AppAgentX: Evolving GUI Agents as Proficient Smartphone Users

AppAgentX: 作为熟练智能手机用户的进化图形界面代理

Wenjia Jiang [1,2] , Yangyang Zhuang [1] , Chenxi Song [1] , Xu Yang [3] , Joey Tianyi Zhou [4,5] Chi Zhang [1] ,

姜文佳 [1,2] ， 庄洋洋 [1] ， 宋晨曦 [1] ， 杨旭 [3] ， 周天翼 [4,5] ， 张驰 [1] ，

[1] Westlake University, China, [2] Henan University, China, [3] Southeast University, China

[1] 中国西湖大学， [2] 中国河南大学， [3] 中国东南大学

[4] IHPC, Agency for Science, Technology and Research, Singapore

[4] 新加坡科学技术研究局 (Agency for Science, Technology and Research， 简称 IHPC)

[5] CFAR, Agency for Science, Technology and Research, Singapore

[5] 新加坡科学技术研究局 (Agency for Science, Technology and Research, CFAR)

{jiangwenjia, chizhang}@westlake.edu.cn,

{jiangwenjia, chizhang}@westlake.edu.cn,

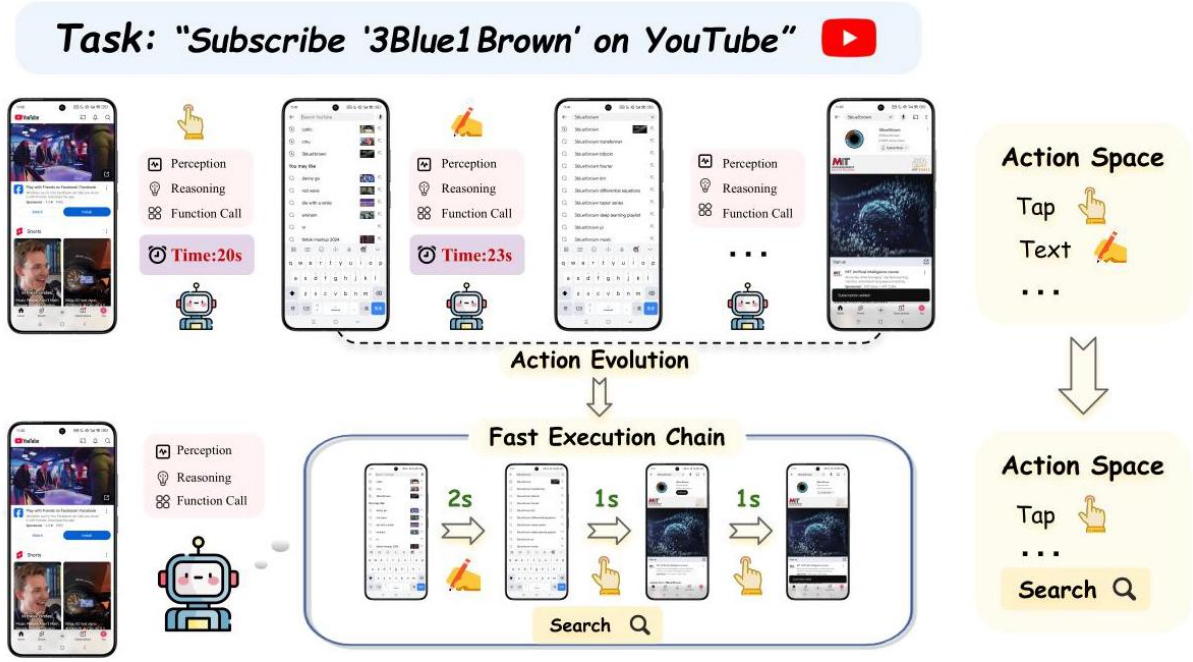

Figure 1: Illustration of the proposed evolutionary mechanism in GUI agents. The agent evolves a high-level action, ”Search”, which replaces a sequence of inefficient low-level actions. This evolution eliminates the need for step-by-step reasoning, significantly enhancing the agent’s efficiency.

图 1: 所提进化机制在图形用户界面 (GUI) 代理中的示意图。代理进化出一个高级动作“搜索”，取代了一系列低效的低级动作。该进化消除了逐步推理的需求，显著提升了代理的效率。

Abstract

摘要

Recent advancements in Large Language Models (LLMs) have led to the development of intelligent LLM-based agents capable of interacting with graphical user interfaces (GUIs). These agents demonstrate strong reasoning and adaptability, enabling them to perform complex tasks that traditionally required predefined rules. However, the reliance on step-by-step reasoning in LLM-based agents often results in inefficiencies, particularly for routine tasks. In contrast, traditional rule-based systems excel in efficiency but lack the intelligence and flexibility to adapt to novel scenarios. To address this challenge, we propose a novel evolutionary framework for GUI agents that enhances operational efficiency while retaining intelligence and flexibility. Our approach incorporates a memory mechanism that records the agent's task execution history. By analyzing this history, the agent identifies repetitive action sequences and evolves high-level actions that act as shortcuts, replacing these low-level operations and improving efficiency. This allows the

近年来大型语言模型 (LLMs) 的进展催生了能够与图形用户界面 (GUI) 交互的智能基于 LLM 的代理。这些代理展现出强大的推理能力和适应性，使其能够执行传统上依赖预定义规则的复杂任务。然而，基于 LLM 的代理依赖逐步推理，常导致效率低下，尤其是在处理常规任务时。相比之下，传统的基于规则系统在效率上表现优异，但缺乏智能性和灵活性，难以适应新颖场景。为解决这一挑战，我们提出了一种新颖的 GUI 代理进化框架，旨在提升操作效率的同时保持智能性和灵活性。我们的方法引入了记忆机制，记录代理的任务执行历史。通过分析这些历史，代理识别重复的动作序列，并进化出作为捷径的高级动作，以替代这些低级操作，从而提升效率。这使得

agent to focus on tasks requiring more complex reasoning, while simplifying routine actions. Experimental results on multiple benchmark tasks demonstrate that our approach significantly outperforms existing methods in both efficiency and accuracy. The code will be open-sourced to support further research.

代理专注于需要更复杂推理的任务，同时简化常规操作。在多个基准任务上的实验结果表明，我们的方法在效率和准确性方面均显著优于现有方法。代码将开源以支持进一步研究。

1 Introduction

1 引言

Recent advancements in artificial intelligence have been significantly influenced by the development of LLMs such as GPT-4 (OpenAI, 2024) and DeepSeek-V3 (DeepSeek-AI, 2024). These models, renowned for their ability to process and generate human-like text, have catalyzed innovations across various domains, including natural language understanding, generation, and reasoning. One promising application is the creation of LLM-based agents, which use natural language to autonomously interact with users and perform a wide range of tasks. Unlike traditional rule-based systems, LLM-based agents exhibit the flexibility to

近年来，人工智能领域的重大进展在很大程度上得益于大型语言模型 (LLMs) 的发展，如 GPT-4(OpenAI，2024) 和 DeepSeek-V3(DeepSeek-AI，2024)。这些模型以其处理和生成类人文本的能力著称，推动了自然语言理解、生成和推理等多个领域的创新。其中一个有前景的应用是基于大型语言模型的智能代理，这些代理利用自然语言自主与用户交互并执行各种任务。与传统的基于规则的系统不同，基于大型语言模型的智能代理展现出灵活性，能够

understand complex tasks and generalize to novel scenarios, enhancing human-computer interaction.

理解复杂任务并推广到新颖场景，提升人机交互体验。

A notable development in this area is the emergence of LLM-based agents capable of operating graphical user interfaces (GUIs). Unlike Robotic Process Automation (RPA), which relies on predefined rules, these agents can engage with GUIs dynamically, mimicking human-like interactions by outputting low-level actions such as clicks, drags, and scrolls. This approach not only increases operational flexibility but also enables the execution of tasks that require adaptability and reasoning, without needing access to backend systems or APIs.

该领域的一个显著进展是基于大型语言模型 (LLM) 的代理能够操作图形用户界面 (GUI)。不同于依赖预定义规则的机器人流程自动化 (RPA)，这些代理能够动态地与 GUI 交互，通过输出点击、拖动和滚动等低级操作，模拟人类的交互方式。这种方法不仅提升了操作的灵活性，还使得执行需要适应性和推理能力的任务成为可能，无需访问后端系统或 API。

To achieve optimal performance, various mechanisms have been introduced to guide the LLMs in generating accurate and contextually appropriate actions. Techniques such as reflection(Huang et al., 2022) and chain-of-thought(Wei et al., 2023) reasoning have been employed to help the model carefully consider the implications of each action before execution. While these methods can significantly enhance the agent's decision-making abilities, they often come at the cost of efficiency, particularly for tasks that do not require advanced reasoning. For example, as illustrated in Figure 1, during simple tasks like search operations, the agent may need to reason each step, such as clicking the search box, typing text, and pressing submit, leading to unnecessary delays. While traditional RPA methods can execute these fixed steps rapidly, they lack the flexibility to handle tasks requiring intelligent judgment. This highlights the need for a more efficient, adaptable approach to automate routine tasks.

为了实现最佳性能，引入了多种机制来指导大型语言模型 (LLMs) 生成准确且符合上下文的操作。诸如反思 (Huang et al., 2022) 和链式思维 (Wei et al., 2023) 推理等技术被用来帮助模型在执行前仔细考虑每个操作的影响。虽然这些方法能显著提升智能体的决策能力，但往往以效率为代价，尤其是在不需要复杂推理的任务中。例如，如图 1 所示，在搜索等简单任务中，智能体可能需要对每一步进行推理，如点击搜索框、输入文本和按提交，导致不必要的延迟。传统的机器人流程自动化 (RPA) 方法虽然能快速执行这些固定步骤，但缺乏处理需要智能判断任务的灵活性。这凸显了开发更高效、适应性更强的自动化常规任务方法的必要性。

In response to this challenge, we propose a novel, evolving framework for GUI agents that aims to enhance both the efficiency and intelligence of the agent's behavior.

针对这一挑战，我们提出了一个新颖且不断发展的图形用户界面代理框架，旨在提升代理行为的效率和智能水平。

Our approach enables the agent to learn from their past interactions and dynamically evolve more abstract,

high-level actions, eliminating the need for repetitive low-level operations. Specifically, the agent will analyze its execution history to identify patterns in repetitive, low-intelligence actions, such as those involved in routine tasks. From this analysis, the agent can generate a high-level action, encapsulating a series of low-level actions, enabling it to perform tasks more efficiently. For example, in a search operation, the agent would automatically evolve a ”search” action that directly executes the required sequence, improving both speed and accuracy.

