CHOP: Mobile Operating Assistant with Constrained High-frequency Optimized Subtask Planning

Anonymous ACL submission

Abstract

The advancement of visual language models (VLMs) has enhanced mobile device operations, allowing simulated human-like actions to address user requirements. Current VLM-based mobile operating assistants can be structured into three levels: task, subtask, and action. The subtask level, linking high-level goals with low-level executable actions, is crucial for task completion but faces two challenges: **ineffective subtasks** that lower-level agent cannot execute and **inefficient subtasks** that fail to contribute to the completion of the higher-level task. These challenges stem from VLM's lack of experience in decomposing subtasks within GUI scenarios in multi-agent architecture. To address these, we propose a new mobile assistant architecture with **c**onstrained **h**igh-frequency **o**ptimized **p**lanning (CHOP.) Our approach overcomes the VLM's deficiency in GUI scenarios planning by using human-planned subtasks as the "basis vector". We evaluate our architecture in both English and Chinese contexts across 20 Apps, demonstrating significant improvements in both effectiveness and efficiency. Our dataset and code is available at `https://anonymous.4open.science/r/CHOP_-CFEA`.

1 Introduction

Mobile operating assistants (Wang et al., 2024c; Zhang et al., 2024a; Nguyen et al., 2024; Hu et al., 2024) automate mobile App control by simulating human actions like clicking or typing. These assistants are widely used in recommendation (Sun et al., 2022), task automation (Liu et al., 2024), and user assistance (Zhang et al., 2023; Wang et al., 2024a; Zhu et al., 2024). Early assistants, based on slot-filling and neural networks (Sun et al., 2022; Zhang and Zhang, 2023; Zhu et al., 2023), struggle with generalization. LLMs (OpenAI, 2021) improve this through multitask learning and cross-domain integration (Brown et al., 2020), while

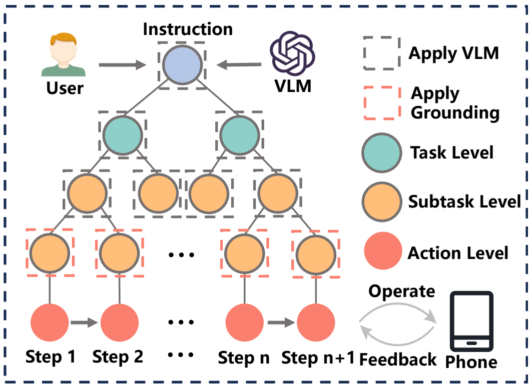


Figure 1: Execution flowchart for VLM-based assistant.

VLMs (Yang et al., 2024a; OpenAI, 2023) advance assistants by incorporating visual processing, making them the dominant approach in modern mobile environments (Wang et al., 2024c; Zhang et al., 2024a; Nguyen et al., 2024; Hu et al., 2024).

In mobile App operations, we structure the VLM-based assistant architecture into three hierarchical levels: **instruction**, **task**, and **action**, as shown in Figure 1. An **instruction** refers to the original user command, which may involve interactions across multiple apps (e.g., "Send Bob the playlist and ask for his feedback".) To execute such instructions, the system decomposes them into one or more **tasks**, each corresponding to operations within a single app (e.g., "Play Bob's songs" in a music app.) Each task is further divided into **subtasks** (Zhu et al., 2024), which are context-specific steps (e.g., "Search Bob" on the music app's interface,) and finally into atomic **actions** (Lin et al., 2024; Yang et al., 2024b) (e.g., tapping the search bar.) This hierarchical architecture enables the assistant to coordinate modules at different granularities to complete complex instructions.

Although recent work in mobile assistants has attempted to improve subtask execution success by constraining the granularity of task decomposition (Zhu et al., 2024), subtask-level operations still face two main challenges: (1) **Ineffective sub-**

tasks, where the subtask cannot be executed due to the VLM's lack of real-world knowledge (Ahn et al., 2022). For instance, "Go to Bob's office" in response to "Ask Bob to attend the meeting" is unachievable, whereas "Send Bob an email" is more feasible. (2) **Inefficient subtasks**, where sequential actions unnecessarily delay task completion without contributing to progress. For example, "Wait for Bob's feedback" stalls the task without advancing it. These challenges stem from VLM's lack of experience in decomposing sub-tasks within GUI scenarios in multi-agent frameworks.

