10 mobile agents to handle tasks on their respective platforms. We employ both manual and automated agent generation. **Manual Generation for Computer:** Due to operation inconsistencies among computer apps, achieving uniform automated agent generation remains a significant challenge. Consequently, we manually designed computer agents tailored to common applications. These agents are evolved by injecting domain-specific knowledge documents into

GUI-based MMAgent (Xie et al., 2024) and CLI-based FridayAgent (Wu et al., 2024). These agents range from unimodal to multimodal, from open-source to closed-source models, and from GUI to CLI. This heterogeneous combination provides a solid foundation to validate the effectiveness of the AgentStore concept. The details of these agents are presented in Appendix A and B.

**Automated Generation for Mobile:** In contrast, the operations of mobile apps are entirely GUI-based with a more uniform design, enabling us to automate the generation of domain-specific agents. Following the approach of APPAgent (Yang et al., 2023), we initially develop a general GUI-based APPAgent. Subsequently, we guide it with self-exploration to navigate and interact with nine common mobile apps, ultimately evolving it into specialized agents for mobile platforms. This approach minimizes manual effort and ensures the efficient creation of tailored agents for mobile applications.

#### 3.2 MetaAgent with AgentToken

We employ the powerful open-source MLLM as the foundation for our MetaAgent  $M$ . This enables it to process multi-modal information covering task descriptions and OS states. Since the number of agents in AgentStore dynamically grows and reaches a large scale, common methods like In-Context Learning (ICL) (Chase, 2022; Li et al., 2023) and Fine-Tuning (FT) (Qin et al., 2023) become impractical due to the excessive context length and the high cost of retraining, respectively.

To address these, we propose the **AgentToken** strategy. Inspired by ToolkenGPT (Hao et al., 2024), which captures tool semantics using special tokens, AgentToken extends this concept by encoding enrolled agents as special tokens in the MetaAgent’s vocabulary. Specifically, the agent tokens are parameterized as an embedding matrix  $W_{\mathcal{A}} \in R^{|\mathcal{A}| \times d}$  and appended to the original word token head  $W_{\mathcal{V}} \in R^{|\mathcal{V}| \times d}$ . Assuming the agent tokens  $W_{\mathcal{A}}$  have been trained and available (as described in Section 3.3), the concatenated result forms the new language modeling head of MetaAgent. In this way, MetaAgent predicts the next token with the following probability:

$$P_M(t_i | t_{<i}) = \text{softmax}([W_{\mathcal{V}}; W_{\mathcal{A}}] \cdot h_{i-1}),$$

where the next token can be either a word token or an agent token, *i.e.*,  $t_i \in \mathcal{V} \cup \mathcal{A}$ . The operation  $[\cdot]$  denotes concatenation, and  $h_{i-1} \in R^d$  represents

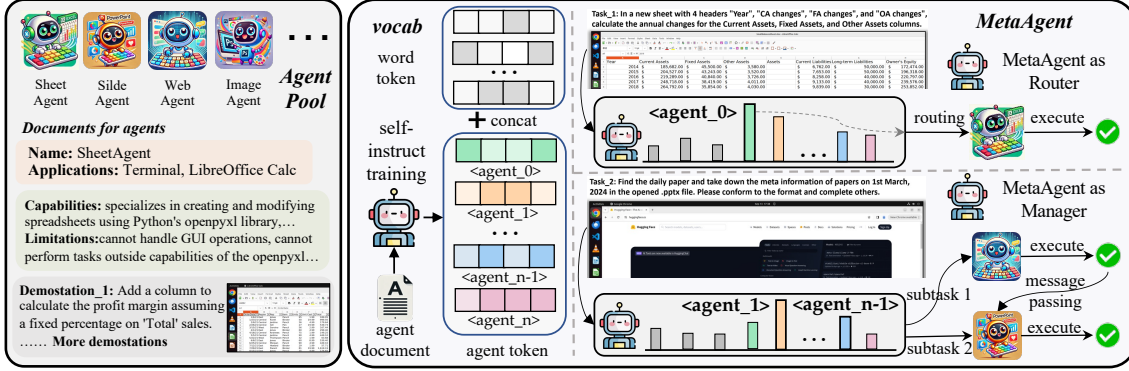


Figure 2: The illustration on the main components in AgentStore.

the last hidden state. In this context, AgentToken enables MetaAgent with two primary functions: **MetaAgent as Router:** Following the above manner, the most probable next token is obtained by maximizing the conditional probability:

$$t_i^* = \arg \max_{t_i \in \mathcal{V}_{UA}} (P_M(t_i | t_{<i})).$$

Once an agent token is predicted, *i.e.*,  $t_i^* \in \mathcal{A}$ , the MetaAgent halts decoding, and the corresponding agent is invoked to execute the task. As illustrated in Figure 2, the above method enables MetaAgent to act as an efficient router, predicting the most appropriate agent when a single agent is sufficient. **MetaAgent as Hash Manager:** We discover that, although each agent token is trained on individual tasks, they exhibit generalization capabilities for complex, collaborative tasks. Specifically, when a task requires multiple agents, the trained agent tokens often appear among the top candidates in the next token predictions. This observation led us to enhance this approach by shifting from single-token to multi-token prediction:

$$T_i^* = \text{TopK}_{t \in \mathcal{A}} (P_M(t_i | t_{<i}), K),$$

where  $\text{TopK}(\cdot)$  is a function that returns the set of  $K$  tokens from the vocabulary  $\mathcal{A}$  that have the highest probabilities. These predicted tokens represent the  $K$  agents most relevant to this task. The MetaAgent then switches to Manager mode by using a new prompt consisting of in-context documents for these selected agents, outlining how to generate subtasks for the complex task and assign them to the corresponding agents. This narrows the management scope to a few selected agents, leaving ample context space for detailed documentation of these fixed agents. This design shares similarities with hashing methods (Aggarwal and Verma, 2015), which convert inputs of arbitrary size into fixed-size outputs to facilitate retrieval.

### 3.3 Train AgentToken with SELF-INSTRUCT

The embedding  $W_{\mathcal{A}}$  corresponding to agent tokens are the only tunable parameters, introducing minimal additional training overhead. However, training these agent tokens requires a number of agent demonstrations that consist of the task descriptions and initial OS states. The corresponding token demonstrations were pre-collected for training in previous efforts (Hao et al., 2024; Chai et al., 2024). However, this strategy is not applicable in our scenario, as developers only provide a document about the agent, and it is unrealistic to expect them to supply massive demonstrations. Therefore, we propose an automated process with self-instruct (Wang et al., 2023b) for tuning these tokens.

The overall process follows an iterative algorithm to guide the generation of extra demonstrations, beginning with a limited set of original demonstrations  $S_i = \{(y_k)\}_{k=1}^{n_i}$  and the agent description  $c_i$  provided in document  $d_i$ . Specifically, we first prompt MetaAgent with existing demonstrations and agent descriptions:

$$S'_i = M(S_i, c_i),$$

where MetaAgent  $M$  is expected to produce the new set of demonstrations  $S'_i$ . Following this, to ensure the quality of the generated outputs, we apply BERTScore (Zhang et al., 2019) to all newly generated outputs  $y' \in S'_i$ , ensuring both consistency and diversity. Specifically, we use a greedy algorithm (see Appendix D) to iteratively filter elements from  $S'_i$ , resulting in a refined set  $S_i^{\text{new}} \subseteq S'_i$ . The new set satisfies the following conditions:

$$\begin{aligned} \tau_1 &\leq \text{BETRScore}(y_k, y_j) \leq \tau_2, \\ \forall y_k, y_j &\in S_i \cup S_i^{\text{new}} \text{ and } k \neq j, \end{aligned}$$

where  $\text{BETRScore}(\cdot)$  represents the similarity between two demonstrations, with imposing a lower bound  $\tau_1$  to avoid overly irrelevant outputs and  $\tau_2$

ensuring diversity among them. In this way, we filter the generated data, and the refined set is merged, *i.e.*,  $S_i = S_i \cup S_i^{new}$ . The entire process is an automated iterative bootstrapping. MetaAgent further generates additional examples until sufficient demonstrations to meet the training requirements. **Training with self-generated data:** During training, each task description and initial state in  $S_i$  serve as the prefix, and a special agent token  $\langle \text{Agent}_i \rangle$  is appended as the ground truth for the next token prediction. The training objective is:

$$\mathcal{L}(W_{\mathcal{A}}) = \sum_i^{|A|} \sum_{y_j \in S_i} -\log P(\langle \text{Agent}_i \rangle | y_j),$$

the embedding  $W_{\mathcal{A}}$  represents the only tunable parameters for all agents  $\mathcal{A}$  in AgentPool. Notably, this training paradigm simply introduces additional tokens to the MetaAgent. The original language generation of the MLLM remains entirely unaffected when agent tokens are masked. This guarantees that ICL methods can be invoked seamlessly.

## 4 Experiments

To assess the effectiveness and versatility of AgentStore, we conduct comprehensive experiments across various tasks. We demonstrate its outstanding performance on three real-world OS benchmarks (in Section 4.1), followed by an in-depth analysis of agent integration and AgentToken (in Section 4.2 and 4.3). We also provide running cases for qualitative analysis (in Section 4.4).

**Settings** We employ InternVL2-8B (Chen et al., 2024b) as the base model. Additionally, details regarding the Agents in the AgentPool can be found in Appendix A, along with the threshold selection for  $\tau_1$  and  $\tau_2$  in Appendix D. We generate 100 examples for each agent using self-instruct for token training. We also apply BertScore filtering to exclude any overlap with test tasks, thereby preventing data leakage. The AdamW optimizer is used with a learning rate of  $4e-5$  and a weight decay of 1.0, for a total of 10 training epochs. When executing the Hash Manager,  $K$  is set to 5.