我们的方法使代理能够从过去的交互中学习，并动态演化出更抽象的高层次动作，消除重复执行低层次操作的需求。具体来说，代理将分析其执行历史，识别重复且低智能的动作模式，如常规任务中的操作。通过这种分析，代理可以生成一个高层次动作，封装一系列低层次动作，从而更高效地完成任务。例如，在搜索操作中，代理会自动演化出一个“搜索”动作，直接执行所需的动作序列，提高速度和准确性。

Intuitively, our framework enables the agent to focus its reasoning capabilities on tasks requiring more intelligent judgments, while simplifying repetitive tasks into a more compact form. To support this approach, we design a knowledge base structured as a chain to record task execution history and facilitate the abstraction and evolution of behaviors. This knowledge base allows the agent to continuously improve its task execution strategies, further optimizing its performance and enabling the evolution of more intelligent, high-level actions over time.

直观地说，我们的框架使代理能够将推理能力集中于需要更智能判断的任务，同时将重复性任务简化为更紧凑的形式。为支持这一方法，我们设计了一个链式结构的知识库，用于记录任务执行历史，促进行为的抽象与演化。该知识库使代理能够持续改进任务执行策略，进一步优化性能，并随着时间推移演化出更智能的高层次动作。

Our approach relies entirely on visual information, eliminating the need for backend access or APIs. Extensive experiments show that it outperforms baseline and state-of-the-art (SOTA) methods in both efficiency and accuracy across several benchmark tasks.

我们的方法完全依赖视觉信息，无需后台访问或 API。大量实验表明，在多个基准任务中，其效率和准确性均优于基线方法和最先进 (SOTA) 方法。

The main contributions of this paper are summarized as follows:

本文的主要贡献总结如下:

- We propose an evolutionary mechanism that enables a GUI Agent to learn from its task execution history and improve its efficiency by abstracting repetitive operations.
    - 我们提出了一种进化机制，使 GUI 代理能够从任务执行历史中学习，通过抽象重复操作提升效率。
- We design a chain-based framework for recording and optimizing the agent’s execution behavior.
    - 我们设计了一个基于链式结构的框架，用于记录和优化代理的执行行为。
- Our code will be open-sourced to facilitate further research in this area.
    - 我们的代码将开源，以促进该领域的进一步研究。

2 Related Works

2 相关工作

Large language models. Recent advancements in large language models (LLMs) have significantly expanded the scope of AI-driven automation. Models such as GPT-4 (OpenAI, 2024) and DeepSeek-V3 (DeepSeek-AI, 2024) exhibit strong natural language understanding and reasoning abilities. These capabilities allow LLMs to process complex UI structures and facilitate interactive decision-making, forming the foundation for LLM-driven GUI Agents (Naveed et al., 2024). Unlike traditional script-based or rule-based approaches (Tentarelli et al., 2022; Hellmann and Maurer, 2011), LLM-powered agents can generalize across diverse applications and dynamic interfaces without explicitly predefined rules. However, challenges remain in model efficiency, adaptability, and spatial reasoning, necessitating further optimization in both architectural design and training methodologies.

大型语言模型。近年来，大型语言模型 (LLMs) 的进展显著拓展了 AI 驱动自动化的范围。诸如 GPT-4(OpenAI，2024) 和 DeepSeek-V3(DeepSeek-AI，2024) 等模型展现了强大的自然语言理解和推理能力。这些能力使 LLMs 能够处理复杂的用户界面结构并促进交互式决策，构成了基于 LLM 的 GUI 代理 (Naveed 等，2024) 的基础。与传统的基于脚本或规则的方法 (Tentarelli 等，2022；Hellmann 和 Maurer，2011) 不同，LLM 驱动的代理能够跨多样化应用和动态界面进行泛化，无需显式预定义规则。然而，模型效率、适应性和空间推理仍面临挑战，需在架构设计和训练方法上进一步优化。

LLMs as Agents. LLMs have significantly advanced intelligent agents, enabling complex task execution. AutoGPT(Yang et al., 2023), AFlow(Zhang et al., 2024b), MetaGPT(Hong et al., 2024), and AutoAgent(Chen et al., 2024) exemplify autonomous task decomposition and execution, while Stanford Smallville(Park et al., 2023) and Agent Hospital(Li et al., 2025) showcase multi-agent simulations. LLM-driven multimodal agents also enhance perception and decision-making across domains. GPT-Driver(Mao et al., 2023) enables adaptive motion planning for autonomous driving, SmartPlay(Wu et al., 2024) improves agent intelligence in gaming, and MP5(Qin et al., 2024) integrates active perception for efficient task execution in robotics.

LLMs 作为代理。LLMs 显著推动了智能代理的发展，实现了复杂任务的执行。AutoGPT(Yang 等，2023)、AFlow(Zhang 等，2024b)、MetaGPT(Hong 等，2024) 和 AutoAgent(Chen 等，2024) 展示了自主任务分解与执行，而 Stanford Smallville(Park 等，2023) 和 Agent Hospital(Li 等，2025) 则体现了多代理仿真。基于 LLM 的多模态代理还增强了跨领域的感知与决策能力。GPT-Driver(Mao 等，2023) 实现了自动驾驶的自适应运动规划，SmartPlay(Wu 等，2024) 提升了游戏中代理的智能，MP5(Qin 等，2024) 则整合了主动感知以实现机器人高效任务执行。

LLMs as GUI Agents. Beyond general agents, LLMs have also enhanced GUI automation, surpassing traditional script-based methods in flexibility and adaptability. WebVoyager(He et al., 2024), AppAgent(Zhang et al., 2023), and Mo-bileAgent(Wang et al., 2024b) leverage multimodal perception for interactive interfaces, while UFO(Zhang et al., 2024a), AutoGLM(Liu et al., 2024), and MMAC-Copilot(Song et al., 2025) improve cross-platform adaptability and multi-agent collaboration. To refine UI understanding, OmniParser(Lu et al., 2024) and Ferret-UI(You et al., 2024a) enhance element recognition, while TinyClick(Pawlowski et al., 2024) and CogA-gent(Hong et al., 2023) improve interaction precision and vision-based task execution. Additionally, WebGUM(Furuta et al., 2024), MobileVLMS(Chu et al., 2024), and DigiRL(Bai et al., 2024) optimize performance in dynamic web and mobile environments. However, most existing agents rely on static training data and lack continuous adaptation. To address this limitation, we propose AppAgentX, integrating task trajectory memory

to enhance efficiency and adaptability in GUI automation.

LLMs 作为 GUI 代理。除了通用代理,LLMs 还提升了 GUI 自动化,超越了传统脚本方法在灵活性和适应性上的局限。WebVoyager(He 等, 2024)、AppAgent(Zhang 等, 2023) 和 MobileAgent(Wang 等, 2024b) 利用多模态感知实现交互界面操作，而 UFO(Zhang 等，2024a)、AutoGLM(Liu 等，2024) 和 MMAC-Copilot(Song 等,2025) 则提升了跨平台适应性和多代理协作。为优化 UI 理解,OmniParser(Lu 等,2024) 和 Ferret-UI(You 等，2024a) 增强了元素识别，TinyClick(Pawlowski 等，2024) 和 CogAgent(Hong 等，2023) 提升了交互精度和基于视觉的任务执行。此外，WebGUM(Furuta 等，2024)、MobileVLMS(Chu 等，2024) 和 DigiRL(Bai 等，2024) 优化了动态网页和移动环境中的性能。然而，大多数现有代理依赖静态训练数据，缺乏持续适应能力。为解决此限制，我们提出了 AppAgentX，集成任务轨迹记忆，提升 GUI 自动化的效率和适应性。

3 Preliminary

3 预备知识

Before delving into our proposed methodology, we first introduce a baseline method for LLM-based GUI agents. This baseline serves as a foundation for understanding the core components and tasks involved in enabling LLMs to control smartphones.

在深入介绍我们提出的方法之前，首先介绍一种基于大语言模型 (LLM) 的 GUI 代理的基线方法。该基线方法为理解使 LLM 能够控制智能手机的核心组件和任务提供了基础。

The process of utilizing LLMs for smartphone control involves two key stages: screen perception and action execution. The screen perception phase begins with capturing a screenshot of the device's

利用 LLM 控制智能手机的过程包括两个关键阶段: 屏幕感知和动作执行。屏幕感知阶段始于捕获设备当前界面的截图。

current interface. In order to accurately interpret this screenshot, we employ OmniParser (Lu et al., 2024) to detect and label all interactive elements within the interface, such as buttons and text boxes. OmniParser annotates these elements with tagged bounding boxes, which are subsequently overlaid onto the original screenshot for clear visualization. Following this, the annotated screenshot is passed to the LLM for action planning. At this stage, the LLM interprets the UI components and generates corresponding actions based on its understanding of the interface.

为了准确解读该截图，我们采用 OmniParser(Lu 等，2024) 检测并标注界面中的所有交互元素，如按钮和文本框。OmniParser 使用带标签的边界框对这些元素进行注释，随后将其叠加在原始截图上以便清晰展示。接着，带注释的截图被传递给 LLM 进行动作规划。在此阶段，LLM 解读 UI 组件并基于对界面的理解生成相应的操作。

In the second stage, action execution, we follow AppAgent (Zhang et al., 2023) to define a set of low-level actions that the agent can perform within the smartphone environment. These actions include common gestures such as tapping, long-pressing, swiping, text input, and navigating back. These actions collectively define a basic, app-agnostic action space to simulate typical human interactions with a smartphone interface. Formally, the low-level action space is defined as follows:

在第二阶段动作执行中，我们遵循 AppAgent(Zhang 等，2023) 定义了一组代理可在智能手机环境中执行的低级动作。这些动作包括常见手势，如点击、长按、滑动、文本输入和返回导航。这些动作共同定义了一个基础的、与应用无关的动作空间，用以模拟人类与智能手机界面的典型交互。形式上，低级动作空间定义如下:

$$\mathcal{A}_{\text{basic}} = \{a_{\text{tap}}, a_{\text{long\_press}}, a_{\text{swipe}}, a_{\text{text}}, a_{\text{back}}\}, \tag{1}$$

where $\mathcal{A}_{\text{basic}}$ represents the set of atomic actions available to the agent.

其中 $\mathcal{A}_{\text{basic}}$ 表示代理可用的原子动作集合。

The LLM employs a structured process of observation, reasoning, and function-calling to interact with the smartphone interface. In this process, the LLM iteratively analyzes the current UI, reasons about the appropriate next steps to achieve the desired task, and invokes the corresponding actions from the defined action space. This cycle continues until the task is completed successfully. An illustration of this process is shown in Figure 2.

LLM 采用观察、推理和函数调用的结构化过程与智能手机界面交互。在此过程中，LLM 迭代分析当前 UI，推理实现目标任务的适当下一步，并从定义的动作空间中调用相应动作。该循环持续进行，直至任务成功完成。该过程示意见图 2。

4 Methodology

4 方法论

This section outlines the core methodology of the proposed evolutionary framework. The framework comprises three main components: a memory mechanism for recording the agent's operational history, an evolutionary mechanism for improving performance, and an execution strategy for utilizing the evolved agent. We will detail each of these elements in the following subsections.