To address these challenges, we propose CHOP (**C**onstrained **H**igh-frequency **O**ptimized Subtask **P**lanning,) a method that optimizes subtask planning by using basis subtasks as constraints during task decomposition. Specifically, in GUI scenarios, the same subtasks across different Apps share common operational logic, allowing users to quickly adapt to new Apps. This allows us to collect such subtasks and apply them to the task decomposition of the plan agent, meaning any task can be decomposed into a combination of "basis subtasks", inspired by "basis vectors". Meanwhile, we ensure the orthogonality of different basis subtasks by merging similar subtasks (Wu et al., 2024). Furthermore, to better leverage the fixed-flow nature of basis subtasks, we provide documentation for each subtask to enhance effectiveness and allow the action agent to generate multiple steps in a single forward pass, thereby improving efficiency.

We evaluate CHOP in both English and Chinese contexts. CHOP-En, the English dataset, is based on Mobile-Agent-V2 (Wang et al., 2024a), covering 10 apps with three difficulty levels each. To extend this work to a broader linguistic context, we introduce CHOP-ZH, **the first Chinese dataset with user planning processes.** CHOP-ZH is created by hiring 10 annotators to complete 200 daily usage instructions across 10 apps, with annotators providing a plan and reasoning for each action. This allows us to evaluate the quality of the subtasks generated by the agent. We assess CHOP in terms of both effectiveness and efficiency, introducing new metrics to measure the inference cost of the action agent, grounding model, and overall architecture. Experimental results show that CHOP achieves state-of-the-art (SOTA) performance, outperforming mainstream VLM-based assistants.

Our summarized contributions are as follows: (1) We propose a new architecture, CHOP, which introduces "basis subtasks" for the first time and addresses the lack of planning capability in VLMs for GUI scenarios. (2) We construct the first Chinese dataset with user planning processes and introduce three new metrics for evaluating efficiency. (3) CHOP achieves SOTA performance on both English and Chinese datasets, with experimental results showing it generates higher-quality subtasks.

2 Related Work

GUI Agent. GUI agents have evolved from rule-based control to multimodal and reasoning-driven approaches. Early methods rely on predefined scripts but struggle in dynamic environments (Li et al., 2017, 2019). Multimodal pre-trained models enabled end-to-end learning by integrating dialogue, screenshots, and operation history (Bai et al., 2021; Burns et al., 2022, 2024; Li and Li, 2023; Li et al., 2021; Wang et al., 2021; Sun et al., 2022). With the rise of VLMs, recent agents incorporate complex reasoning and tool use (Qu et al., 2025, 2024; Qu et al.), often leveraging view hierarchies for efficient UI grounding (Lee et al., 2024; Zhang et al., 2024b, 2023). Image-only approaches handle hierarchy-free settings but remain brittle in dynamic environments (Hong et al., 2024b; Wang et al., 2024a; Zhu et al., 2024; Zhang et al., 2024c). While end-to-end agents (Hong et al., 2024b) unify perception, reasoning, and execution, their performance heavily depends on backbone scale. In contrast, modular agents (Wang et al., 2024a; Zhu et al., 2024; Zhang et al., 2023) offer better control and clearer attribution of improvements. We build on this line by incorporating structured human planning without model fine-tuning.

Multi-agent Application. LLMs possess strong comprehension and reasoning abilities, enabling LLM-based agents to autonomously execute tasks (Wang et al., 2024b; Guo et al., 2024). Inspired by human collaboration, multi-agent frameworks are widely adopted, such as Smallville (Park et al., 2023) and role-playing-based frameworks (Li et al., 2023). Recent advances include expert-agent coordination (Chen et al., 2024), meta-programming (Hong et al., 2024a), and multi-agent debating (Chan et al., 2024). In GUI agents, multi-agent frameworks (Wang et al., 2024a; Zhu et al., 2024) often involve a plan agent for task planning, an action agent for interaction, and a grounding model that maps outputs to executable commands. However, these methods focus on introducing new modules while overlooking coordination among