### 4.1 Main Results on Real-world Benchmarks

#### 4.1.1 Computer OS Assistant: OSWorld

OSWorld (Xie et al., 2024) provides a scalable and real environment for evaluating computer agents, encompassing 369 tasks involving real web and desktop applications across open domains. As one

of the most realistic and challenging benchmarks, OSWorld is ideal for capturing the diversity and complexity of computer tasks. Details of OSWorld are provided in the Appendix E.

Table 1 presents the performance comparison between our approach and previous SoTA generalist agents on OSworld. While more advanced base models can improve performance (*e.g.*, GPT-4o outperforming GogVLM in **CogAgent** (Hong et al., 2024)), even the best base models still face significant challenges. Notably, these methods exhibit not only overall weak performance but also significant disparities and weaknesses in specific task categories, despite using the same base models. For instance, **MMAgent** (Xie et al., 2024) and **CRA-DLE** (Tan et al., 2024) struggle with calculation tasks due to their lack of knowledge and operational capability in Excel, while **Friday** (Wu et al., 2024) and **Open-Interpreter** (Open-Interpreter, 2024), CLI-based agents, fails to execute GUI operation effectively in tasks, *e.g.*, Chrome or Thunderbird.

In contrast, AgentStore overcomes these limitations by integrating various specialized agents, each proficient in specific software and operations (the base model “Hybrid” indicates that these agents are built on different base models, as detailed in Appendix A). “AgentStore(GT)” in Table 1 refers to each task being assigned to the most suitable agents, representing the upper bound of performance for the current AgentStore implementation. As shown, using specialized agents to handle tasks in their respective domains consistently outperforms generalist agents, with no significant performance shortcomings in almost all domains. This underscores the importance of various capabilities. Furthermore, when different methods are used to manage task allocation, all approaches outperform previous single-agent systems. AgentToken (AT) demonstrates the best performance. We will elaborate on this in Section 4.3.

**Cost Analysis:** While AgentStore’s deployment introduces some overhead, it is able to assign specialized agents to complete tasks with fewer steps and lower costs. We detail this in Appendix C.

#### 4.1.2 Agent Collaboration: OSWorld-Multi

We develop a novel benchmark OSWorld-Multi based on OSWorld, comprising over 100 diverse tasks that necessitate collaboration among multiple agents. This newly proposed benchmark allows us to assess the accuracy of both task decomposition and subtasks handling in a real-world environment.

Table 1: Detailed success rates of previous methods and AgentStore on OSWorld. Methods marked with “\*” represent our re-implementation of the corresponding agents. Additionally, due to the significant overlap between the OS and Workflow domains in the original division, we have merged these two domains into “OS\*”.

Agent	Base	Success Rate (%)									
		OS*	Calc	Impress	Writer	VLC	TB	Chrome	VSC	GIMP	AVG
CogAgent	GogVLM	1.60	2.17	0.00	4.35	6.53	0.00	2.17	0.00	0.00	1.32
MMAgent	GPT-4o	14.44	4.26	6.81	8.70	9.50	6.67	15.22	30.43	0.00	11.21
CRADLE	GPT-4o	8.00	0.00	4.65	8.70	6.53	0.00	8.70	0.00	38.46	7.81
Friday*	GPT-4o	15.20	25.50	0.00	21.73	0.00	0.00	0.00	17.39	15.38	11.11
Open-Inter*	GPT-4o	12.80	12.76	0.00	13.04	0.00	0.00	0.00	17.39	15.38	8.94
AgentStore(GT)	Hybrid	20.00	36.17	10.63	47.83	47.06	40.00	34.78	47.82	38.46	29.54
AgentStore(ICL)	Hybrid	9.60	0.00	2.13	4.34	35.29	33.33	30.43	30.43	15.38	13.55
AgentStore(FT)	Hybrid	8.80	27.65	4.26	13.04	41.17	40.00	34.78	8.60	15.38	17.34
AgentStore(AT)	Hybrid	13.86	31.91	8.51	39.13	47.06	40.00	32.61	39.13	30.77	23.85

Table 2: Performance comparison On OSworld-Multi.

Method	Agent Match	Subtask Acc	Execution Acc
MMAgent	-	-	6.93%
AgentStore(FT)	-	-	-
AgentStore(ICL)	24.75%	40.00%	9.90%
AgentStore(ICL*)	28.71%	51.72%	14.85%
AgentStore(AT)	36.63%	62.16%	22.77%

Additionally, we propose three metrics: Agent-Match, SubtaskAcc, and ExecutionAcc, which respectively measure multi-agent prediction accuracy, subtask decomposition accuracy, and execution success rate. Detailed benchmark constructions and metric descriptions are provided in Appendix F.

As shown in Table 2, single-agent MMAgent, which lacks Agent prediction and task decomposition, performs poorly on this benchmark. The Fine-Tuning method is not applicable in this scenario due to the infinite combinations of agents, making it impossible to pre-organize the necessary data for training. Moreover, while the ICL methods function to a certain extent, even with advanced commercial models (with ICL\* indicating the use of GPT-4o), the constraints of overly long contexts and vast combinatorial spaces result in subpar outcomes. In contrast, AgentToken leverages its inherent task awareness, significantly narrowing the scope to a few selected agents. It demonstrates excellent performance across all metrics.

#### 4.1.3 Mobile OS Assistant: APPAgent

We also employ the APPAgent (Yang et al., 2023) benchmark to validate that AgentStore can generalize to mobile OS platforms. It consists of 9 popular mobile applications, each serving distinct purposes

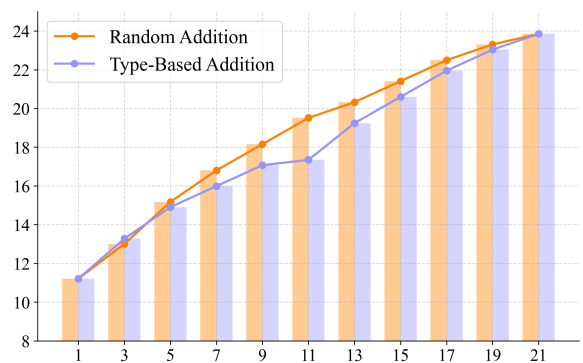


Figure 3: The performance curve as the number of agents increases, with the y-axis representing the success rate (%) on OSWorld and the horizontal x-axis representing the number of agents.

and collectively forming 45 tasks.

Table 3 compares the performance of AgentStore with a single general agent in (Yang et al., 2023). As shown, the performance of the generalist agent, lacking specific knowledge of each app, is subpar across many applications, even when utilizing the strongest base model. In contrast, AgentStore constructs dedicated agents tailored to their respective applications, effectively addressing performance deficiencies in certain apps and demonstrating a significant performance improvement from 26.7% to 57.8%. This underscores the applicability of the AgentStore concept to other OS platforms.

#### 4.2 Analysis of Agent Quantity and Diversity

To comprehensively analyze the advantages of scalable integration, we further explore the impact of the quantity and diversity of integrated agents within AgentStore on performance. To ensure thoroughness, we analyze AgentStore starting from a generalist MMAgent and incrementally add agents

Table 3: Success rates of generalist agents and AgentStore. Methods marked with “\*” indicate the re-implementation of the single agent in APPAgent without app-specific knowledge. *Due to differences between the original paper and the public benchmark, the results vary.* Additionally, while Appagent also generated app-specific agents, it did not integrate them, instead only evaluating individual apps, and thus it is not included in the comparison.

Agent	Base	Success Rate (%)									
		Maps	X	TG	Temu	YT	Spotify	Yelp	Gmail	Clock	AVG
APPAgent-Single*	Qwen-VL	20.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	20.0	4.4
APPAgent-Single*	GPT-4o	60.0	20.0	20.0	0.0	40.0	20.0	20.0	20.0	40.0	26.7
AgentStore(GT)	GPT-4o	80.0	60.0	40.0	40.0	60.0	80.0	80.0	60.0	60.0	66.7
AgentStore(AT)	GPT-4o	80.0	40.0	40.0	40.0	60.0	60.0	80.0	60.0	60.0	57.8

Table 4: Routing success rates of different strategies for enabling MetaAgent as the router.

Agent	Base	Success Rate (%)									
		OS	Calc	Impress	Writer	VLC	TB	Chrome	VSC	GIMP	AVG
ICL	GPT-4o	58.33	14.89	12.77	13.04	88.24	100	97.83	60.87	53.85	49.63
ICL	InternVL	37.50	6.38	21.28	8.70	35.29	33.33	52.17	30.43	30.77	41.57
FT-LoRA	InternVL	50.00	74.47	55.32	13.04	88.23	100	89.13	30.43	34.61	60.82
AgentToken	InternVL	75.00	80.85	72.34	43.47	100	100	95.65	91.30	73.08	80.60

Table 5: Efficiency comparison.

Method	Params	Memory	Time
FT-Full	7.78B	>80G	-
FT-Pt	86K	26G	-
FT-LoRA	38M	28G	2.5 hours
<b>AgentToken</b>	<b>86K</b>	<b>17G</b>	<b>0.2 hours</b>

in AgentPool to compare their effects.

We employ two strategies for adding agents: one involves randomly selecting agents to incrementally add to the AgentPool, while the other categorizes agents into GUI and CLI types, starting with one type before supplementing with the other. As shown in Figure 3, performance gradually increases with the growing number of agents, confirming the performance benefits of scalable integration within AgentStore. Additionally, we observe differences between the two strategies: random selection maintains a consistent mix of agent types, leading to a more stable growth. In contrast, adding agents of only one type causes the growth rate to slow over time, but this is mitigated when the other type is introduced (the x-axis reaches 11). This highlights the crucial role of agent diversity, demonstrating the importance of integrating heterogeneous agents.

### 4.3 Analysis of AgentToken

In this section, extensive experiments demonstrate the advantages of AgentToken in terms of effectiveness, efficiency, and low data requirements.

**Effectiveness** As shown in Table 4, ICL methods perform poorly as routers, even when using advanced models like GPT-4o. This confirms our assertion that relying on simple descriptions and few-shot demonstrations to master new agents can be challenging. In contrast, other tuning methods show some improvement by training on more task demonstrations. However, these methods are highly dependent on the quantity of data, while their overall performance improvement remains marginal. In comparison, our AgentToken overcomes these challenges, requiring only minimal data to efficiently train the corresponding tokens.