本节概述所提进化框架的核心方法论。该框架包含三个主要组成部分: 用于记录代理操作历史的记忆机制、用于提升性能的进化机制以及用于利用进化代理的执行策略。以下小节将详细介绍这些要素。

4.1 Memory mechanism

4.1 记忆机制

To support the agent's evolution from inefficient to efficient operational modes, it is essential for the agent to retain a record of its past actions and the corresponding outcomes. This enables the agent

为了支持代理从低效到高效的操作模式进化，代理必须保留其过去动作及相应结果的记录。这使代理能够

Step X: Playing the Music

步骤 X: 播放音乐

Figure 2: Overview of the LLM-based GUI agent baseline. At each step, the agent captures the current screen of the device and analyzes the interface to select an appropriate action from the predefined action space. The chosen action is then executed to interact with the GUI.

图 2: 基于 LLM 的 GUI 代理基线概览。在每一步，代理捕获设备当前屏幕并分析界面，从预定义动作空间中选择合适动作。随后执行所选动作以与 GUI 交互。

to learn from its experiences and improve future interactions. To achieve this, we propose a memory mechanism designed to capture and store the agent's trajectory during its interactions with the environment.

从经验中学习并改进未来交互。为此，我们提出了一种记忆机制，用于捕捉并存储代理与环境交互过程中的轨迹。

The agent's interaction with the UI is modeled as a series of page transitions, where each UI page is represented as a "page node". Interactions performed on these pages, such as button clicks or text input, lead to transitions between these nodes. Each page node is characterized by several key attributes, including:

代理与 UI 的交互被建模为一系列页面跳转，每个 UI 页面表示为一个“页面节点”。在这些页面上执行的交互，如按钮点击或文本输入，会导致节点间的跳转。每个页面节点具有若干关键属性，包括:

>Page Description: This attribute stores a text description of the entire UI page, initially setting it to empty at the beginning.

> 页面描述: 该属性存储整个 UI 页面的文本描述，初始时为空。

>Element List: This property holds a JSON list containing information of all elements detected by OmniParser(Lu et al., 2024), such as the relative screen position, OCR results, etc.

> 元素列表: 该属性保存一个 JSON 列表，包含 OmniParser(Lu 等，2024) 检测到的所有元素信息，如相对屏幕位置、OCR 结果等。

>Other Properties: The page nodes also include other properties required for logging, such as page screenshots, ID, timestamps, etc.

> 其他属性: 页面节点还包括日志记录所需的其他属性，如页面截图、ID、时间戳等。

For more detailed information, we introduce ”element nodes”. Each element node corresponds to a specific UI element, such as a button, text field, or icon. The interactions with these UI elements lead to the page transitions. Similarly, each element node encapsulates essential attributes:

为了更详细的信息，我们引入“元素节点”。每个元素节点对应一个特定的 UI 元素，如按钮、文本框或图标。与这些 UI 元素的交互导致页面跳转。同样，每个元素节点封装了关键属性:

>Element Description: This attribute records a textual description of the element’s functionality, providing a semantic understanding of the element’s purpose within the UI. Similar to page descriptions, this attribute is initialized as empty.

> 元素描述: 该属性记录元素功能的文本描述，提供对元素在 UI 中用途的语义理解。与页面描述类似，该属性初始为空。

>Element Visual Embeddings: This property stores the identifier of the element’s screenshot in the vector database. The visual features are extracted using a pre-trained ResNet50 (Microsoft, 2024) model.

> 元素视觉嵌入: 该属性存储元素截图在向量数据库中的标识符。视觉特征由预训练的 ResNet50(微软，2024) 模型提取。

>Interaction Details:This property includes information related to the basic action for the current element, e.g., tapping, along with the corresponding arguments and other relevant interaction details.

> 交互细节: 该属性包含当前元素的基本操作信息，如点击，以及相应的参数和其他相关交互细节。

This structure enables the agent to record and learn from both the high-level transitions (from one page to another) and the low-level interactions (with individual UI elements) that lead to those transitions. Figure 3 (c) illustrates this process.

该结构使代理能够记录并学习从高层次的页面跳转 (从一个页面到另一个页面) 到低层次的交互 (与单个 UI 元素的交互) 所导致的跳转。图 3(c) 展示了该过程。

Extracting Features and Descriptions. We next leverage the LLM to generate functional descriptions of pages and individual elements based on observed action sequences. Specifically, we follow the approach in (Zhang et al., 2023) by decomposing the agent’s trajectory into multiple overlapping triples. Each triple consists of a source page, an action performed on an element, and a target page. These triples capture the changes in page states before and after an action is executed.

特征与描述提取。接下来，我们利用大型语言模型 (LLM) 基于观察到的动作序列生成页面和单个元素的功能描述。具体而言，我们遵循 (Zhang et al., 2023) 的方法，将代理的轨迹分解为多个重叠的三元组。每个三元组由源页面、对元素执行的动作和目标页面组成。这些三元组捕捉了动作执行前后页面状态的变化。

The resulting triples are then passed to the LLM for reasoning. The model generates detailed descriptions and functionalities for both the page and element nodes based on the context of the action. This process, illustrated in Figure 3 (a), enables the system to build accurate and contextually aware descriptions.

生成的三元组随后传递给 LLM 进行推理。模型基于动作的上下文，为页面和元素节点生成详细的描述和功能说明。该过程如图 3(a) 所示，使系统能够构建准确且具上下文感知的描述。

Merging Overlapping Descriptions. Since the descriptions of page nodes are generated from multiple overlapping action triples, it is likely that the descriptions for the same page node will be generated twice, based on different contexts. Therefore, it is necessary to merge them to generate a unified description for each page node. To achieve this, we instruct the LLM to combine the descriptions generated from different triples, taking into account both the specific context of each individual action and the broader global task the agent is performing. This allows the LLM to generate a more enriched and complete description of the page, considering its function in the overall task. The merged descriptions provide a detailed, unified record of the agent's interactions with the UI, contributing to a coherent chain of node attributes that document the agent's progress as it completes the task.

合并重叠描述。由于页面节点的描述是从多个重叠的动作三元组生成的，同一页面节点的描述可能基于不同上下文被生成多次。因此，需要将它们合并以生成每个页面节点的统一描述。为此，我们指示 LLM 结合不同三元组生成的描述，兼顾每个动作的具体上下文和代理执行的整体任务。这使 LLM 能够生成更丰富完整的页面描述，考虑其在整体任务中的功能。合并后的描述提供了代理与 UI 交互的详细统一记录，有助于形成连贯的节点属性链，记录代理完成任务的进展。

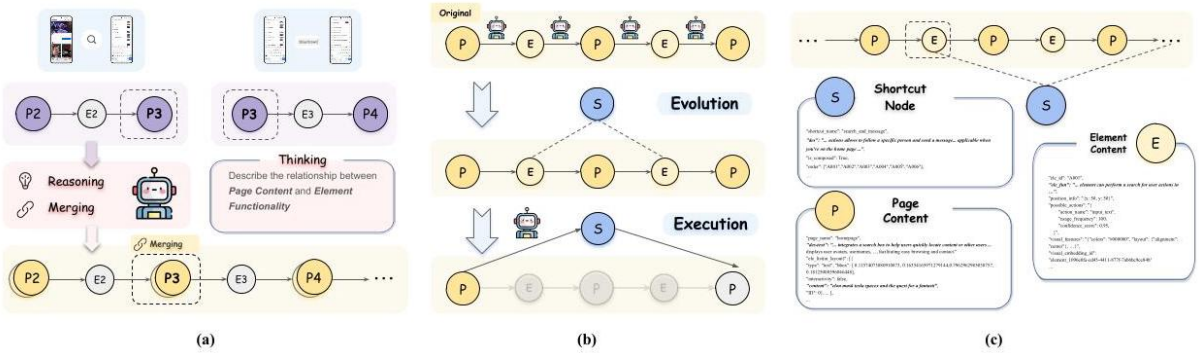

Figure 3: Overview of the proposed framework. (a) The trajectory of a task execution is decomposed into multiple overlapping triples. Based on these triples, the LLM generates functional descriptions of both pages and UI elements. Descriptions of pages that are repeatedly generated are then merged. The entire interaction history is recorded using a chain of nodes. (b) The proposed evolutionary mechanism and execution process. The evolutionary mechanism generates shortcut nodes, which allow the agent to execute a series of actions efficiently without reasoning step by step. (c) An example of the content of nodes within the chain. Each node records essential

information, including descriptions of pages, UI elements, and high-level actions, to facilitate understanding of the agent's interactions.

图 3: 所提框架概览。(a) 任务执行轨迹被分解为多个重叠三元组。基于这些三元组，LLM 生成页面和 UI 元素的功能描述。重复生成的页面描述随后被合并。整个交互历史通过节点链记录。(b) 所提进化机制及执行过程。进化机制生成快捷节点，使代理能够高效执行一系列动作，无需逐步推理。(c) 链中节点内容示例。每个节点记录关键信息，包括页面描述、UI 元素和高层动作，便于理解代理的交互。

4.2 Evolutionary Mechanism

4.2 进化机制

The primary goal of the evolutionary mechanism is to improve the agent's accuracy and efficiency in executing similar tasks by learning from its previous execution histories. In typical task execution scenarios, the action sequence may contain repetitive patterns that do not require deep reasoning at each step. For example, to search for a content item, the agent may repeatedly perform actions such as tapping the search box, inputting text, and tapping the search button. By identifying and exploiting these repetitive patterns, the agent can replace inefficient action sequences with higher-level actions that streamline the process, thereby improving overall task efficiency and accuracy. To achieve this, we introduce the concept of a shortcut node, which represents a high-level action that bypasses previously inefficient action sequences. The shortcut node is designed to streamline its decision-making process, eliminating unnecessary reasoning process and accelerating task completion.

进化机制的主要目标是通过学习先前的执行历史，提高代理执行类似任务的准确性和效率。在典型任务执行场景中，动作序列可能包含无需每步深度推理的重复模式。例如，为搜索内容项，代理可能反复执行点击搜索框、输入文本、点击搜索按钮等动作。通过识别并利用这些重复模式，代理可以用更高级的动作替代低效的动作序列，从而简化流程，提高整体任务效率和准确性。为此，我们引入快捷节点的概念，代表绕过先前低效动作序列的高级动作。快捷节点旨在简化决策过程，消除不必要的推理，加速任务完成。

To identify whether an action sequence contains repetitive patterns that can be replaced by shortcut nodes, we design mechanisms that evaluate the trajectory of each task. Initially, LLM, drawing from prior knowledge, is tasked with determining whether a given task contains repetitive patterns that may be optimized through the introduction of shortcut nodes. If the task is deemed to contain such patterns, the next step involves inspecting the actual trajectory data. The trajectory data, which

为识别动作序列中是否包含可由快捷节点替代的重复模式，我们设计了评估每个任务轨迹的机制。最初，LLM 基于先验知识判断给定任务是否包含可通过引入快捷节点优化的重复模式。如果任务被判定包含此类模式，下一步则是检查实际轨迹数据。轨迹数据，

includes the descriptions of all relevant pages and elements, is input into the LLM. Upon receiving this data, the model is prompted to generate the description of a shortcut node, specifying the scenarios in which the shortcut node can be invoked and the specific low-level action sequences that it is intended to replace. This process effectively constructs an evolved action space by integrating new, higher-level actions into the agent's operational repertoire, allowing it to perform tasks more efficiently.