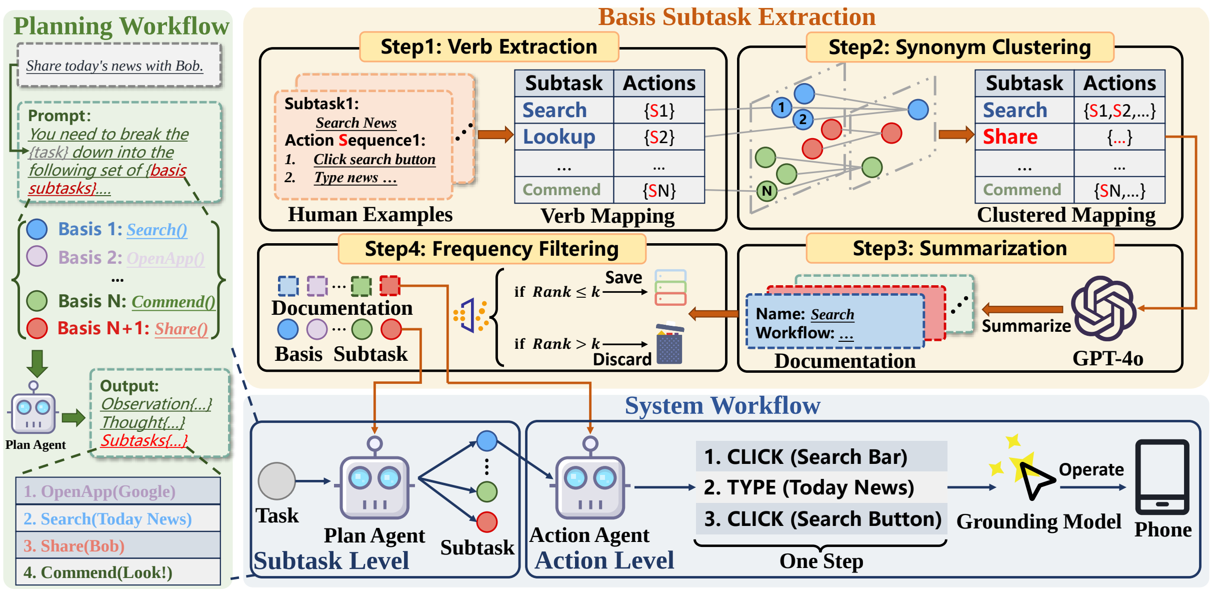


Figure 2: Illustration of the VLM-based GUI assistant framework with basis subtask extraction.

modules. Moreover, although Moba (Zhu et al., 2024) also considers decomposing tasks multiple times to ensure the generated subtasks can be executed by the action agent, the issues of ineffective and inefficient subtasks we mentioned still persist. Instead, we propose constraining subtask-level outputs to improve executability by action-level agents and better facilitate task-level goals.

3 Method

CHOP is an end-to-end pipeline that executes user instructions on real-world mobile devices, similar to (Zhang et al., 2023; Wang et al., 2024a; Zhu et al., 2024). As shown in Figure 2, we present the CHOP and the extraction processes of its basis subtasks. § 3.1 first introduces the problem setup and environment construction. Then, § 3.2 outlines the extraction of basis subtasks used in task decomposition. Finally, § 3.3 describes how CHOP integrates basis subtasks into its architecture, which consists of both the plan agent for task decomposition and the action agent for executing actions.

3.1 Problem Setup

A mobile operating task consists of a screen s and an instruction q (e.g., "Send an email to Bob".) Given a tuple (s, q), a mobile operating assistant f decides and performs a sequence of actions $\mathbf{a} = \{a_1, a_2, \dots, a_t, \dots\}$ to interact with the Android environment $\mathcal{E}$ on the mobile device. This task execution is modeled as a sequential decision-making process. The formal definitions of the action and state spaces are as follows:

Action Type	Attributes	Description
`CLICK`	(x, y): Screen coordinates	Click at an element
`SCROLL`	(direction): One of up, down, left, right	Scroll the page
`TYPE`	(text): Text input	Type text
`BACK`	-	Back to previous page
`EXIT`	-	Task complete
`WAIT`	(time): Wait time in seconds	Stop for a while

Table 1: The supported action space for CHOP.