**Efficiency** In Table 5, we compared the efficiency of the AgentToken with other efficient-tuning methods, *i.e.*, prompt tuning (Pt) and adapter tuning (LoRA), focusing on the number of trainable parameters, memory requirements, and training time on the same A100 device. Results indicate that AgentToken is the most efficient across all dimensions, requiring the least amount of parameters and memory with the shortest training duration. Specifically, because AgentToken eliminates the need for gradients to flow through the main body of MLLM, training time is significantly reduced, and the process becomes more stable. Conversely, full fine-tuning and prompt tuning fail to converge properly due to their sensitivity to data.

**Data Requirement** Generally, the larger and higher-quality the demonstration set  $S_i$ , the more beneficial it is for training AgentToken. However,

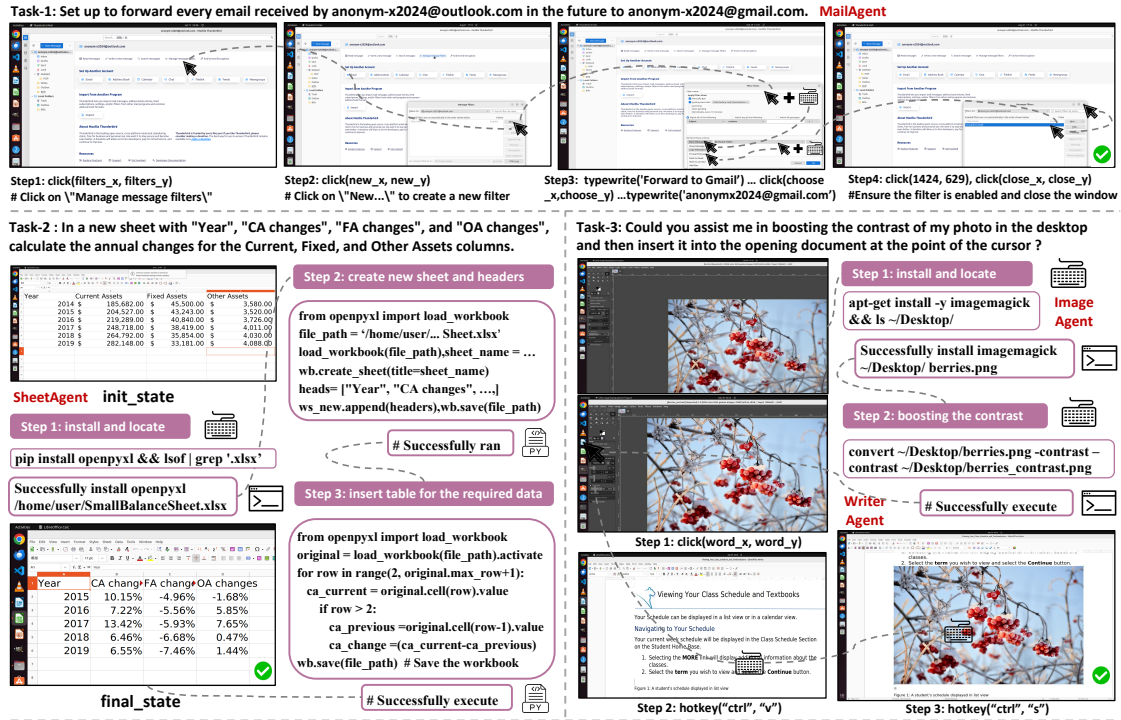


Figure 4: Specific steps involved in executing three tasks mentioned in the qualitative analysis.

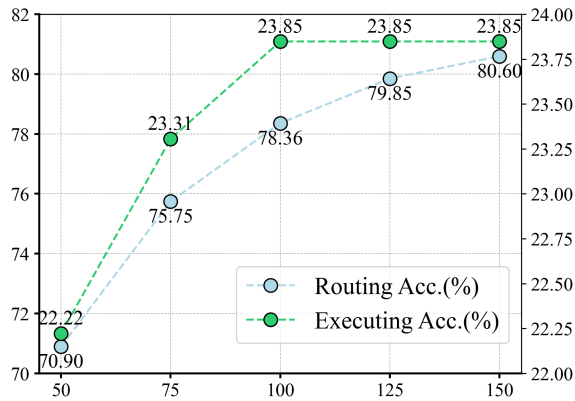


Figure 5: The accuracy curves with increasing training data corresponding to one agent. The x-axis represents the demonstration set size corresponding to each agent. The left y-axis represents the routing accuracy while the right y-axis indicates the executing accuracy.

in practical scenarios, manually acquiring a large volume of high-quality demonstrations poses significant challenges. The proposed automated process can mitigate this issue by generating data automatically; nevertheless, the scope of the generated data remains relatively limited (Shumailov et al., 2024). Fortunately, AgentToken can still be effectively trained due to its small parameter size and stable training process. As shown in Figure 5, when the demonstration set size reaches 100, a satisfactory accuracy rate can be achieved.

#### 4.4 Qualitative Analysis

In Figure 4, we provide running cases to illustrate how AgentStore enhances the overall system’s capability. In Task-1, the agent is tasked with setting up automatic email forwarding, which involves frequent GUI interactions and requires understanding of Thunderbird settings. MetaAgent assigns the specialized MailAgent to handle the task which navigates the exact steps to configure the forwarding settings. In particular, during the Step3, it executes a sequence of actions to accurately fill out the required forms and options, showcasing its advanced understanding capabilities within the mail domain. Similarly, in Task-2, which requires complex processing of a spreadsheet, MetaAgent selects the SheetAgent from the AgentPool to handle the task, avoiding overly complex GUI interactions. SheetAgent possesses knowledge of “openpyxl” and a deep understanding of the steps needed to manipulate sheets, efficiently completing this task. In addition, Task-3 illustrates a system-wide task that requires collaboration among multiple agents. MetaAgent successfully decomposes the task into subtasks and assigns the appropriate agents to complete each one. These cases highlight AgentStore’s specialized generalist abilities in handling both domain-specific and system-wide tasks.



## 5 Conclusion

In this paper, we introduce AgentStore, a flexible and scalable platform for dynamically integrating heterogeneous agents to independently or collaboratively complete OS tasks. Furthermore, we propose MetaAgent with the AgentToken strategy to achieve efficient management of the growing number of agents. Comprehensive quantitative analysis and qualitative results show that AgentStore expands the capabilities of existing agent systems in both generalization and specialization. We believe that as basic AGI models continue to evolve, AgentStore will integrate more powerful agents, progressively advancing toward the vision of building the specialized generalist computer assistant.

## 6 Acknowledgements

This work was supported by the National Nature Science Foundation of China (No. 62192781, No. 62272374), the Natural Science Foundation of Shaanxi Province (2024JC-JCQN-62), the National Nature Science Foundation of China (No. 62202367, No. 62250009), the Key Research and Development Project in Shaanxi Province No. 2023GXLH-024, Project of China Knowledge Center for Engineering Science and Technology, and Project of Chinese academy of engineering “The Online and Offline Mixed Educational Service System for ‘The Belt and Road’ Training in MOOC China”, and the K. C. Wong Education Foundation.

## Ethics Statement

This research focuses on building a scalable platform to integrate heterogeneous agents dynamically. The data datasets or benchmarks we employed are properly cited. There are no discrimination, bias, or fairness issues that need to be declared in this paper. Further, the outputs are not expected to be potentially harmful. To ensure reproducibility, we provide experimental details in Section 4 and their corresponding appendices. All source code will be made public.

## Limitations

While AgentStore has proven effective in multiple benchmarks, there still exist the following two directions for exploration: (1) Latency under high workloads: While task routing is efficient, the simultaneous deployment of multiple agents in high-demand scenarios can lead to increased response

times. Further optimization is required to minimize delays in such environments. (2) Scaling with larger AgentPool: The current framework demonstrates promising results with moderately sized agent pools, but further research is needed to enable seamless integration and management of significantly larger AgentPools. It is essential to explore more robust mechanisms that can dynamically allocate resources and resolve conflicts among a large number of agents. Additionally, evaluating the system on more complex and diverse benchmarks involving extensive multi-agent collaboration will be critical for validating its scalability and adaptability.

## References

- Kirti Aggarwal and Harsh K Verma. 2015. Hash\_rc6—variable length hash algorithm using rc6. In *2015 International Conference on Advances in Computer Engineering and Applications*, pages 450–456. IEEE.
- Ziwei Chai, Guoyin Wang, Jing Su, Tianjie Zhang, Xuanwen Huang, Xuwu Wang, Jingjing Xu, Jianbo Yuan, Hongxia Yang, Fei Wu, et al. 2024. An expert is worth one token: Synergizing multiple expert llms as generalist via expert token routing. *arXiv preprint arXiv:2403.16854*.
- Harrison Chase. 2022. [LangChain](#).
- Yibin Chen, Yifu Yuan, Zeyu Zhang, Yan Zheng, Jinyi Liu, Fei Ni, and Jianye Hao. 2024a. Sheetagent: A generalist agent for spreadsheet reasoning and manipulation via large language models. *arXiv preprint arXiv:2403.03636*.
- Zhe Chen, Weiyun Wang, Hao Tian, Shenglong Ye, Zhangwei Gao, Erfei Cui, Wenwen Tong, Kongzhi Hu, Jiapeng Luo, Zheng Ma, et al. 2024b. How far are we to gpt-4v? closing the gap to commercial multimodal models with open-source suites. *arXiv preprint arXiv:2404.16821*.
- Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. 2024. Seeclck: Harnessing gui grounding for advanced visual gui agents. *arXiv preprint arXiv:2401.10935*.
- Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2024. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36.
- Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.

- Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. 2023. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*.
- Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2024. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems*, 36.
- Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*.
- Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*.
- Wenyi Hong, Weihang Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. 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.
- Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for "mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008.
- Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and ZHAO-XIANG ZHANG. 2024. Sheetcopilot: Bringing software productivity to the next level through large language models. *Advances in Neural Information Processing Systems*, 36.
- Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2023. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization. *arXiv preprint arXiv:2310.02170*.
- Open-interpreter. 2024. [open-interpreter](#).
- OpenAI. 2023. [GPT-4 technical report](#). *CoRR*, abs/2303.08774.
- Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.
- Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy P. Lillicrap, and et al. 2024. [Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context](#). *CoRR*, abs/2403.05530.
- Ilya Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. 2024. [Ai models collapse when trained on recursively generated data](#). *Nature*, 631(8022):755–759.
- Qiushi Sun, Zhirui Chen, Fangzhi Xu, Kanzhi Cheng, Chang Ma, Zhangyue Yin, Jianing Wang, Chengcheng Han, Renyu Zhu, Shuai Yuan, et al. 2024. A survey of neural code intelligence: Paradigms, advances and beyond. *arXiv preprint arXiv:2403.14734*.
- Qiushi Sun, Zhangyue Yin, Xiang Li, Zhiyong Wu, Xipeng Qiu, and Lingpeng Kong. 2023. Corex: Pushing the boundaries of complex reasoning through multi-model collaboration. *arXiv preprint arXiv:2310.00280*.
- Weihao Tan, Ziluo Ding, Wentao Zhang, Boyu Li, Bohan Zhou, Junpeng Yue, Haochong Xia, Jiechuan Jiang, Longtao Zheng, Xinrun Xu, et al. 2024. Towards general computer control: A multimodal agent for red dead redemption ii as a case study. *arXiv preprint arXiv:2403.03186*.
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.
- Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. 2024. Open Devin: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*.
- Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023b. Self-instruct: Aligning language models with self-generated instructions. In *The 61st Annual Meeting Of The Association For Computational Linguistics*.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. AutoGen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.
- Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and

- Lingpeng Kong. 2024. [Os-copilot: Towards generalist computer agents with self-improvement](#). *Preprint*, arXiv:2402.07456.
- Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. [Os-world: Benchmarking multimodal agents for open-ended tasks in real computer environments](#). *Preprint*, arXiv:2404.07972.
- Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2023. [Appagent: Multimodal agents as smartphone users](#). *arXiv preprint arXiv:2312.13771*.
- Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. [Webshop: Towards scalable real-world web interaction with grounded language agents](#). *Advances in Neural Information Processing Systems*, 35:20744–20757.
- Kaiyan Zhang, Biqing Qi, and Bowen Zhou. 2024. [Towards building specialized generalist ai with system 1 and system 2 fusion](#). *arXiv preprint arXiv:2407.08642*.
- Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. [Bertscore: Evaluating text generation with bert](#). *arXiv preprint arXiv:1904.09675*.
- Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. 2023. [Webarena: A realistic web environment for building autonomous agents](#). *arXiv preprint arXiv:2307.13854*.

## A AgentPool

The AgentPool is a collection of all available agents within AgentStore. To build the prototype of AgentStore, we organized 20 agents within AgentPool, each with distinct functionalities. As shown in Table 6, these agents range from unimodal to multimodal, from open-source to closed-source models, and from Command-Line Interfaces (CLI) to Graphical User Interfaces (GUI). The diverse capabilities of these agents cover common applications and tasks in both daily life and professional settings. In addition to the domain-specific agents we developed, we also integrated existing agents, such as Friday (Wu et al., 2024) and (He et al., 2024). This demonstrates the scalability of our approach, which allows third-party agents to be added to the platform.

Specifically, for closed-source model agents, we uniformly use GPT-4o as the base model. For open-source model agents, single-modality agents are based on Llama 3.1 (Touvron et al., 2023), while multi-modality agents are built on InternVL2 (Chen et al., 2024b). The last column of Table 6 indicates whether the agent has the capability to solve tasks outside its own domain.

Figure 6 illustrates the distribution of different types of agents, showing that the initial version of AgentStore maintains a consistent balance between GUI and CLI agents. Most models also support extensions to handle additional tasks. Due to the significant gap between open-source and close-commercial models, most agents in this version are currently based on close-commercial models.

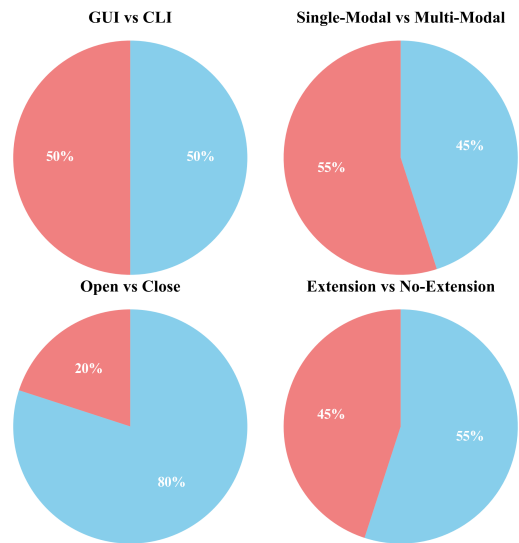


Figure 6: The agent distribution across different types.

Table 6: The presentation of agents in the AgentPool.

	CLI or GUI?	Single or Multi Modal?	Open or Close Base Model?	Domain for OSworld	Support Extension?
OSAgent	GUI	Multi	Close	OS	✓
Friday (Wu et al., 2024)	CLI	Single	Close	OS	✓
SheetAgent	CLI	Single	Close	Calc	✗
CalcAgent	GUI	Multi	Close	Calc	✓
SlideAgent	CLI	Single	Close	Impress	✗
ImPressAgent	GUI	Multi	Close	Impress	✓
WordAgent	CLI	Single	Close	Writer	✗
WriterAgent	GUI	Multi	Close	Writer	✓
VLCAgent	GUI	Multi	Close	VLC	✓
MailAgent	GUI	Multi	Close	TB	✓
ChromeAgent	GUI	Multi	Close	Chrome	✓
WebAgent (He et al., 2024)	GUI	Multi	Close	Chrome	✗
VSAgent	GUI	Multi	Open	VSC	✗
VSGUIAgent	CLI	Single	Close	VSC	✓
GimpAgent	GUI	Multi	Close	GIMP	✓
ImageAgent	CLI	Single	Open	GIMP	✓
Searcher	CLI	Single	Close	-	✗
GoogleDrive	CLI	Single	Close	-	✗
CoderAgent	CLI	Single	Open	-	✗
VisionAgent	CLI	Multi	Open	-	✗

**Integration Protocol:** When a developer creates a new OS agent and seeks to integrate it into AgentStore, it is essential to register the agent’s information in a standardized format. To ensure consistency in the integration process, we established an **agent integration protocol**. As shown in the template below, during enrollment, the developer completes a predefined form outlining the agent’s capabilities, limitations, the applications it interacts with, and demonstrations of its functionality.

In the actual enrollment process, we encourage developers to provide more demonstrations—the greater the number, the more comprehensive the document will be, which also facilitates agentToken training during the self-instruct process. In this paper, we provide 10 demonstrations for each agent, which is relatively lightweight but still effectively aids the Metaagent in learning and understanding the corresponding agent.

```
Template: AgentName

# Applications:
# List the applications or tools that the agent supports or interacts with.

# Capabilities
# Describe the main functions and abilities of the agent. Include details about the tasks it
can perform and the libraries or technologies it utilizes.

# Limitations
# Outline the constraints and tasks the agent cannot perform. This helps set clear boundaries
for the agent's functionality.

# Demonstrations

# Demostation_1: <Description of the first demonstration task.> <path_to_demonstration_image_1>
# Demostation_2: <Description of the second demonstration task.> <path_to_demonstration_image_2>
# Demostation_3: <Description of the third demonstration task.> <path_to_demonstration_image_3>
# Demostation_4: <Description of the fourth demonstration task.> <path_to_demonstration_image_4>
.....

End!
```

## B Details of Agents within AgentPool

Following the above template, we present six typical agent documents related to LibreOffice tasks to provide a clearer view of the agents in the AgentPool. Due to space limitations, further details on additional agents will be available when the entire project is open-sourced.

**SlideAgent and ImpressAgent:** These two agents serve distinct roles in slide processing tasks, with SlideAgent being CLI-based and ImpressAgent GUI-based. SlideAgent is derived through enhanced prompting of the FridayAgent (Wu et al., 2024), leveraging the Python-pptx library documentation. It effectively utilizes bash commands to locate file paths and generate Python code to perform operations on slides, such as adding titles or modifying text. In contrast, ImpressAgent specializes in handling tasks via GUI interactions with LibreOffice Impress, excelling in modifying software defaults, adjusting layout details, and performing other visual modifications that require direct interface manipulation.

**WordAgent and WriterAgent:** These two agents serve distinct roles in document processing tasks, with WriterAgent being CLI-based and WordAgent GUI-based. WriterAgent utilizes Python’s python-docx library to perform operations such as modifying paragraphs, adding or removing tables, and executing other programmatic edits on Word documents. In contrast, WordAgent specializes in handling GUI operations in LibreOffice Writer software, excelling in modifying software defaults, adjusting layout settings, and performing visually oriented tasks that require direct interaction with the user interface.

**SheetAgent and CalcGUI:** These two agents are designed for spreadsheet processing tasks, with SheetAgent being CLI-based and CalcGUI GUI-based. SheetAgent leverages Python’s openpyxl library to excel in tasks such as data entry, formula insertion, chart creation, and spreadsheet formatting, offering

programmatic efficiency for managing spreadsheets. On the other hand, CalcGUI specializes in handling GUI operations in LibreOffice Calc software, excelling in modifying software defaults, adjusting layout settings, and performing visually driven tasks that require direct interface manipulation.