包括所有相关页面和元素的描述，被输入到大型语言模型 (LLM) 中。接收到这些数据后，模型被提示生成快捷节点的描述，指定快捷节点可被调用的场景以及其旨在替代的具体低级动作序列。该过程通过将新的高级动作整合到代理的操作库中，有效构建了一个进化的动作空间，使其能够更高效地执行任务。

To formally define the construction of high-level actions and the expansion of the action space, we introduce the following representations. A high-level action $\widetilde{a}$ is composed by abstracting a sequence of low-level actions from the basic action set $\mathcal{A}_{\text{basic}}$. This abstraction allows the agent to perform more complex tasks with fewer steps, leveraging previously learned patterns of interaction. We then define the expanded action space $\mathcal{A}_{\text{evolved}}$, which incorporates both the original low-level actions and the newly introduced high-level actions:

为了正式定义高级动作的构建及动作空间的扩展，我们引入以下表示。一个高级动作 $\widetilde{a}$ 是通过从基本动作集 $\mathcal{A}_{\text{basic}}$ 中抽象出一系列低级动作组成的。这种抽象使代理能够以更少的步骤执行更复杂的任务，利用先前学习的交互模式。然后我们定义扩展后的动作空间 $\mathcal{A}_{\text{evolved}}$ ，它包含了原始的低级动作和新引入的高级动作:

$$\mathcal{A}_{\text{evolved}} = \mathcal{A}_{\text{basic}} \cup \{\widetilde{a}\}. \tag{2}$$

Here, $\mathcal{A}_{\text{evolved}}$ represents the enriched action space that includes both basic, low-level actions and the newly composed high-level actions. This expanded action space enables the agent to perform tasks more efficiently by utilizing higher-level abstractions that reduce the need for repetitive action sequences.

这里，$\mathcal{A}_{\text{evolved}}$ 表示包含基本低级动作和新组合的高级动作的丰富动作空间。该扩展动作空间使代理能够通过利用更高级的抽象，减少重复动作序列的需求，从而更高效地执行任务。

4.3 Dynamic Action Execution

4.3 动态动作执行

Following the evolution of the action space, the agent can now select high-level actions from the expanded action space to efficiently execute tasks.

随着动作空间的演进，代理现在可以从扩展的动作空间中选择高级动作，以高效地执行任务。

4.3.1 Execution Process

4.3.1 执行过程

The high-level execution process begins after the agent captures a screenshot of the current page. At this stage, the system then matches the parsed elements on the page to the stored element nodes in memory, by comparing their visual embeddings. Next, we check whether these identified element nodes are associated with any shortcut nodes. If an association between the element nodes and shortcut nodes exists, the system leverages the LLM to determine whether the corresponding high-level actions can be executed. This decision is made by inspecting the description

of the shortcut node in conjunction with the current task context. If the conditions for executing the high-level action are met, the LLM generates an action execution template. This template includes the sequence of low-level actions to be executed by the shortcut node, along with the corresponding function arguments necessary for each action. In cases where multiple elements on the page are associated with shortcut nodes, the system prioritizes executing the actions based on the order of matching elements. This order is determined by their execution sequence, ensuring that the actions are performed in a logical and efficient manner. As the sequence actions progresses, partially repetitive operations or even the entire task can be completed more efficiently through the use of high-level actions.

高级执行过程始于代理截取当前页面的截图。在此阶段，系统通过比较视觉嵌入，将页面上解析出的元素与内存中存储的元素节点进行匹配。接着，我们检查这些识别出的元素节点是否与任何快捷节点相关联。如果元素节点与快捷节点存在关联，系统利用大型语言模型 (LLM) 判断是否可以执行对应的高级动作。该决策通过结合快捷节点的描述和当前任务上下文进行。如果满足执行高级动作的条件，LLM 将生成一个动作执行模板。该模板包括快捷节点要执行的低级动作序列及每个动作所需的相应函数参数。在页面上多个元素与快捷节点关联的情况下，系统优先根据匹配元素的顺序执行动作。该顺序由其执行次序决定，确保动作以逻辑且高效的方式执行。随着动作序列的推进，部分重复操作甚至整个任务都能通过使用高级动作更高效地完成。

4.3.2 Fallback Strategy

4.3.2 备选策略

To ensure robustness and task completion reliability, we introduce a fallback strategy that allows the agent to dynamically recover from failures. If the conditions for executing a high-level action are not met, the agent will default to selecting actions from the basic action space $\mathcal{A}$. Additionally, in cases where execution errors occur due to incorrect shortcut node matching or unexpected UI responses, the agent reverts to using actions from the basic action space as a fallback. This ensures that the agent can still operate effectively, even when high-level abstractions cannot be applied to the current task.

为确保鲁棒性和任务完成的可靠性，我们引入了备选策略，允许代理动态从失败中恢复。如果不满足执行高级动作的条件，代理将默认从基本动作空间 $\mathcal{A}$ 中选择动作。此外，在因快捷节点匹配错误或意外的用户界面响应导致执行错误的情况下，代理也会回退使用基本动作空间的动作作为备选。这确保即使高级抽象无法应用于当前任务，代理仍能有效运行。

During the execution of high-level paths, operations that would traditionally require multiple reasoning steps by the LLM are transformed into a page-matching and retrieval-based process. This

在执行高级路径时，传统上需要大型语言模型多步推理的操作被转化为基于页面匹配和检索的过程。这种

shift significantly enhances the overall execution efficiency, as the agent can bypass repeated reasoning processes and rely on pre-determined, efficient action sequences.

转变显著提升了整体执行效率，因为代理可以绕过重复的推理过程，依赖预先确定的高效动作序列。

5 Experiments

5 实验

In this section, we will present our evaluation of the evolutionary framework through a combination of various experiments on multiple benchmarks.

本节将通过在多个基准上的多种实验，展示我们进化框架的评估结果。

5.1 Experimental Setup

5.1 实验设置

Evaluation metrics. To ensure a fair and accurate comparison of our proposed method with baseline models and existing works, we adopt several evaluation metrics, which have been commonly used in prior research (Zhang et al., 2023; Wang et al., 2024b; Li et al., 2024). The metrics we report and compare are as follows:

评估指标。为了确保我们提出的方法与基线模型及现有工作之间的公平且准确的比较，我们采用了若干评估指标，这些指标在先前研究中被广泛使用 (Zhang et al., 2023；Wang et al., 2024b；Li et al., 2024)。我们报告和比较的指标如下:

>Average Steps per Task (Steps): This measures the average number of operations the agent performs to complete a task.

> 每任务平均步骤数 (Steps): 衡量代理完成一个任务所执行的平均操作次数。

>Average Overall Success Rate (SR): This metric evaluates the proportion of tasks completed successfully by the agent.

> 平均整体成功率 (SR): 该指标评估代理成功完成任务的比例。

>Average Task Time (Task Time): The total time taken to complete a task, starting from the initiation of the task execution process to the determination of task completion by the agent.

> 平均任务时间 (任务时间): 完成任务所花费的总时间，从任务执行过程开始到代理确定任务完成为止。

>Average Step Time (Step Time): The average time consumed per operation (step) during task execution.

> 平均步骤时间 (步骤时间): 任务执行过程中每个操作 (步骤) 所消耗的平均时间。

>Average LLM Token Consumption (Tokens): The total number of tokens used by the language model (LLM) during the task execution, including both prompt tokens and completion tokens.

> 平均大语言模型 (LLM) 令牌消耗 (Tokens): 任务执行期间大语言模型 (LLM) 使用的令牌总数，包括提示令牌和完成令牌。

For time-related comparisons, we focus only on tasks that are successfully executed by all methods. This ensures that the comparisons are not skewed by failures due to early terminations or excessive retry attempts, which might distort the results. To calculate token consumption, we compute the total number of prompt and completion tokens used by the LLM for each task and report the average number across all tasks. To mitigate the impact of random fluctuations, each task is repeated five times by default, and we report the averaged results across these repetitions. This helps ensure that our findings are statistically robust and not affected by outlier performance from individual task executions.

对于时间相关的比较，我们仅关注所有方法均成功执行的任务。这确保比较结果不会因提前终止或过多重试导致的失败而产生偏差，从而扭曲结果。令牌消耗的计算方法是统计每个任务中大语言模型使用的提示令牌和完成令牌总数，并报告所有任务的平均值。为减小随机波动的影响，每个任务默认重复执行五次，并报告这些重复的平均结果。这有助于确保我们的结论具有统计学上的稳健性，不受单次任务执行异常表现的影响。

Benchmarks. We evaluate the performance of our

基准测试。我们评估了我们的

Table 1: Analysis of Different Components in AppAgentX. This table compares the performance differences resulting from the different designs with the baseline. In that table for the GPT-4o approach, we use direct LLM invocation. Both our memory design and evolution mechanism can improve success rate and efficiency.

表 1:AppAgentX 中不同组件的分析。该表比较了不同设计相较基线的性能差异。对于 GPT-4o 方法，我们采用直接调用大语言模型 (LLM)。我们的记忆设计和进化机制均能提升成功率和效率。

Method	Memory Type	Action Space	Steps↓	Step Time (s) ↓	Tokens (k) ↓	SR ↑
GPT-40	None	Basic	10.8	26	6.72	16.9%
AppAgent	Element	Basic	9.3	24	8.46	69.7%
AppAgentX	Chain	Basic	9.1	23	9.26	70.8%
AppAgentX	Chain	Basic+Evolve	5.7	16	4.94	71.4%

方法	内存类型	动作空间	步骤 ↓	单步时间 (秒) ↓	标记数 (千) ↓	成功率 ↑
GPT-40	无	基础	10.8	26	6.72	16.9%
应用代理	元素	基础	9.3	24	8.46	69.7%
应用代理 X	链	基础	9.1	23	9.26	70.8%
应用代理 X	链	基础 + 进化	5.7	16	4.94	71.4%

method on several widely used benchmarks to validate its effectiveness and generalizability. These benchmarks cover a diverse range of tasks and applications, providing a comprehensive assessment of our framework's capabilities. The benchmarks used in our experiments include:

在多个广泛使用的基准测试上验证该方法的有效性和泛化能力。这些基准涵盖了多样化的任务和应用，全面评估了我们框架的能力。我们实验中使用的基准包括:

>AppAgent Benchmark (Zhang et al., 2023): This benchmark consists of 50 tasks across 9 different applications, including 45 general tasks and 5 long-duration tasks.

>AppAgent 基准 (Zhang 等，2023): 该基准包含 9 个不同应用中的 50 个任务，包括 45 个通用任务和 5 个长时任务。

>DroidTask (Wen et al., 2024): Comprising 158 high-level tasks derived from 13 widely used mobile applications.

>DroidTask(Wen 等，2024): 由 13 个广泛使用的移动应用中提取的 158 个高级任务组成。

>AndroidWorld (Rawles et al., 2025): Android-World is a fully functional and dynamic Android benchmark that supports 116 programmatic tasks across 20 widely used real-world Android applications.