Action Space A: We define an action as a function call (Niu et al., 2024). When the assistant outputs an action in the required format, it is parsed and executed by the environment. This includes various action types such as click, scroll, and type. Table 1 provides a detailed list of action types and their corresponding attributes. **State Space** S: Since CHOP is an image-only architecture, it does not use textual information such as XML to assist decision-making. Instead, the state space is defined solely by the current screenshot s_t, which represents the environment at time step t.

At each time step t, the assistant selects an action a_t based on the current state s_t and the accumulated history $H_t = \{s_0, a_0, \dots, s_{t-1}, a_{t-1}\}$, as determined by the policy function: $a_t = f(s_t, H_t)$. The action a_t leads to a state transition, where the Android environment $\mathcal{E}$ updates the state from s_t to s_{t+1} by the transition function T, reflecting the environmental changes resulting from the action: $s_{t+1} = T(s_t, a_t)$. At the same time, the history H_t is updated to incorporate the most recent action a_t and the previous state s_{t-1}, which results in: $H_{t+1} = \text{concat}(H_t, s_{t-1}, a_t)$.

In summary, the decision-making process begins with the initial state S_0, which represents the homepage of the mobile phone, and the initial history H_0, which is empty at the start. The assistant then proceeds by iterating through the policy f and the transition function T, selecting an action at each time step t and updating the state s_t and history H_t. This continues until the action is `EXIT` or the maximum number of rounds is reached.

3.2 Basis Subtask Extraction

We first find two issues with subtask generation in the current multi-agent architecture: **Ineffective subtasks** and **Inefficient subtasks**. To address these issues, ideal subtasks should meet two criteria: **High Effectiveness** – Executable by the action agent: The plan agent must generate subtasks that the action model can execute (Ahn et al., 2022). **High Efficiency** – On the critical path: Any missing subtasks should lead to task failure, ensuring they are essential for task completion.

Inspired by human task planning (Correa et al., 2023), where individuals typically break down tasks based on familiar operations rather than methods perceived as the most efficient but not necessarily aligned with the individual's familiar operations, we introduce basis subtasks—high-frequency subtasks commonly performed by humans. These subtasks enhance effectiveness (as they are familiar to humans due to their frequent use, making them easier to execute) and efficiency (since they are typically on the critical path of the task.)

Specifically, given the high cost of manually annotated data and the expensive fine-tuning of VLMs (Lai et al., 2024), rather than training a new model, we focus on directly collecting these common subtasks from human-executed app commands to construct a "basis subtask" space. The collection process consists of four steps: Verb Extraction, Synonym Clustering, Summarization, and Frequency Filtering (Figure 2.) **Clustering ensures that each basis subtask independently handles different task types, while filtering makes these "basis subtasks" easier to execute than others.** In summary, such subtasks can be seen as "basis vectors". Any task can be decomposed into a combination of independent basis subtasks, with their fixed nature enabling easier handling.

Verb Extraction. We extract verbs from user instructions in the AITZ dataset (Zhang et al., 2024c) using *spaCy* to represent the core action of each subtask. **Synonym Clustering.** We group semantically similar verbs using WordNet synsets to reduce redundancy in subtask representation. **Summarization.** We use GPT-4 to summarize action sequences of each subtask into standardized processes for consistency across GUIs. **Frequency Filtering.** Due to the long-tail distribution of subtasks, we retain the top 10 most frequent basis subtasks, which together cover over 80% of user instructions in the AITZ dataset. This ensures that common and essential operations are efficiently captured, while rare cases are handled by prompting GPT-4 to generate task-specific subtasks outside this fixed set. Detailed procedures can be found in Appendix A.

All the basis subtasks can be found in Table 10 in the Appendix. An example of a basis subtask and corresponding documentation is provided below:

A Basis Subtask with Documentation

Basis subtask: Search Item (parameter: search term)
Standardized process: 1. Click on the search bar located at the designated area of the screen. 2. Type in the content specified by the search term parameter. 3. If applicable, select a search suggestion from the dropdown list that appears after typing. 4. Press enter or click on the search button to execute the search.
Boundary conditions: 1. If the search term is not found, check for spelling errors. 2. If selecting a suggestion, ensure it is the correct item before proceeding. 3. If navigating to a specific website, ensure the URL is entered correctly in the address bar.