AgentName: *SlideAgent*

**# Applications:**

Terminal, LibreOffice Impress

**# Capabilities**

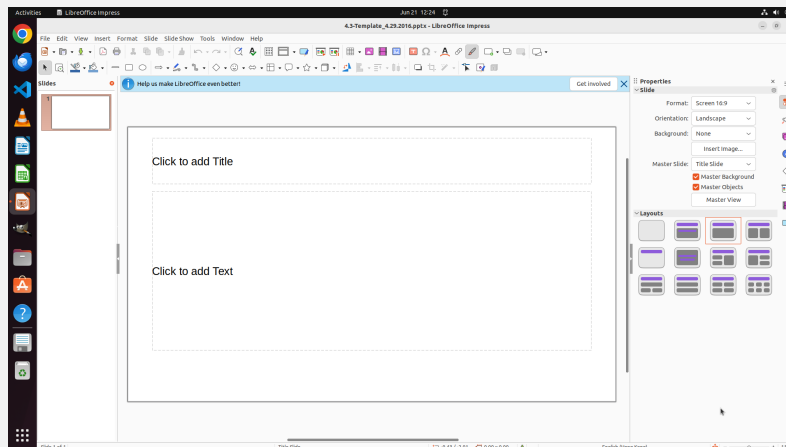
Specializes in creating and modifying PowerPoint presentations using Python's python-pptx library. It can handle tasks involving slide creation, layout management, text and content insertion, and formatting adjustments. Also capable of detecting open PowerPoint presentations using Bash commands.

**# Limitations**

Cannot handle GUI operations, cannot perform tasks outside the capabilities of the python-pptx library such as directly interacting with embedded videos and complex animations. Additionally, cannot modify LibreOffice Impress software defaults or preferences.

**# Demostations**

Demostation\_1: Can you add a new slide at the end of my presentation with the title 'Conclusion' and the text 'Thank you for your attention?'



Demostation\_2: Can you add a footer with text 'Company Confidential' to all slides in the current PowerPoint presentation?



.....  
**End!**

AgentName: *ImPressAgent*

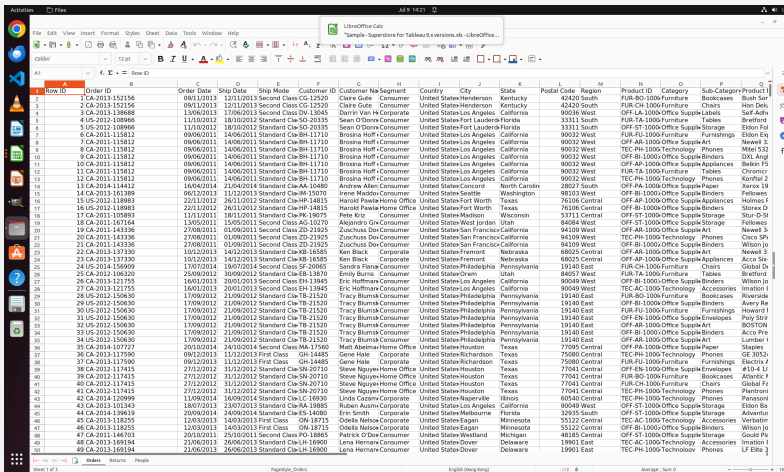
# Applications:  
LibreOffice Impress

# Capabilities  
Specializes in handling tasks using GUI operations and can modify LibreOffice Impress software defaults or preferences.

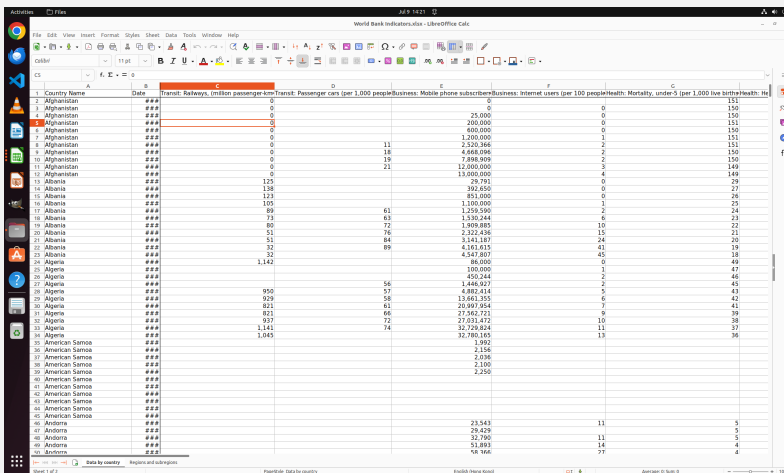
# Limitations  
Cannot handle complex tasks such as creating and modifying PowerPoint presentations using Python's python-pptx library.

# Demonstrations

Demonstration\_1: Enable the "Grid" view to help with precise placement of objects.



Demonstration\_2: Change the default font for all text in the presentation to "Helvetica".



End!

AgentName: *WordAgent*

**# Applications:**

Terminal, LibreOffice Writer

**# Capabilities**

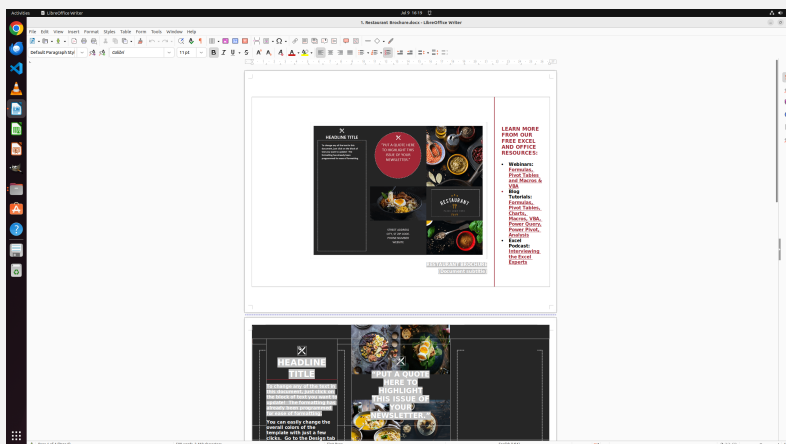
Excels at identifying and manipulating Word documents using Python's python-docx library. Can manage tasks involving document modification, data insertion, and formatting adjustments. Capable of detecting open Word or other documents using Bash commands.

**# Limitations**

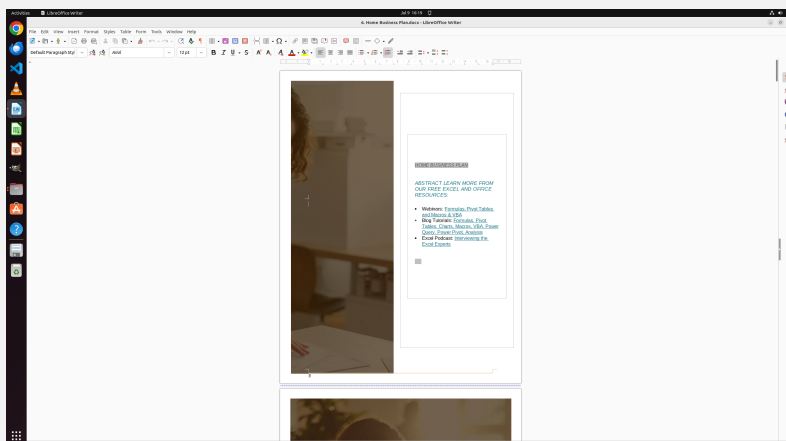
Cannot handle GUI operations, cannot perform tasks outside the capabilities of the python-docx library such as directly interacting with embedded media and scripts within the documents. Additionally, cannot modify LibreOffice Writer software defaults or preferences.

**# Demonstrations**

Demostation\_1: Add the text 'Grand Opening' as a title at the beginning of the document."



Demostation\_2: Insert a horizontal line above the 'ABSTRACT' heading.



.....

End!



AgentName: *WriterAgent*

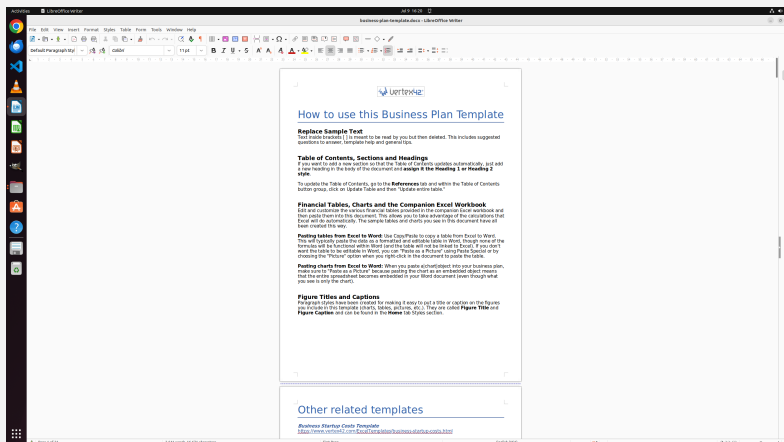
# Applications:  
LibreOffice Writer

# Capabilities  
Specializes in handling GUI operations and can perform tasks outside the capabilities of the python-docx library such as directly interacting with embedded media and scripts within documents. Can modify LibreOffice Writer software defaults or preferences.

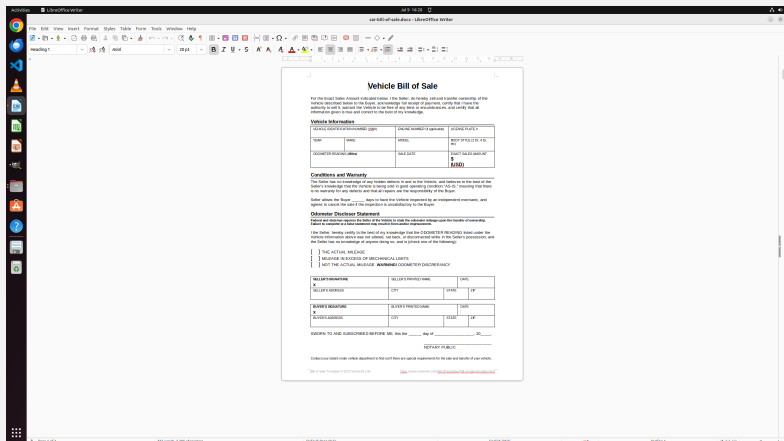
# Limitations  
Cannot identify and manipulate Word documents using Python's python-docx library, and cannot manage tasks involving document modification, data insertion, and formatting adjustments. Additionally, cannot detect open Word or other documents using Bash commands.