>AndroidWorld(Rawles 等，2025):AndroidWorld 是一个功能完备且动态的 Android 基准，支持 20 个广泛使用的真实 Android 应用中的 116 个程序化任务。

>A3 (Android Agent Arena) (Chai et al., 2025): It consists of 201 tasks derived from 20 widely used third-party applications, covering common user scenarios. The benchmark provides a comprehensive evaluation mechanism, which significantly accelerates our experimental work. Implementation details. The implementation of our framework leverages several key platforms. For the foundational LLM, we selected GPT-40 (OpenAI, 2024) as the default model unless otherwise stated. The LangGraph framework (LangChain, 2024) is used as the agent platform, providing essential features for LLM input-output parsing and process control. To implement the memory mechanism, we integrated Neo4j (Neo4j, 2024) for graph-based storage and retrieval and Pinecone (Pinecone, 2024) for vector search. For feature matching, we employed cosine similarity with embeddings derived from ResNet-50 (He et al., 2015), which enables effective task representation and retrieval. All experiments were conducted using the Android Debug Bridge (ADB), enabling on-device evaluations for mobile applications.

>A3(Android Agent Arena)(Chai 等，2025): 由 20 个广泛使用的第三方应用中提取的 201 个任务组成，涵盖常见用户场景。该基准提供了全面的评估机制，显著加速了我们的实验工作。实现细节。我们的框架实现依托多个关键平台。基础大语言模型 (LLM) 默认选用 GPT-40(OpenAI，2024)，除非另有说明。LangGraph 框架 (LangChain，2024) 作为代理平台，提供了 LLM 输入输出解析和流程控制的核心功能。为实现记忆机制，我们集成了 Neo4j(Neo4j，2024) 用于基于图的存储与检索，Pinecone(Pinecone，2024) 用于向量搜索。特征匹配采用基于 ResNet-50(He 等，2015) 生成的嵌入向量的余弦相似度，实现有效的任务表示与检索。所有实验均通过 Android 调试桥 (ADB) 进行，支持移动应用的设备端评估。

5.2 Experimental Analysis

5.2 实验分析

In this part, the experiments are designed to validate AppAgentX's advantages in terms of efficiency and accuracy. We conducted five experiments for the baseline comparison and two experiments for the large dataset to mitigate the effects of randomness in LLMs.

本部分设计实验以验证 AppAgentX 在效率和准确性方面的优势。我们进行了五组基线对比实验和两组大规模数据集实验，以减轻大语言模型随机性的影响。

Comparative Analysis. A comparison of our experimental results with model variants is shown in Table 1. We report the results on the AppA-gent benchmark. We begin with a baseline model that does not incorporate any memory mechanism and progressively introduce our proposed enhancements. The first model variant integrates a module that records only information about elements, providing a basic form of memory. Subsequently, we introduce a more advanced memory design that maintains a structured memory chain, enabling the model to retain a more comprehensive representation of past interactions. Finally, we augment the system with an evolutionary mechanism that expands the action space to include high-level actions, further optimizing task execution.

对比分析。表 1 展示了我们实验结果与模型变体的对比，结果基于 AppAgent 基准。我们从不包含任何记忆机制的基线模型开始，逐步引入提出的改进。第一个模型变体集成了仅记录元素信息的模块，提供了基础的记忆形式。随后引入更先进的记忆设计，维护结构化的记忆链，使模型能够保留更全面的历史交互表示。最后，系统增加了进化机制，扩展动作空间以包含高级动作，进一步优化任务执行。

As the result shows, incorporating the memory mechanism significantly improves the task success rate. The baseline model, which lacks memory, achieves an SR of only 16.9%. Introducing the chain-structured memory significantly enhances the success rate, reaching 70.8%, highlighting the clear advantage of maintaining a more structured history of interactions. Furthermore, it proves to be more effective than element memory in facilitating task completion. Moreover, the integration of the evolutionary mechanism leads to substantial efficiency gains. Expanding the action space to include high-level actions decreases the average required number of steps from 9.1 to 5.7, while the step execution time is reduced from 23 to 16 seconds. Additionally, average token consumption is significantly minimized, dropping from 9.26k to 4.94k, indicating a more efficient decision-making

结果显示，加入记忆机制显著提升了任务成功率。缺乏记忆的基线模型成功率仅为 16.9%。引入链式结构记忆后，成功率大幅提升至 70.8%，凸显了维护更结构化交互历史的明显优势。此外，该机制在促进任务完成方面优于元素记忆。进化机制的整合带来了显著的效率提升。扩展动作空间包含高级动作后，平均所需步骤数从 9.1 降至 5.7，步骤执行时间从 23 秒缩短至 16 秒。同时，平均令牌消耗显著减少，从 9.26k 降至 4.94k，表明决策过程更高效。

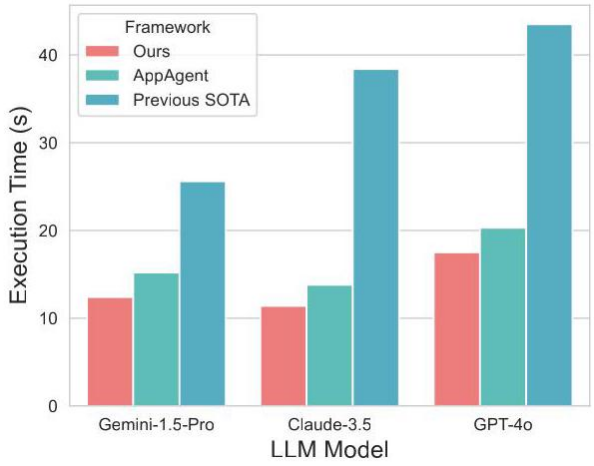

Figure 4: Comparison of Average Execution Time per Step. This figure presents the average execution time per steps across different LLMs and frameworks.

图 4: 各步骤平均执行时间比较。该图展示了不同大语言模型和框架下各步骤的平均执行时间。

process. Notably, this enhancement reduces computational overhead and improves the average success rate to 71.4%. These results show that our approach is effective. The memory mechanisms significantly improve task performance, while the evolutionary strategy boosts efficiency by reducing steps, execution time, and token use.

过程。值得注意的是，该改进降低了计算开销，平均成功率提升至 71.4%。这些结果表明我们的方法有效。记忆机制显著提升了任务表现，进化策略通过减少步骤数、执行时间和令牌使用量提升了效率。

To further highlight the advantages of the Ap-pAgentX framework, we compare its performance to other state-of-the-art frameworks on additional benchmarks, including DroidTask (Wen et al., 2024), Android Agent Arena(Chai et al., 2025) and AndroidWorld(Rawles et al., 2025). As shown in Figure 2, AppAgentX outperforms AppAgent, in both task execution time and accuracy. Specifically, AppAgentX achieves significantly higher efficiency while maintaining higher task success rates across a broader range of applications.

为进一步突出 AppAgentX 框架的优势，我们将其性能与其他先进框架在额外基准上的表现进行了比较，包括 DroidTask(Wen 等，2024)、Android Agent Arena(Chai 等，2025) 和 AndroidWorld(Rawles 等，2025)。如图 2 所示，AppAgentX 在任务执行时间和准确率上均优于 AppAgent。具体而言，AppAgentX 在更广泛的应用中实现了显著更高的效率，同时保持了更高的任务成功率。

Additionally, we compare the execution efficiency of AppAgentX against two other frameworks (Wang et al., 2024a; Zhang et al., 2023) across several prominent foundational LLMs, including GPT-4o (OpenAI, 2024), Claude 3.5 Sonnet (Anthropic., 2024), and Gemini 1.5 Pro (Gemini, 2024). While acknowledging that certain discrepancies in the experimental setup may exist, we focus our evaluation primarily on execution time, as it serves as a practical and reproducible metric for assessing efficiency. As shown in Table 4, AppAgentX consistently demonstrates faster per-step completion times compared to the previous state-of-the-art (Wang et al., 2024a) and AppAgent, across multiple LLM backends. These results suggest improved efficiency and better alignment with real-world deployment constraints.

此外, 我们将 AppAgentX 的执行效率与另外两个框架 (Wang et al., 2024a; Zhang et al., 2023) 在多个知名基础大型语言模型 (LLM) 上进行了比较, 包括 GPT-4o(OpenAI, 2024)、Claude 3.5 Sonnet(Anthropic, 2024) 和 Gemini 1.5 Pro(Gemini, 2024)。尽管承认实验设置中可能存在某些差异, 我们的评估主要聚焦于执行时间, 因为它是衡量效率的实用且可复现的指标。如表 4 所示, AppAgentX 在多个 LLM 后端中均表现出比先前最先进方法 (Wang et al., 2024a) 和 AppAgent 更快的单步完成时间。这些结果表明了效率的提升以及与实际部署约束的更好契合。

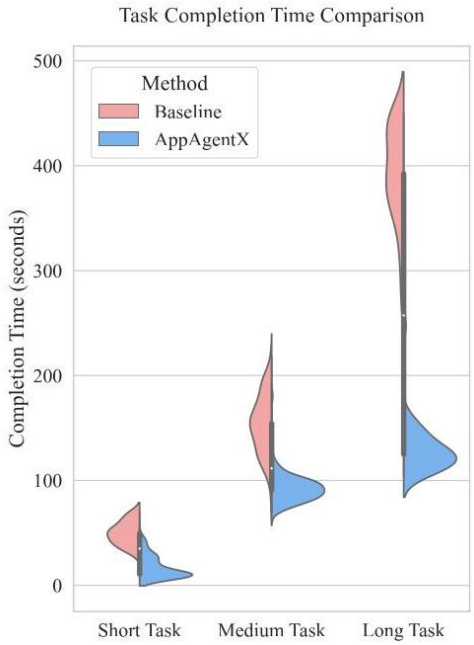

Figure 5: Task Completion Times Across Different Task Lengths. This figure shows the distribution of task completion times for short, medium, and long tasks. Each violin plot represents the density of completion times, with wider sections indicating higher data concentration. AppAgentX consistently outperforms the baseline, particularly for longer tasks.

图 5: 不同任务长度下的任务完成时间分布。该图展示了短、中、长任务的完成时间分布情况。每个小提琴图表示完成时间的密度, 宽度较大的部分表示数据集中度较高。AppAgentX 在所有任务长度中均持续优于基线, 尤其在较长任务中表现更为突出。

Task difficulties analysis. To investigate the performance of AppAgentX across tasks of varying complexity, we categorize the tasks based on their ground-truth operation steps, annotated through expert demonstrations. Tasks with up to 5 steps are considered short, tasks with 6 to 10 steps are classified as medium-length, and tasks requiring more than 10 steps are categorized as long tasks. As shown in Figure 5, the violin plot provides an intuitive visualization of how task complexity affects performance. Notably, our method demonstrates a clear advantage in terms of task execution time as the task complexity increases. To account for random fluctuations, we conducted one-tailed paired t-tests (Ross and Willson, 2017) on each data group to verify that AppAgentX significantly outperforms the baseline in efficiency, with all p-values falling below 0.05. Our model consistently outperforms the baseline across all task lengths, resulting in reduced task completion times. This finding reinforces the robustness and efficiency of our approach, confirming its stability under varying task difficulties.