3.3 CHOP: The Multi-Agent Architecture

To guide the assistant f in multi-step tasks, VLMs (OpenAI, 2023; Yang et al., 2024a) are a strong candidate due to their visual understanding in mobile environments. However, applying VLMs to real-world screenshots with thousands of tokens is inefficient. Recent work (Zhu et al., 2024) uses a two-stage architecture: decomposing tasks into subtasks and executing them, reducing sequence length, and improving accuracy (Wang et al., 2024a). However, without subtask constraints, ineffective and inefficient subtasks arise. To address these issues, we introduce basis subtasks during planning and limit outputs to predefined tasks, which incorporate human-designed heuristics to overcome VLM's limitations in GUI scenarios. The process is described below.

The Plan Agent. Given a user instruction q, the plan agent f_{plan} decomposes it into a sequence of subtasks, each executable by the action agent:

$$\{q_1, q_2, ..., q_n\} = f_{\text{plan}}(q, Q_{\text{basis}})$$

where Q_{basis} is the set of predefined basis subtasks, and each q_i must be selected from it. To enhance execution, the plan agent also generates the purpose and stopping condition for each subtask. If a necessary subtask is missing from Q_{basis}, a placeholder is used, prompting the model to define, structure, and refine new subtasks as needed. This ensures all generated subtasks are well-defined, actionable, and contribute effectively to task completion.

The Action Agent. For each subtask q_i, the action agent f_{action} determines the next executable action. At step t, it generates an action a_{t+1} based on the user task q, the current subtask q_i, the execution documentation d_i, the current screenshot s_t, and the accumulated summary memories $\mathbf{m} = \{m_1, \ldots, m_{i-1}\}$. The selected action is then executed, updating the environment state:

$$a_{t+1} = f_{\text{action}}(q, q_i, d_i, s_t, \mathbf{m},)$$

$$s_{t+1} = T(s_t, a_{t+1}.)$$

To guide the execution of these actions, the agent generates observation, thought, and summarization. The summarization extracts key task-related details, such as weather information for the subtask "Check today's weather", which is stored as memory m_t for future tasks. Since VLMs output actions like `CLICK` without coordinates, we integrate Aria-UI (Yang et al., 2024b) to map these commands to precise locations (e.g., `CLICK`(Search Bar) $\rightarrow$ `CLICK`(200, 300.)) To improve efficiency, d_i provides standardized execution steps, and for basis subtasks with fixed workflows (e.g., "Search item",) the agent generates the full action sequence in one step, minimizing latency and reducing the need for multiple action agent calls, which are a key source of computational bottleneck.

4 Experiments

In this section, we evaluate the performance of CHOP by answering the following research questions: **RQ1:** Can the basis subtask improve overall task performance? **RQ2:** Can the basis subtask enhance the quality of task planning? **RQ3:** Can the basis subtask improve performance under certain conditions? RQ1 investigates whether adding the basis subtask constraint improves the execution of user instructions. RQ2 examines how the basis subtask affects the quality of subtasks generated by the plan agent. RQ3 analyzes the conditions under which the basis subtask demonstrates effectiveness in real-world, complex environments.

4.1 Settings

Test set. We evaluate our method using two real-life scenario test datasets: **CHOP-En** and **CHOP-ZH**. The CHOP-En dataset, constructed based on a publicly available benchmark (Wang et al., 2024a), consists of 30 English-language instructions designed to test operating assistants in real-world mobile applications. It covers 10 widely used Apps with tasks of varying difficulty levels: easy, medium, and difficult. The CHOP-ZH dataset consists of 200 Chinese instructions across 10 Apps, with 20 instructions per app. This is the first real-life Chinese test set for mobile devices. In addition to instruction-action pairs, it enables a deeper evaluation of task decomposition. Due to resource constraints, we sample 3 instructions per app, as in CHOP-En. Further details on the dataset and the annotation procedure can be found in Appendix B.