# Demonstrations

Demonstration\_1: Enable the "Show Changes" feature to track document edits location.



Demonstration\_2: Create a custom keyboard shortcut for "Print" set to Ctrl+P.



.....  
**End!**

AgentName: *SheetAgent*

### # Applications:

Terminal, LibreOffice Calc

### # Capabilities

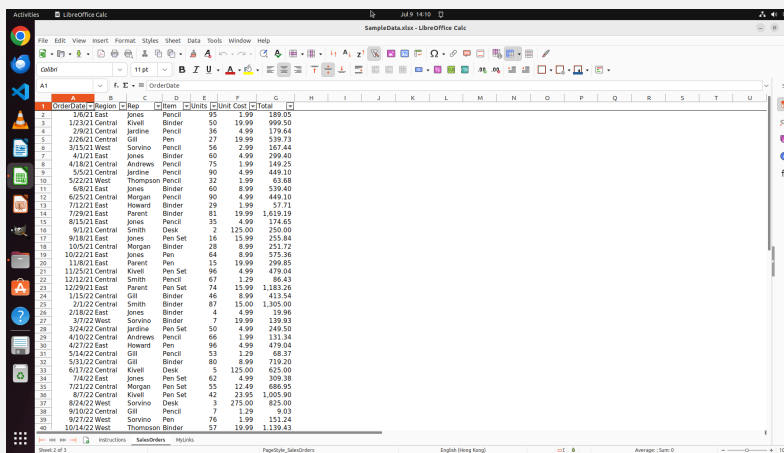
Specializes in creating, analyzing, and modifying Excel spreadsheets using Python's openpyxl library. It can handle tasks involving data entry, formula insertion, chart creation, and spreadsheet formatting. Also capable of detecting open Excel files using Bash commands.

### # Limitations

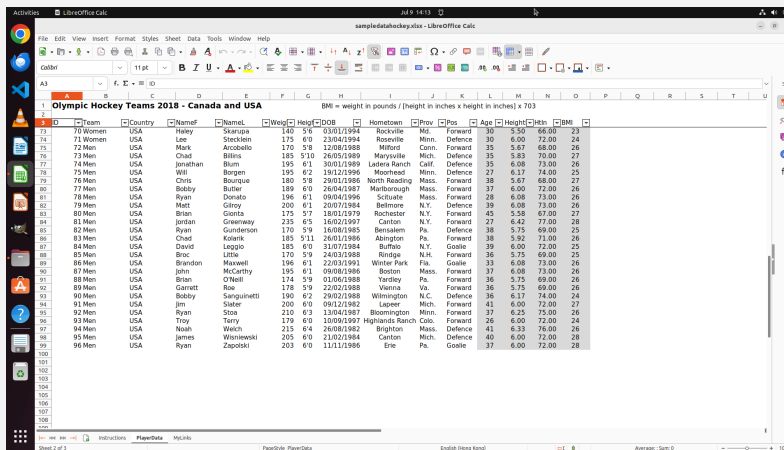
Cannot handle GUI operations, cannot perform tasks outside the capabilities of the openpyxl library such as directly interacting with complex macros. Additionally, cannot modify LibreOffice Calc software defaults or preferences.

### # Demonstrations

Demostation\_1: Highlight rows where the total sales exceed \$1000.



Demostation\_2: Filter out players older than 35 and list their names and ages in a new sheet named "Veteran Players".



End!

AgentName: CalcGUI

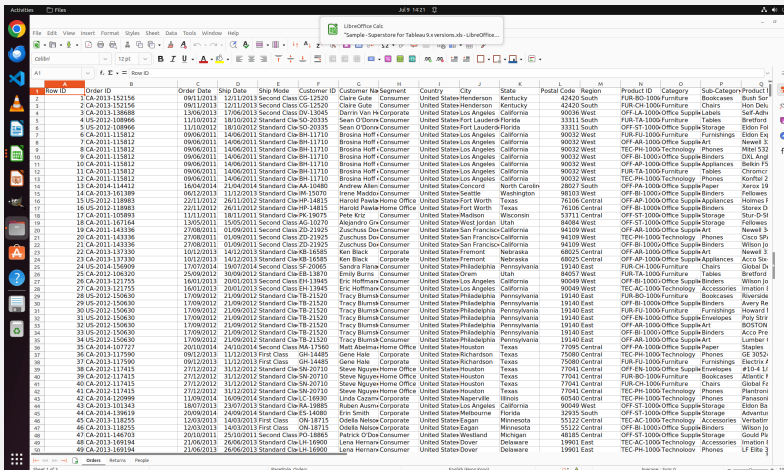
# Applications:  
LibreOffice Calc

# Capabilities  
Specializes in handling tasks using GUI operations and can modify LibreOffice Calc software defaults or preferences.

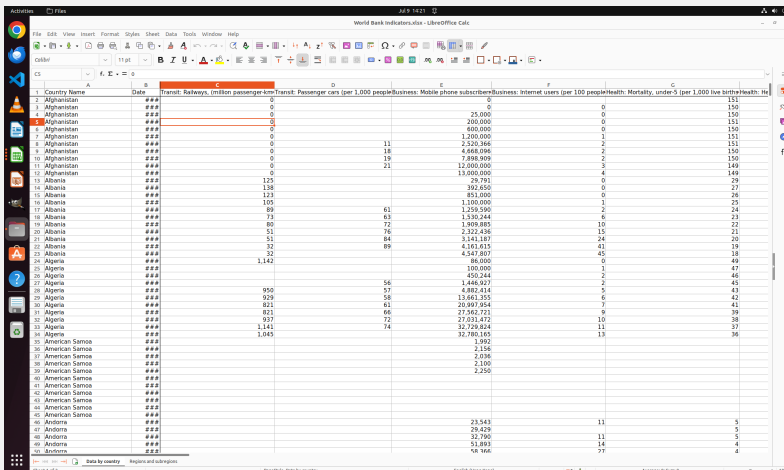
# Limitations  
Cannot handle complex tasks such as creating, analyzing, and modifying Excel spreadsheets using Python's openpyxl library.

# Demonstrations

Demostation\_1: Set the row height to 18 pixels for better readability.



Demostation\_2: Filter the data to show only rows where "Health: Mortality, under-5" is greater than 50.



End!

## C Cost analysis

While AgentStore introduces some computational overhead during deployment, its architecture demonstrates notable advantages in reducing task execution time and cost through the efficient assignment of specialized agents. The comparative analysis of AgentStore and MMAgent highlights these benefits, as summarized in Table 7. Due to potential variations in API response times influenced by network conditions, the reported running time and cost should be considered approximate estimates.

Table 7: The running times and costs between MMAgent and AgentStore.

Method	Base-model	running time	Budget Cost
MMAgent (Xie et al., 2024)	GPT-4o	$\approx 12\text{h}$	$\approx 250\text{\$}$
AgentStore(AT)	GPT-4o+Llama 3.1+InternVL2	$\approx 7\text{h}$	$\approx 100\text{\$}$

Such improvements are attributable to the following factors:

(1) **Task-Specialized Assignment:** By employing a diverse pool of agents, AgentStore allocates tasks to agents best equipped for specific domains, minimizing redundant or inefficient processing steps.

(2) **Optimized Collaboration Framework:** The MetaAgent’s AgentToken strategy enables effective coordination among agents, ensuring tasks are completed collaboratively where appropriate, without unnecessary computational duplication.

(3) **Model Efficiency:** The inclusion of open-source models, such as Llama 3.1 and InternVL2, reduces dependence on commercial models like GPT-4o for tasks.

These findings underscore the potential of AgentStore to serve as a cost-effective and time-efficient platform for automating diverse tasks in operating system environments.

## D Automated process with self-instruct

In this section, we provide more details about the Automated data generation process, including threshold selection and the greedy filtering algorithm.

**Threshold Selection** To ensure the reliability of threshold selection, we first studied the distribution of thresholds in real-world tasks based on human-labeled standards. As shown in Figure 7, in tasks labeled by OSworld, the 95% threshold distribution of BertScore across different domains is primarily concentrated between 0.77 and 0.92. Therefore, to further strictly control the quality of generated data, we ultimately selected a threshold of 0.8 for  $\tau_1$  and 0.9 for  $\tau_2$  to filter the data.

This approach offers several advantages. By selecting thresholds of 0.8 for  $\tau_1$  and 0.9 for  $\tau_2$ , we strike a balance between retaining high-quality data and ensuring the diversity necessary for robust training. The  $\tau_1$  threshold helps in eliminating low-quality samples, while  $\tau_2$  enforces stricter criteria for the final selection of data, ensuring that only the most relevant and high-quality data points are used. This dual-threshold filtering process not only improves the precision of the generated data but also enhances the overall performance of agent training, reducing the risk of overfitting to noise or irrelevant tasks.

**Greedy Filtering Algorithm** Algorithm 1 presents a greedy algorithm for filtering a set of newly generated demonstrations,  $S'_i$ , ensuring that each selected demonstration maintains a BERTScore similarity within the specified bounds  $\tau_1$  and  $\tau_2$  relative to both existing demonstrations  $S_i$  and previously selected new demonstrations  $S_i^{new}$ . The key improvement lies in the prioritization of demonstrations that are optimally positioned between the two thresholds, thereby enhancing both relevance and diversity.

A prioritization mechanism selects demonstrations optimally positioned between the similarity thresholds. By calculating the minimum distance of each candidate’s BERTScore to the thresholds, the algorithm ensures that selected demonstrations are neither too similar nor too dissimilar to existing ones. This strategic ordering facilitates the inclusion of the most appropriate demonstrations first, thereby maximizing both the relevance and diversity of the refined set  $S_i^{new}$ . Consequently, the quality of the training data for AgentToken is significantly improved, fostering more effective training outcomes.