任务难度分析。为探究 AppAgentX 在不同复杂度任务中的表现，我们根据专家示范标注的真实操作步骤对任务进行分类。步骤数不超过 5 步的任务视为短任务，6 至 10 步的任务归为中等长度任务，超过 10 步的任务则归为长任务。如图 5 所示，小提琴图直观展示了任务复杂度对性能的影响。值得注意的是，随着任务复杂度的增加，我们的方法在任务执行时间上表现出明显优势。为排除随机波动影响，我们对每组数据进行了单尾配对 t 检验 (Ross 和 Willson，2017)，结果显示 AppAgentX 在效率上显著优于基线，所有 p 值均低于 0.05。我们的模型在所有任务长度上均持续优于基线，显著缩短了任务完成时间。该发现进一步验证了我们方法的稳健性和高效性，确认其在不同任务难度下的稳定表现。

Ablation Study. To further evaluate the performance gains of our agent, we substitute different screen perceptrons to assess the effectiveness of

消融研究。为进一步评估我们代理的性能提升，我们替换了不同的屏幕感知器以评估其有效性

Table 2: Comparison with AppAgent on Large Benchmarks. This table evaluates the efficiency and accuracy of different frameworks on benchmarks containing a large number of tasks.

表 2: 与 AppAgent 在大型基准测试上的比较。该表评估了不同框架在包含大量任务的基准测试中的效率和准确性。

Benchmarks	Task Num.	Framework	Task Time ↓	Tokens (k) ↓	SR↑
DroidTask(Wen et al., 2024)	158	AppAgent	106.24	11.5	46.3%
		AppAgentX	56.29	5.1	88.2%
AndroidWorld(Rawles et al., 2025)	116	AppAgent	147.17	18.9	41.7%
		AppAgentX	59.74	6.2	62.5%
A3 (Android Agent Arena)(Chai et al., 2025)	201	AppAgent	134.67	19.2	10.3%
		AppAgentX	48.12	4.7	39.3%

基准测试	任务数量	框架	任务时间 ↓	标记数 (千) ↓	成功率 ↑
DroidTask(Wen 等，2024)	158	AppAgent	106.24	11.5	46.3%
		AppAgentX	56.29	5.1	88.2%
AndroidWorld(Rawles 等，2025)	116	AppAgent	147.17	18.9	41.7%
		AppAgentX	59.74	6.2	62.5%
A3(Android Agent Arena)(Chai 等，2025)	201	AppAgent	134.67	19.2	10.3%
		AppAgentX	48.12	4.7	39.3%

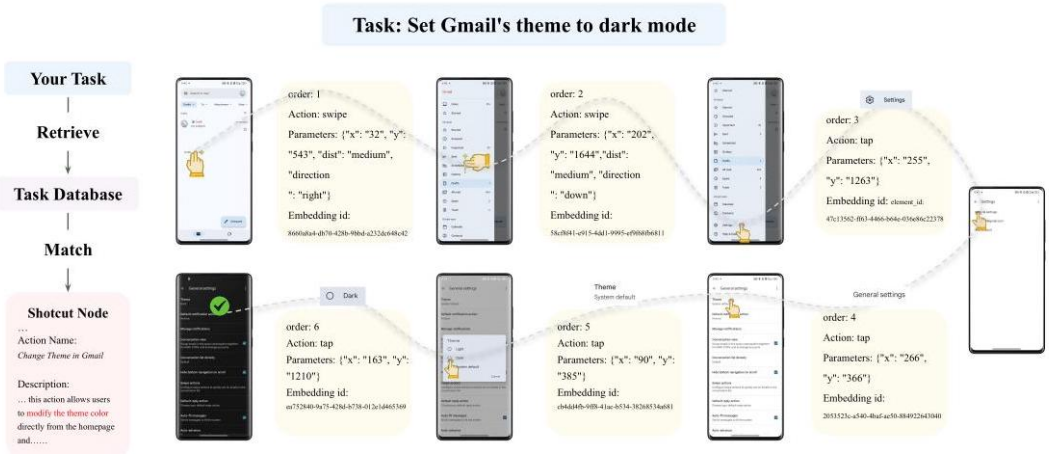

Figure 6: Qualitative Results. This figure presents the path execution utilizing shortcuts on Gmail. It demonstrates the efficiency of AppAgentX in decreasing the utilization of LLM for per-step reasoning.

图 6: 定性结果。该图展示了在 Gmail 上利用快捷方式执行路径，体现了 AppAgentX 在减少每步推理中大型语言模型 (LLM) 使用量方面的高效性。

Table 3: Ablation Study of Different Screen Percep-trons and Action Spaces. This table compares task success rates and times for different screen parsers and two action space variants: BAS (Basic Action Space) and FAS (Full Action Space).

表 3: 不同屏幕感知器和动作空间的消融研究。该表比较了不同屏幕解析器及两种动作空间变体: 基础动作空间 (BAS) 和完整动作空间 (FAS) 的任务成功率和时间。

Methods	Task Time ↓	SR↑
Omniparser + BAS	209.3s	70.8%
Omniparser + FAS	91.2s	71.4%
Ferret-UI + BAS	201.7s	68.1%
Ferret-UI + FAS	92.5s	70.6%

方法	任务时间 ↓	成功率 ↑
Omniparser + BAS	209.3s	70.8%
Omniparser + FAS	91.2s	71.4%
Ferret-UI + BAS	201.7s	68.1%
Ferret-UI + FAS	92.5s	70.6%

our evolutionary approach. Specifically, Omni-Parser(Lu et al., 2024) is employed as the default visual element extractor to locate UI components on the screen. In this study, we replace it with Ferret-UI(You et al., 2024b), a component with equivalent functionality, and compare the baseline action space with the evolved action space to examine their respective impacts.

我们的进化方法。具体来说，Omni-Parser(Lu et al., 2024) 被用作默认的视觉元素提取器，用于定位屏幕上的 UI 组件。在本研究中，我们用具有同等功能的 Ferret-UI(You et al., 2024b) 替代它，并比较基线动作空间与进化动作空间，以检验它们各自的影响。

As shown in Table 3, the Full Action Space significantly improves task performance compared to the Basic Action Space, significantly reducing execution time while simultaneously improving success rates. These results validate the effectiveness of the evolved action space.

如表 3 所示，完整动作空间相比基础动作空间显著提升了任务性能，显著缩短了执行时间，同时提高了成功率。这些结果验证了进化动作空间的有效性。

Qualitative Analysis. In Figures 6, we show a qualitative analysis of some of the actions that used high-level actions to operate the screen. On the left side we label the various processes in the relevant execution operations, from the retrieval of tasks to the matching of shortcut nodes. Despite the different user interfaces and actions performed by these applications, AppAgentX successfully completed the given task.

定性分析。在图 6 中，我们展示了一些使用高级动作操作屏幕的动作的定性分析。左侧标注了相关执行操作中的各个过程，从任务检索到快捷节点匹配。尽管这些应用的用户界面和执行动作不同，AppAgentX 仍成功完成了指定任务。

6 Conclusion

6 结论

We propose an evolving GUI agent framework that enhances efficiency and intelligence by abstracting high-level actions from execution history. Unlike rule-based automation, our approach generalizes task execution by compressing repetitive operations, balancing intelligent reasoning with efficient execution. A chain-based knowledge framework enables continuous behavior refinement, improving adaptability and reducing redundancy. Experiments show our framework outperforms existing methods in accuracy and efficiency.

我们提出了一个进化的 GUI 代理框架，通过从执行历史中抽象出高级动作来提升效率和智能。不同于基于规则的自动化，我们的方法通过压缩重复操作实现任务执行的泛化，平衡智能推理与高效执行。基于链的知识框架支持持续的行为优化，提高适应性并减少冗余。实验表明，我们的框架在准确性和效率上优于现有方法。

References

参考文献

Anthropic. 2024. Claude 3.5 sonnet.

Anthropic. 2024. Claude 3.5 sonnet.

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. 2024. Di-girl: Training in-the-wild device-control agents with autonomous reinforcement learning. Preprint, arXiv:2406.11896.

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. 2024. Di-girl: Training in-the-wild device-control agents with autonomous reinforcement learning. 预印本, arXiv:2406.11896.

Yuxiang Chai, Hanhao Li, Jiayu Zhang, Liang Liu, Guangyi Liu, Guozhi Wang, Shuai Ren, Siyuan Huang, and Hongsheng Li. 2025. A3: Android agent arena for mobile gui agents. Preprint, arXiv:2501.01149.

Yuxiang Chai, Hanhao Li, Jiayu Zhang, Liang Liu, Guangyi Liu, Guozhi Wang, Shuai Ren, Siyuan Huang, and Hongsheng Li. 2025. A3: Android agent arena for mobile gui agents. 预印本, arXiv:2501.01149.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F. Karlsson, Jie Fu, and Yemin Shi. 2024. Autoagents: A framework for automatic agent generation. Preprint, arXiv:2309.17288.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F. Karlsson, Jie Fu, and Yemin Shi. 2024. Autoagents: A framework for automatic agent generation. 预印本, arXiv:2309.17288.

Xiangxiang Chu, Limeng Qiao, Xinyu Zhang, Shuang Xu, Fei Wei, Yang Yang, Xiaofei Sun, Yiming Hu, Xinyang Lin, Bo Zhang, and Chunhua Shen. 2024. Mobilevlm v2: Faster and stronger baseline for vision language model. Preprint, arXiv:2402.03766.

Xiangxiang Chu, Limeng Qiao, Xinyu Zhang, Shuang Xu, Fei Wei, Yang Yang, Xiaofei Sun, Yiming Hu, Xinyang Lin, Bo Zhang, and Chunhua Shen. 2024. Mobilevlm v2: Faster and stronger baseline for vision language model. 预印本, arXiv:2402.03766.

DeepSeek-AI. 2024. Deepseek-v3 technical report. Preprint, arXiv:2412.19437.

DeepSeek-AI. 2024. Deepseek-v3 technical report. 预印本, arXiv:2412.19437.

Hiroki Furuta, Kuang-Huei Lee, Ofir Nachum, Yutaka Matsuo, Aleksandra Faust, Shixiang Shane Gu, and Izzeddin Gur. 2024. Multimodal web navigation with instruction-finetuned foundation models. Preprint, arXiv:2305.11854.

Hiroki Furuta, Kuang-Huei Lee, Ofir Nachum, Yutaka Matsuo, Aleksandra Faust, Shixiang Shane Gu, and Izzeddin Gur. 2024. Multimodal web navigation with instruction-finetuned foundation models. 预印本, arXiv:2305.11854.

Gemini. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. Preprint, arXiv:2403.05530.

Gemini. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. 预印本, arXiv:2403.05530.

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. Preprint, arXiv:2401.13919.

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. 预印本, arXiv:2401.13919.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. Preprint, arXiv:1512.03385.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. 预印本, arXiv:1512.03385.