Baselines. To evaluate our method, we compare it with several baseline approaches, including the Human Baseline and agent-based automation methods. **Human Baseline** represents the ideal solution, reflecting the best performance achieved by a human. **AppAgent** (Zhang et al., 2023) employs an exploration-deployment framework where the agent learns app functions and uses these to plan and select actions. **Mobile Agent(v2)** (Wang et al., 2024a) is a multi-agent system that integrates planning, decision-making, and reflection agents for mobile task automation, using screenshots and additional models like OCR and Qwen-VL-Plus. **Moba** (Zhu et al., 2024) uses a two-level agent architecture (Global Agent and Local Agent,) combining visual inputs and XML view hierarchy data for task planning and action execution. Detailed descriptions can be found in the Appendix C.

Evaluation Metrics. To evaluate agent performance, we define two action sequences for each task q: $\mathbf{a}^q_{\text{human}} = \{a_1, \ldots, a_n\}$ (human actions) and $\mathbf{a}^q_{\text{agent}} = \{a_1, \ldots, a_m\}$ (agent actions,) with lengths n and m. Based on these, we assess **Effectiveness** and **Efficiency**. **Effectiveness.** We use two metrics. **Success Rate (SR)** denotes the proportion of tasks successfully completed within 20 actions. **Completion Rate (CR)** (Zhu et al., 2024) measures the ratio of correctly executed steps, using human actions as ground truth:

$$\mathbf{CR} = \frac{\sum_{q \in Q} \left|\mathbf{a}^q_{\text{human}} \cap \mathbf{a}^q_{\text{agent}}\right|}{\sum_{q \in Q} \left|\mathbf{a}^q_{\text{human}}\right|}.$$

Efficiency. We consider three metrics. **Mapping Efficiency (ME)** quantifies how efficiently the agent generates action sequences:

$$\mathbf{ME} = \frac{\sum_{q\in Q} \left|\mathbf{a}^q_{\text{human}}\right|}{\sum_{q\in Q} C_a}.$$

Action Efficiency (AE) compares the lengths of human and agent action sequences:

$$\mathbf{AE} = \frac{\sum_{q\in Q} \left|\mathbf{a}^q_{\text{human}}\right|}{\sum_{q\in Q} \left|\mathbf{a}^q_{\text{agent}}\right|}.$$

Average API Cost (AAC) reflects the overall API usage per correctly completed human action, including calls from planning, memory, and reflection modules (Zhu et al., 2024; Wang et al., 2024a):

$$\mathbf{AAC} = \frac{\text{API}_{\text{count}}}{\sum_{q\in Q} \left|\mathbf{a}^q_{\text{human}} \cap \mathbf{a}^q_{\text{agent}}\right|}.$$

Experimental Setup. All experiments are conducted using the `GPT-4o` model version to ensure a fair comparison. The maximum output length is set to 4096, and the temperature during generation is set to 0.0 to ensure reproducibility. The starting point for all instruction executions is set to the Homepage to ensure consistent evaluation. Due to the Moba method requiring additional tools to open the app, which are not available in our dataset, we use Aria-UI to handle app launching, as it ensures 100% accuracy. Unless specified, we will use CHOP-CH for the analysis experiments.

4.2 RQ1: Task Performance Improvement

Main Results. In RQ1, we investigate whether incorporating the basis subtask Q_{basis} and corresponding documentation D_{basis} into the plan agent's subtask generation improves the effectiveness and efficiency of CHOP. The main results are shown in Table 2, with human-executed trajectories serving as the ground truth. We compare CHOP with mainstream methods and draw the following conclusions:

(1) CHOP achieves the highest effectiveness: CHOP outperforms other methods in **SR** and **CR** across most instruction sets. However, Mobile Agent(v2) outperforms CHOP on the Hard part of the Chinese dataset, likely due to CHOP's use of English documentation. **(2) CHOP demonstrates superior efficiency:** By generating multi-actions in one step for specific basis subtasks, CHOP achieves the best **ME** performance. It minimizes model calls with a single request to the plan agent. The high **AAC** confirms CHOP's efficiency, using the fewest API calls and reducing resource consumption. **(3) Other methods show a trade-off between effectiveness and efficiency:** Mobile Agent(v2) offers comparable performance but requires at least three API calls per action, limiting practicality. AppAgent and Moba, though less efficient, perform well with good resource utilization.