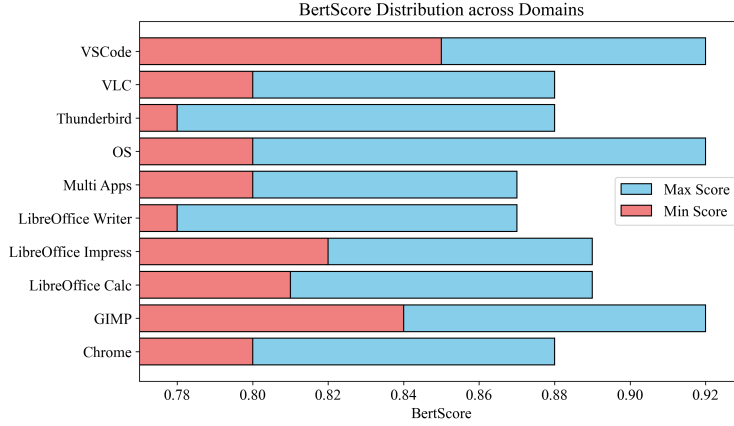


Figure 7: BertScore distribution across different domains.

---

**Algorithm 1** Greedy Filtering of Generated Demonstrations using BERTScore with Prioritized Selection

---

**Require:** •  $S'_i = \{y'_1, y'_2, \dots, y'_m\}$ : Set of newly generated demonstrations

- $S_i = \{y_1, y_2, \dots, y_n\}$ : Existing set of demonstrations
- $\tau_1$ : Lower bound for BERTScore similarity
- $\tau_2$ : Upper bound for BERTScore similarity

**Ensure:** •  $S_i^{new}$ : Refined set of new demonstrations satisfying the similarity constraints

- 1: Initialize  $S_i^{new} \leftarrow \emptyset$
- 2: For each  $y' \in S'_i$ , compute the minimum distance to the thresholds:

$$d(y') = \min(|\text{BERTScore}(y', y) - \tau_1|, |\text{BERTScore}(y', y) - \tau_2|) \quad \forall y \in S_i$$

- 3: Sort  $S'_i$  in descending order based on  $d(y')$
  - 4: **for** each  $y' \in S'_i$  in sorted order **do**
  - 5:     Initialize a flag  $valid \leftarrow \text{True}$
  - 6:     **for** each  $y \in S_i \cup S_i^{new}$  **do**
  - 7:         Compute  $\text{BERTScore}(y', y)$
  - 8:         **if**  $\text{BERTScore}(y', y) < \tau_1$  **or**  $\text{BERTScore}(y', y) > \tau_2$  **then**
  - 9:              $valid \leftarrow \text{False}$
  - 10:         **break**
  - 11:         **end if**
  - 12:     **end for**
  - 13:     **if**  $valid$  **then**
  - 14:         Add  $y'$  to  $S_i^{new}$
  - 15:     **end if**
  - 16: **end for**
  - 17: **return**  $S_i^{new}$
- 

## E OSWorld

OSWorld (Xie et al., 2024) is a scalable, computer environment designed for multimodal agents. This platform provides a real-world environment for assessing open-ended computer tasks involving various applications. In this section, we provide a detailed introduction to OSworld, focusing on three key aspects: the open-ended and diverse nature of tasks, the reliability of evaluations in real-world environments, and the varied capability requirements for agents. This aims to help readers understand the rationale behind

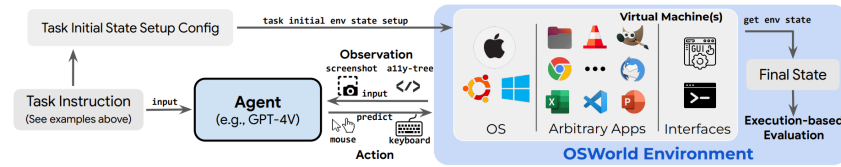


Figure 9: OSWorld can serve as a unified environment for evaluating *open-ended* computer tasks in the real-world computer environment.

using OSworld as the primary evaluation platform in the main text.

### E.1 OSWorld Tasks

OSWorld is a benchmark suite consisting of 369 real-world computer tasks, primarily based in an Ubuntu Linux environment, with a smaller portion covering Microsoft Windows. The tasks are sourced from the authors as well as various platforms like forums, tutorials, and guidelines. Each task is paired with a natural language instruction and a hand-crafted evaluation script for scoring. Figure 8 provides a detailed classification of tasks, showcasing their diversity and effectively reflecting the nature of open-ended tasks in real-world scenarios.

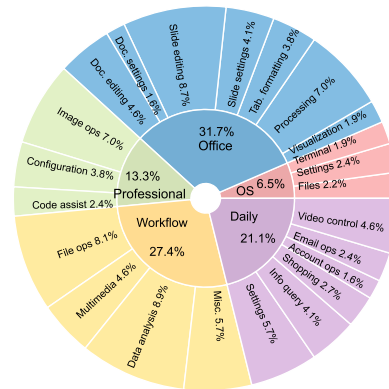


Figure 8: Task instructions distribution in OSWorld (Xie et al., 2024)

### E.2 Real-world Computer Environment

As shown in Figure 9, OSworld provides an executable and controllable environment that supports task initialization, execution-based evaluation, and interactive agent learning in a range of *real* operating systems. It also provides easily accessible system screenshots, ally-tree information, and interfaces that facilitate agent output for mouse and keyboard operations. This rich system information, real-time execution, and comprehensive task evaluation offer an interactive environment that is not limited to specific applications or domains.

### E.3 Representative Examples

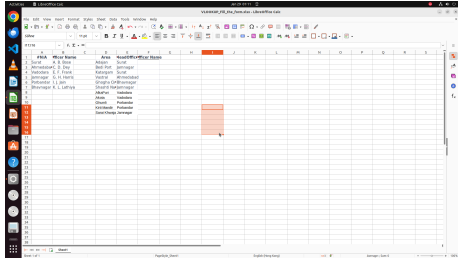
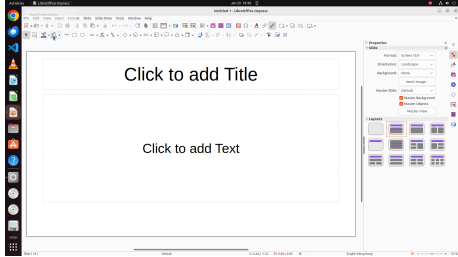
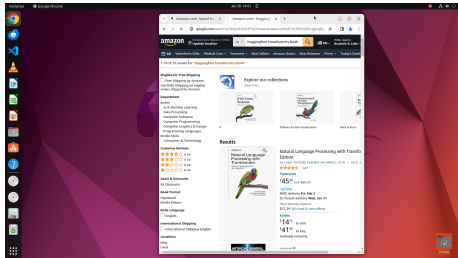
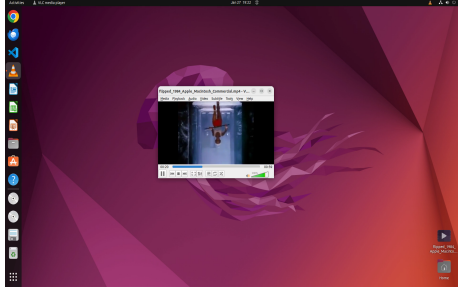
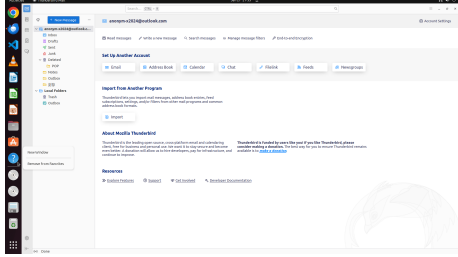
In Table 8, we present several representative examples from OSworld, which aim to illustrate the distinct operational logic involved in different tasks and the diverse capabilities required. These examples help readers better understand the broad range of generalization and specialized skills necessary in real-world computer environments, which are challenging for a single agent to fully encompass.

Table 8: Representative Examples from OSWorld to illustrate the distinct operational logic and the diverse capabilities involved in different tasks.

Related App(s)	Instruction(s)	Screenshot	Abilities Needed
OS	<i>I want to install Spotify on my current system. Could you please help me?</i>		specialized knowledge of OS; Proficient GUI operations

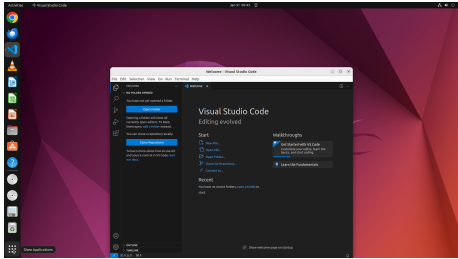
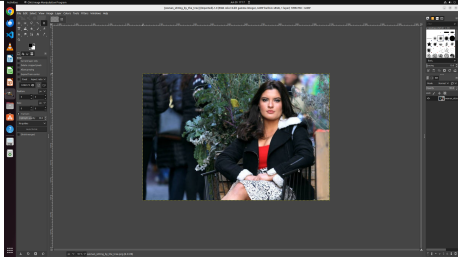
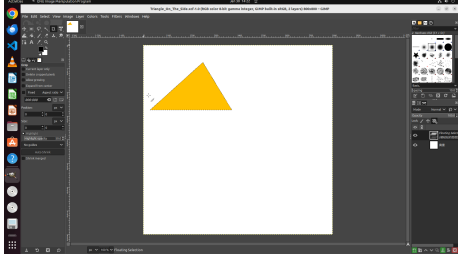
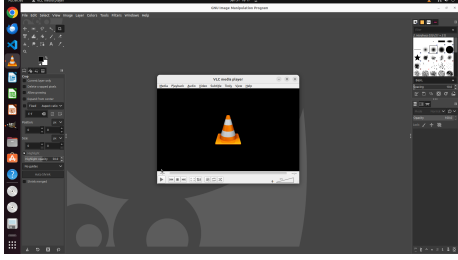
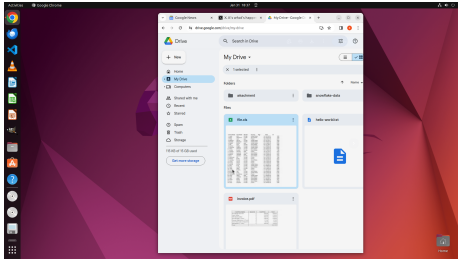
*Continued on next page*