Theodore D. Hellmann and Frank Maurer. 2011. Rule-Based Exploratory Testing of Graphical User Interfaces. In 2011 Agile Conference, pages 107-116, Los Alamitos, CA, USA. IEEE Computer Society.

Theodore D. Hellmann 和 Frank Maurer. 2011. 基于规则的图形用户界面探索性测试. 载于 2011 年敏捷会议，页 107-116，美国加利福尼亚州洛斯阿拉米托斯. IEEE 计算机学会.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. Preprint, arXiv:2308.00352.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, 和 Jürgen Schmidhuber. 2024. Metagpt: 多智能体协作框架的元编程 (Meta programming). 预印本, arXiv:2308.00352.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. 2023. Co-gagent: A visual language model for gui agents. Preprint, arXiv:2312.08914.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, 和 Jie Tang. 2023. Co-gagent: 面向 GUI 代理的视觉语言模型. 预印本, arXiv:2312.08914.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2022. Large language models can self-improve. Preprint, arXiv:2210.11610.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, 和 Jiawei Han. 2022. 大型语言模型 (Large language models) 能够自我提升. 预印本，arXiv:2210.11610.

LangChain. 2024. Langgraph: A framework for agent-based workflows. Accessed: 2024-06-01.

LangChain. 2024. Langgraph: 基于代理的工作流框架. 访问时间:2024-06-01.

Junkai Li, Yunghwei Lai, Weitao Li, Jingyi Ren, Meng Zhang, Xinhui Kang, Siyu Wang, Peng Li, Ya-Qin Zhang, Weizhi Ma, and Yang Liu. 2025. Agent hospital: A simulacrum of hospital with evolvable medical agents. Preprint, arXiv:2405.02957.

Junkai Li, Yunghwei Lai, Weitao Li, Jingyi Ren, Meng Zhang, Xinhui Kang, Siyu Wang, Peng Li, Ya-Qin Zhang, Weizhi Ma, 和 Yang Liu. 2025. Agent hospital: 具有可进化医疗代理的医院模拟系统. 预印本，arXiv:2405.02957.

Yanda Li, Chi Zhang, Wanqi Yang, Bin Fu, Pei Cheng, Xin Chen, Ling Chen, and Yunchao Wei. 2024. Ap-pagent v2: Advanced agent for flexible mobile interactions. Preprint, arXiv:2408.11824.

Yanda Li, Chi Zhang, Wanqi Yang, Bin Fu, Pei Cheng, Xin Chen, Ling Chen, 和 Yunchao Wei. 2024. Ap-pagent v2: 用于灵活移动交互的高级代理. 预印本，arXiv:2408.11824.

Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, Junjie Gao, Junjun Shan, Kangning Liu, Shudan Zhang, Shuntian Yao, Siyi Cheng, Wentao Yao, Wenyi Zhao, Xinghan Liu, Xinyi Liu, Xinying Chen, Xinyue Yang, Yang Yang, Yifan Xu, Yu Yang, Yujia Wang, Yulin Xu, Zehan Qi, Yuxiao Dong, and Jie Tang. 2024. Autoglm: Autonomous foundation agents for guis. Preprint, arXiv:2411.00820.

Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, Junjie Gao, Junjun Shan, Kangning Liu, Shudan Zhang, Shuntian Yao, Siyi Cheng, Wentao Yao, Wenyi Zhao, Xinghan Liu, Xinyi Liu, Xinying Chen, Xinyue Yang, Yang Yang, Yifan Xu, Yu Yang, Yujia Wang, Yulin Xu, Zehan Qi, Yuxiao Dong, 和 Jie Tang. 2024. Autoglm: 面向图形用户界面的自主基础代理. 预印本，arXiv:2411.00820.

Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. 2024. Omniparser for pure vision based gui agent. Preprint, arXiv:2408.00203.

Yadong Lu, Jianwei Yang, Yelong Shen, 和 Ahmed Awadallah. 2024. Omniparser: 基于纯视觉的 GUI 代理. 预印本，arXiv:2408.00203.

Jiageng Mao, Yuxi Qian, Junjie Ye, Hang Zhao, and Yue Wang. 2023. Gpt-driver: Learning to drive with gpt. Preprint, arXiv:2310.01415.

Jiageng Mao, Yuxi Qian, Junjie Ye, Hang Zhao, 和 Yue Wang. 2023. Gpt-driver: 利用 GPT 学习驾驶. 预印本，arXiv:2310.01415.

Microsoft. 2024. Resnet-50 pre-trained model.

微软. 2024. Resnet-50 预训练模型.

Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. 2024. A comprehensive overview of large language models. Preprint, arXiv:2307.06435.

Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, 和 Ajmal Mian. 2024. 大型语言模型综述. 预印本，arXiv:2307.06435.

Neo4j. 2024. Neo4j: The graph database platform.

Neo4j. 2024. Neo4j: 图数据库平台.

OpenAI. 2024. Gpt-4 technical report. Preprint, arXiv:2303.08774.

OpenAI. 2024. GPT-4 技术报告. 预印本，arXiv:2303.08774.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. Preprint, arXiv:2304.03442.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, 和 Michael S. Bernstein. 2023. 生成代理: 人类行为的交互式模拟. 预印本，arXiv:2304.03442.

Pawel Pawlowski, Krystian Zawistowski, Wojciech La-pacz, Marcin Skorupa, Adam Wiacek, Sebastien Postansque, and Jakub Hoscilowicz. 2024. Tinyclick: Single-turn agent for empowering gui automation. Preprint, arXiv:2410.11871.

Pawel Pawlowski, Krystian Zawistowski, Wojciech Lapacz, Marcin Skorupa, Adam Wiacek, Sebastien Postansque, 和 Jakub Hoscilowicz. 2024. Tinyclick: 用于增强 GUI 自动化的单轮代理. 预印本，arXiv:2410.11871.

Pinecone. 2024. Pinecone: Scalable vector search for ai.

Pinecone. 2024. Pinecone: 用于人工智能的可扩展向量搜索。

Yiran Qin, Enshen Zhou, Qichang Liu, Zhenfei Yin, Lu Sheng, Ruimao Zhang, Yu Qiao, and Jing Shao. 2024. Mp5: A multi-modal open-ended embodied system in minecraft via active perception. Preprint, arXiv:2312.07472.

秦一然，周恩深，刘启昌，尹振飞，盛璐，张瑞茂，乔宇，邵靖。2024。Mp5: 通过主动感知实现的 Minecraft 中的多模态开放式具身系统。预印本，arXiv:2312.07472。

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tya-magundlu, Timothy Lillicrap, and Oriana Riva. 2025. Androidworld: A dynamic benchmarking environment for autonomous agents. Preprint, arXiv:2405.14573.

Christopher Rawles，Sarah Clinckemaillie，Yifan Chang，Jonathan Waltz，Gabrielle Lau，Marybeth Fair，Alice Li，William Bishop，Wei Li，Folawiyo Campbell-Ajala，Daniel Toyama，Robert Berry，Divya Tya-magundlu，Timothy Lillicrap，Oriana Riva。2025。Androidworld: 用于自主代理的动态基准测试环境。预印本，arXiv:2405.14573。

Amanda Ross and Victor L. Willson. 2017. Paired Samples T-Test, pages 17-19. SensePublishers, Rotterdam.

Amanda Ross 和 Victor L. Willson。2017。配对样本 t 检验，第 17-19 页。SensePublishers，鹿特丹。

Zirui Song, Yaohang Li, Meng Fang, Yanda Li, Zhen-hao Chen, Zecheng Shi, Yuan Huang, Xiuying Chen, and Ling Chen. 2025. Mmac-copilot: Multi-modal agent collaboration operating copilot. Preprint, arXiv:2404.18074.

宋子睿，李耀航，方萌，李彦达，陈振浩，石泽成，黄源，陈秀英，陈玲。2025。Mmac-copilot: 多模态代理协作操作副驾驶。预印本，arXiv:2404.18074。

Sharon Tentarelli, Rómulo Romero, and Michelle L. Lamb. 2022. Script-based automation of analytical instrument software tasks. SLAS Technology, 27(3):209-213.

Sharon Tentarelli，Rómulo Romero，Michelle L. Lamb。2022。基于脚本的分析仪器软件任务自动化。SLAS Technology，27(3):209-213。

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024a. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. Preprint, arXiv:2406.01014.

王俊阳，徐海洋，贾海涛，张曦，闫明，沈伟舟，张骥，黄飞，桑吉涛。2024a。Mobile-agent-v2: 通过多代理协作实现高效导航的移动设备操作助手。预印本，arXiv:2406.01014。

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024b. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. Preprint, arXiv:2401.16158.

王俊阳，徐海洋，叶家博，闫明，沈伟舟，张骥，黄飞，桑吉涛。2024b。Mobile-agent: 具备视觉感知的自主多模态移动设备代理。预印本，arXiv:2401.16158。

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. Preprint, arXiv:2201.11903.

Jason Wei，王学志，Dale Schuurmans，Maarten Bosma，Brian Ichter，夏飞，Ed Chi，Quoc Le，Denny Zhou。2023。链式思维提示激发大型语言模型的推理能力。预印本，arXiv:2201.11903。

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2024. Autodroid: Llm-powered task automation in android. Preprint, arXiv:2308.15272.

文浩，李元春，刘国宏，赵善辉，余涛，李家俊，蒋世奇，刘云浩，张雅琴，刘云新。2024。Autodroid: 基于大型语言模型的安卓任务自动化。预印本，arXiv:2308.15272。

Yue Wu, Xuan Tang, Tom M. Mitchell, and Yuanzhi Li. 2024. Smartplay: A benchmark for llms as intelligent agents. Preprint, arXiv:2310.01557.

吴越，唐轩，Tom M. Mitchell，李元志。2024。Smartplay: 作为智能代理的大型语言模型基准测试。预印本，arXiv:2310.01557。

Hui Yang, Sifu Yue, and Yunzhong He. 2023. Auto-gpt for online decision making: Benchmarks and additional opinions. Preprint, arXiv:2306.02224.

杨辉, 岳思福, 何云中。2023。在线决策的 Auto-GPT: 基准测试及附加观点。预印本, arXiv:2306.02224。

Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. 2024a. Ferret-ui: Grounded mobile ui understanding with multimodal llms. Preprint, arXiv:2404.05719.

尤勤, 张昊天, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, 杨银飞, 甘喆。2024a。Ferret-ui: 基于多模态大型语言模型的移动界面理解。预印本，arXiv:2404.05719。

Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. 2024b. Ferret-ui: Grounded mobile ui understanding with multimodal llms. Preprint, arXiv:2404.05719.

尤勤, 张昊天, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, 杨银飞, 甘喆。2024b。Ferret-ui: 基于多模态大型语言模型的移动界面理解。预印本，arXiv:2404.05719。

Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qing-wei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. 2024a. Ufo: A ui-focused agent for windows os interaction. Preprint, arXiv:2402.07939.