Table 8 – continued from previous page

Related App(s)	Task Instruction	Screenshot of Initial State	Abilities Needed
Calc	<i>I have a lookup table for the officers of each branch. Please, here is another table in which I need to fill with the officer names according the headoffice (i.e., the branch name). Help me to complete this.</i>		Familiarity with the openpyxl library and command-line proficiency
Impress	<i>I closed the slide pannel on the left and idk how to get it back please help</i>		specialized knowledge of Slide software; imagine about UI layouts; Proficient GUI operations
Chrome	<i>Can you help me clean up my computer by getting rid of all the tracking things that Amazon might have saved? I want to make sure my browsing is private and those sites don't remember me.</i>		specialized knowledge of Chrome browser, proficient GUI operations
VLC	<i>Hey, could you turn this video the right way up for me? And once it's flipped around, could you save it for me with the name '1984_Apple.mp4' on the main screen where all my files are?</i>		software knowledge; spatial judgment ability
Thunderbird	<i>Create a local folder called "Promotions" and create a filter to auto move the inbox emails whose subject contains "discount" to the new folder</i>		Knowledge of the Thunderbird mail system; GUI operations

Continued on next page

Table 8 – continued from previous page

Related App(s)	Task Instruction	Screenshot of Initial State	Abilities Needed
VS Code	<i>Please modify VS Code's settings to disable error reporting for Python missing imports.</i>		software knowledge to deal with settings; reasoning to understand the cause and solution of the error
GIMP	<i>Could you tone down the brightness of my photo?</i>		Proficiency in using ImageMagick and CLI operations
GIMP	<i>Help me choose the yellow triangle and position it at the center of my picture.</i>		spatial perception and reasoning, as well as precise control of actions
Multiple (VLC+GIMP)	<i>Could you help me create an Animated GIF from a video file using VLC and GIMP from the source of video "src.mp4", 5-second clip beginning at 00:03?</i>		specialized software knowledge; ability to process multi-step procedure successfully
Multiple (Chrome+Calc)	<i>Could you help me extract data in the table from a new invoice uploaded to my Google Drive, then export it to a Libreoffice calc .xlsx file in the desktop?</i>		specialized ability to do table data; ability to process multi-step procedure successfully



## F OSWorld-Multi Benchmark

Building on OSworld, we further developed a new benchmark, **OSWorld-Multi**, to evaluate MetaAgent’s ability to predict and coordinate multiple agents for collaborative task execution. OSWorld-Multi consists of 101 tasks, each requiring collaboration with paired agents from the AgentPool. In the following sections, we will introduce the construction process, task examples, and evaluation metrics.

**Construction process** To maximize the reuse of tasks, system states, and evaluation functions from OSworld, we adopted a reverse synthesis approach. By mining paired examples in OSWorld, we generated tasks requiring agent collaboration. Specifically, we first traversed all pairwise combinations of subtasks, applying a two-step validation process: an initial filtering with a large language model (LLM), followed by manual review. This method allowed us to select meaningful collaborative tasks. Moreover, this approach enabled the synthesis of tasks requiring not only two-agent collaboration but also those involving three or more agents. In the following section, we will present some of the generated collaborative task results to demonstrate the outcomes of this synthesis process.

**Task examples** In the table below, we present several synthesized examples to help readers understand the generation process. Another advantage of this reverse synthesis approach is the presence of natural ground truth, allowing us to evaluate not only execution accuracy but also the accuracy of agent predictions and task decomposition. This enables a comprehensive assessment of collaborative task execution. In the following sections, we will provide a detailed explanation of the corresponding evaluation metrics.

### Synthesis task 1

# Agent:Subtask-1

VLCAgent: Snap a photo of the current video scene, save it as 'interstellar.png', and put it on the Desktop, please.

# Agent:Subtask-2

WriterAgent: Add page number for every page at the bottom left.

# Synthesis task

Capture a scene from a video in VLC and insert the image into a LibreOffice document with a page number.

# Required: VLCAgent + WriterAgent

### Synthesis task 2

# Agent:Subtask-1

VLCAgent: Help me modify the folder used to store my recordings to Desktop.

# Agent:Subtask-2

Friday: Change the permission of all regular files under current directory tree to 644.

# Synthesis task

Modify VLC's recording folder to Desktop and set file permissions to 644 for all files in this directory.

# Required: VLCAgent + Friday

### Synthesis task 3

# Agent:Subtask-1

VLCAgent: Can you start streaming the video from this link for me?  
<https://www.youtube.com/watch?v=pgBsyTKAwLw>

# Agent:Subtask-2

ChromeGUI: Could you help me clear browsing history from Youtube?

# Synthesis task

Could you stream a video from a YouTube link in VLC and clear all YouTube browsing history in Chrome after to ensure a clean search experience?

# Required: VLCAgent + ChromeGUI

### Synthesis task .....

.....

**Evaluation metrics** We propose three metrics for evaluation: AgentMatch, SubtaskAcc, and ExecutionAcc, which respectively measure multi-agent prediction accuracy, subtask decomposition accuracy, and execution success rate.

**AgentMatch** is designed to assess the accuracy of the agent prediction process during collaborative task execution. It compares the predicted set of agents selected by the MetaAgent with the ground truth set of agents that are required for successful task completion. Essentially, AgentMatch measures how well the MetaAgent can correctly identify the appropriate agents from the AgentPool for a given task. The metric is computed by calculating the accuracy of the predicted agent set relative to the actual agents involved in the task. Specifically, it checks whether the predicted agents match the expected agents. A high AgentMatch score indicates that the MetaAgent is effectively coordinating and predicting the correct agents for task execution.

**SubtaskAcc** is an evaluation metric that measures the accuracy of task decomposition by comparing the predicted subtasks assigned to each agent with the ground truth subtasks. It evaluates how well the MetaAgent decomposes a given task and assigns the correct subtasks to the respective agents. To assess SubtaskAcc, we use a textual comparison between the predicted subtasks and the actual subtasks for the same agent. This comparison is based on textual similarity, using BERTScore as the evaluation metric. As per our analysis in [D](#), if the BERTScore is below 0.77, the two subtasks are considered too dissimilar, and the decomposition is deemed unsuccessful. Conversely, if the BERTScore exceeds this threshold, the decomposition is considered accurate. This threshold ensures that only decompositions with sufficiently high textual similarity are counted as correct. SubtaskAcc thus reflects how effectively the MetaAgent can break down a complex task and allocate the correct components to individual agents. A high SubtaskAcc score indicates that the MetaAgent is accurately identifying the required subtasks for each agent, contributing to the overall success of the collaborative task execution.

**ExecutionAcc** is an evaluation metric that measures the success rate of task execution by reusing the assessment methods from OSworld. This metric focuses on determining whether the predicted subtasks are correctly executed by the agents, based on their final state in the environment.

To evaluate ExecutionAcc, we rely on OSworld's system of getter and evaluator functions. The getter function extracts key components from the final state of the environment (e.g., a modified file or text contents displayed in a window element), while the evaluator function assesses success based on these extracted components. If a necessary function does not exist, it is constructed and added to the function library of the environment. Each task is evaluated by comparing its final execution state with the expected outcome, and the evaluation process is designed to be robust.

In the context of our system, ExecutionAcc provides a direct measure of how successfully the agents complete their assigned tasks, reflecting the practical performance of task execution in real-world scenarios. A high ExecutionAcc indicates that the agents are accurately following the predicted subtasks and completing them correctly in the environment.

## G Prompt Details

We provide examples of MetaAgent prompts in different modes to help readers understand the inference process. It is important to note that in manager mode, the prompt templates in Section G.3 for AgentToken and ICL are identical. The key difference is that AgentToken reduces the number of input documents, effectively shortening the context length, which in turn improves performance.

Additional prompts, including those related to each individual agent and self-instruct, will be provided when the project is open-sourced.

### G.1 Prompt for router mode for AgentToken

Prompt: *Router for AgentToken*

```
Imagine you have a complex task that needs to be executed on an operating system.
This task can be decomposed into sub-tasks corresponding to the model's capabilities.
You have several agents with different specializations available.
Requirements:
The task is assigned to one agent, the model should return the one token of that agent.
Now your task is: {task_name}
```

### G.2 Prompt for router mode for ICL

Prompt: *Router for ICL*

```
Imagine you have a complex task that needs to be executed on an operating system.
This task can be decomposed into sub-tasks corresponding to the model's capabilities.
You have several agents with different specializations available.
{agent_1_document},{agent_2_document},...{agent_n_document}
Requirements:
The task is assigned to one agent, the model should return the name of that agent.
like:
###CalcAgent###
Now your task is: {task_name}
```

### G.3 Prompt for manager mode

Prompt: *Manager Mode*

```
Imagine you have a complex task that needs to be executed on an operating system.
This task can be decomposed into sub-tasks corresponding to the model's capabilities.
You have agents with different specializations available:
{agent_1_document},{agent_2_document},...{agent_n_document}
Requirements:
The task requires multiple agents, the model should specify which sub-tasks each agent should
handle.
The model should ensure that the task assignment optimizes efficiency and effectiveness,
considering the unique capabilities of each agent.
return like:
###AgentName1:compute the sum of data in a new sheet.###
###AgentName2:upload the computed file to the google Drive###
Be careful not to assign the same agent to perform tasks consecutively.
don't return like this:
###Agent1:compute the sum of data in a new sheet.###
###Agent1:rename this sheet.###
Now your task is: {task_name}
```