张超云，李立群，何世林，张旭，乔博，秦思，马明华，康宇，林庆伟，Saravan Rajmohan，张东梅，张琦。2024a。UFO: 面向 Windows 操作系统交互的界面代理。预印本，arXiv:2402.07939。

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2023. Appagent: Multimodal agents as smartphone users. Preprint, arXiv:2312.13771.

张驰，杨钊，刘家轩，韩宇成，陈昕，黄泽彪，付斌，余刚。2023。Appagent: 作为智能手机用户的多模态代理。预印本，arXiv:2312.13771。

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2024b. Aflow: Automating agentic workflow generation. Preprint, arXiv:2410.10762.

张佳怡，向金玉，于昭阳，滕凤伟，陈雄辉，陈佳琪，诸葛明辰，程鑫，洪思睿，王金林，郑炳楠，刘邦，罗宇宇，吴成林。2024b。Aflow: 自动化代理工作流生成。预印本，arXiv:2410.10762。

A Design and Execution Details

设计与执行细节

This section presents the detailed design of the proposed chain structure and the overall task execution workflow in our system.

本节介绍了我们系统中所提链结构的详细设计及整体任务执行工作流。

A.1 Chain Design Details

A.1 链设计细节

In chain design, there are three types of relationships between nodes:

在链设计中，节点之间存在三种关系类型:

Page-HAS_ELEMENT → Element

页面-HAS_ELEMENT → 元素

Type: HAS_ELEMENT

类型:HAS_ELEMENT

Direction: (Page) -[:HAS_ELEMENT]-> (Element)

方向:(页面)-[:HAS_ELEMENT]->(元素)

Purpose: Indicates that a page contains a specific UI element.

目的: 表示某页面包含特定的 UI 元素。

Shortcut-COMPOSED_OF→Element

快捷方式-COMPOSED_OF→ 元素

Type: COMPOSED_OF

类型:COMPOSED_OF

Direction: (Shortcut) -[:COMPOSED_OF]-> (Element)

方向:(快捷方式)-[:COMPOSED_OF]->(元素)

Relationship Attributes:

关系属性:

- order: Integer, representing the execution sequence.
  - order: 整数，表示执行顺序。
- atomic_action: Type of basic action (e.g., click, text input, etc.).
  - atomic_action: 基本动作类型 (如点击、文本输入等)。
- action_params: JSON format, potentially including input text, click parameters, etc.
  - action_params:JSON 格式，可能包含输入文本、点击参数等。

Purpose: Defines how a composite action consists of certain elements and their corresponding steps, which must be executed in a specific order.

目的: 定义复合动作由某些元素及其对应步骤组成，这些步骤必须按特定顺序执行。

Element-LEADS_TO→Page

元素-LEADS_TO→ 页面

Type: LEADS_TO

类型:LEADS_TO

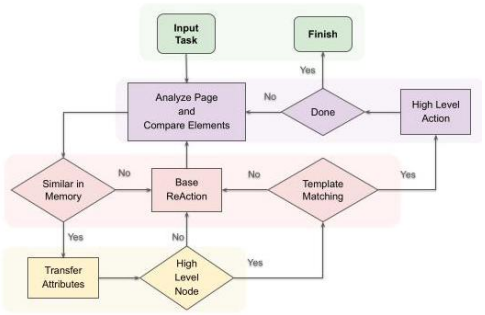

Figure 7: Overall Execution Process. This figure illustrates the flowchart from the input of the task to the final execution result after we then add the shortcut route.

图 7: 整体执行流程。该图展示了从任务输入到最终执行结果的流程图，随后我们添加了快捷路径。

Direction: (Element) -[:LEADS_TO]-> (Page) Purpose: Represents the linkage of a composite action.

方向:(元素)-[:LEADS_TO]->(页面) 目的: 表示复合动作的关联。

Together, these types of edges, along with the nodes described in the main text, constitute the structure of the complete memory store. This modeling approach aligns with the relationships between page jumps and containment, and offers a solid foundation for analyzing the evolution of actions.

这些类型的边与正文中描述的节点共同构成了完整记忆存储的结构。该建模方法符合页面跳转与包含关系，为分析动作演变提供了坚实基础。

A.2 Execution Process Details

A.2 执行过程细节

Figure 7 presents a detailed flowchart that delineates the task processing workflow. The primary components of this workflow include page analysis, element comparison, matching, and memory storage. As depicted in Figure 7, the system utilizes a hierarchical action framework comprising high-level nodes and basic action spaces. During

the operation, if the matching retrieval process of a high-level node fails, the system seamlessly transitions to the corresponding basic action spaces.

图 7 展示了详细的流程图，描述了任务处理工作流。该工作流的主要组成包括页面分析、元素比较、匹配及记忆存储。如图 7 所示，系统采用包含高层节点和基础动作空间的分层动作框架。运行过程中，若高层节点的匹配检索失败，系统会无缝切换至相应的基础动作空间。

B Detailed Numerical Results

B 详细数值结果

This section provides the numerical analysis results of various shortcut designs, including evaluations of robustness and statistical tests to validate the effectiveness of the proposed system design.

本节提供了各种快捷设计的数值分析结果，包括鲁棒性评估和统计检验，以验证所提系统设计的有效性。

To supplement the grouped bar chart presented in Figure 4, this subsection provides the exact per-step execution time (in seconds) for each evaluated framework (MobileAgent2, AppAgent, and Ap-pAgentX) across different LLM backends. These numerical results are intended to enhance reproducibility, facilitate further analysis, and offer a precise basis for comparing execution efficiency across systems.

为补充图 4 中的分组条形图，本小节提供了不同 LLM 后端下各评估框架 (MobileAgent2、AppAgent 和 AppAgentX) 每步执行时间 (秒) 的精确数值。该数值结果旨在增强可复现性，便于进一步分析，并为系统间执行效率比较提供精确依据。

Table 4: Comparison of Average Execution Time per Step. This table presents the average execution time (seconds ↓) across different LLMs and frameworks.

表 4: 每步平均执行时间比较。该表展示了不同 LLM 和框架下的平均执行时间 (秒 ↓)。

LLM	Previous SOTA	AppAgent	Ours
Gemini-1.5-Pro	25.6	15.2	12.4
Claude-3.5	38.4	13.8	11.4
GPT-40	43.5	20.3	17.5

大型语言模型 (LLM)	之前的最先进技术 (SOTA)	应用代理 (AppAgent)	我们的模型
双子座 1.5 专业版 (Gemini-1.5-Pro)	25.6	15.2	12.4
Claude-3.5	38.4	13.8	11.4
GPT-40	43.5	20.3	17.5

Table 4 lists the detailed values corresponding to each bar in the figure.

表 4 列出了图中每个柱状对应的详细数值。

Table 5: Statistical Test Results for Task Completion Times. Results from one-tailed paired t-tests.

表 5: 任务完成时间的统计检验结果。单尾配对 t 检验结果。

Task Length	Mean Difference (s)	t-value	p-value
Short Task	-12.4	-3.45	<0.01
Medium Task	-30.7	-5.12	<0.001
Long Task	-75.3	-6.89	<0.001

任务时长	平均差异 (秒)	t 值	p 值
短任务	-12.4	-3.45	<0.01
中等任务	-30.7	-5.12	<0.001
长任务	-75.3	-6.89	<0.001

To reinforce the observed differences in task completion times, we complemented the analysis with detailed results from one-tailed t-tests, as summarized in Table 5. These results address data not fully elaborated upon in the main text. The negative t-values reported in this table indicate the presence of a directional effect, where AppAgentX consistently outperforms the baseline in reducing task completion times. Specifically, the negative sign reflects that the mean completion time for AppA-gentX is significantly lower than that of the baseline across all task lengths. This directional result aligns with our hypothesis and further validates the consistent efficiency advantage of AppAgentX.

为了强化观察到的任务完成时间差异，我们补充了单尾 t 检验的详细结果，汇总于表 5 中。这些结果涉及主文中未充分展开的数据。表中报告的负 t 值表明存在方向性效应，即 AppAgentX 在缩短任务完成时间方面始终优于基线。具体而言，负号反映出 AppAgentX 的平均完成时间显著低于基线，且适用于所有任务长度。该方向性结果与我们的假设一致，进一步验证了 AppAgentX 持续的效率优势。

C Additional Quantitative Analysis

C 额外的定量分析

To further illustrate the behavior of our system in real-world settings, we include a qualitative example in Figure 8. This example showcases the execution path for a task in Apple Music, highlighting how AppAgentX leverages existing shortcuts to complete the task more efficiently. By using predefined shortcuts, the system reduces the reliance on large language model (LLM) reasoning at each step, thereby improving both speed and resource usage.

为了进一步展示我们系统在实际环境中的表现，我们在图 8 中加入了一个定性示例。该示例展示了 Apple Music 中一个任务的执行路径，突出显示了 AppAgentX 如何利用现有快捷方式更高效地完成任务。通过使用预定义的快捷方式，系统减少了每一步对大型语言模型 (LLM) 推理的依赖，从而提升了速度和资源利用率。

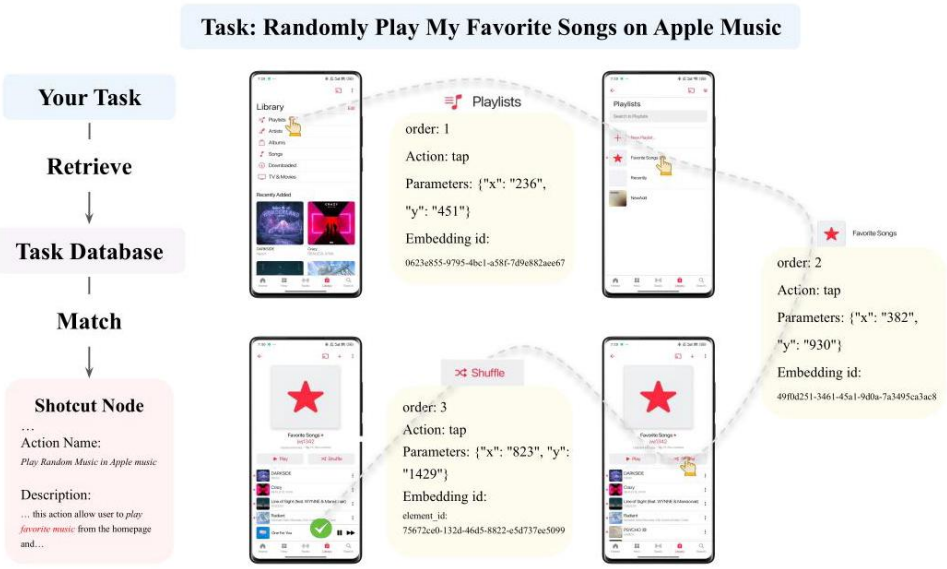

Figure 8: Qualitative Results. This figure presents the path execution utilizing shortcuts on Apple Music. It demonstrates the efficiency of AppAgentX in decreasing the utilization of LLM for per-step reasoning.

图 8: 定性结果。该图展示了在 Apple Music 上利用快捷方式的路径执行，体现了 AppAgentX 在减少每步推理中 LLM 使用的效率。