Ablation Study. We draw two key conclusions from our experiments in Table 3 on removing documentation and the basis subtask constraint during subtask generation.

(1) Removing documentation and the basis subtask both reduce performance, highlighting the importance of these components. Specifically, experiments show that CHOP's performance decreases when documentation is excluded, and performance worsens further without the basis subtask. Additionally, CHOP's **AE** score drops, likely due to the variants adopting simpler behaviors (e.g., searching for contacts directly instead of clicking avatars,) requiring fewer actions. **(2) The basis subtask improves CHOP's performance even on out-of-domain Apps, demonstrating its generalizability.** Although basis subtasks are collected from AITW (which includes four app types,) experiments on both in-domain (same app types) and out-of-domain datasets show that the basis subtask benefits performance across both. This supports the idea that similar subtasks across Apps share common operational logic. Furthermore, compared to AppAgent which collects whole-app documentation, our approach reduces size to the subtask level, improving generalization and data efficiency.

4.3 RQ2: Task Planning Improvement

Subtask Evaluation To better assess the quality of generated subtasks, we adopt two complementary evaluation approaches: **(1) Token-Level Matching:** We use BLEU (Papineni et al., 2002) and ROUGE-L (Lin, 2004) to measure the overlap between generated subtasks and human annotations in CHOP-CH. While these metrics offer a straightforward comparison, they are limited in capturing semantic equivalence—small token-level changes can lead to large score variations without meaningfully affecting the execution quality. **(2) LLM-Based Evaluation:** To overcome these limitations, we further employ GPT-4o to compare the semantic quality of the generated subtasks. For each in-

Language	Model	Easy					Medium					Hard				
		Effectiveness		Efficiency			Effectiveness		Efficiency			Effectiveness		Efficiency		
		SR↑	CR↑	ME↑	AE↑	AAC↓	SR↑	CR↑	ME↑	AE↑	AAC↓	SR↑	CR↑	ME↑	AE↑	AAC↓
English	Human	1.00	1.00	1.00	1.00	-	1.00	1.00	1.00	1.00	-	1.00	1.00	1.00	1.00	-
	AppAgent	0.50	0.62	0.84	0.84	1.19	0.40	0.64	0.80	0.80	1.25	0.10	0.22	0.99	0.99	1.01
	Mobile Agent(v2)	0.50	0.81	0.83	0.83	3.62	0.50	0.73	0.82	0.82	3.65	0.40	0.41	0.68	0.68	4.42
	Moba	0.50	0.69	0.97	0.97	1.07	0.30	0.50	0.99	**0.99**	1.04	0.20	0.46	0.98	0.98	1.05
	Ours	**0.80**	**0.90**	**1.36**	**1.00**	**0.76**	**0.70**	**0.89**	**1.20**	0.94	**0.85**	**0.60**	**0.59**	**1.10**	**1.00**	**0.93**
Chinese	Human	1.00	1.00	1.00	1.00	-	1.00	1.00	1.00	1.00	-	1.00	1.00	1.00	1.00	-
	AppAgent	0.40	0.56	0.78	0.78	1.28	0.30	0.51	1.07	0.78	1.29	0.20	0.41	0.96	**0.96**	1.04
	Mobile Agent(v2)	0.80	0.75	0.70	0.70	4.26	0.20	0.46	1.00	0.87	3.44	**0.30**	0.51	0.76	0.70	4.31
	Moba	0.40	0.61	0.90	0.90	1.14	0.30	0.75	0.95	0.84	1.22	0.10	0.35	0.85	0.85	1.23
	Ours	**1.00**	**1.00**	**1.30**	**0.95**	**0.79**	**0.80**	**0.95**	**1.10**	**0.95**	**0.93**	0.10	**0.59**	**1.09**	0.93	**0.95**

Table 2: Performance evaluation of different GUI agents on English and Chinese tasks, categorized by difficulty. Metrics include effectiveness (**S**uccess **R**ate, **C**ompletion **R**ate) and efficiency (**M**apping **E**fficiency, **A**ction **E**fficiency, **A**verage **API** **C**ounts), with human as the baseline. Best results are bolded, and second-best are underlined.

Model	All (10 Apps)				In-domain (4 Apps)				Out-of-domain (6 Apps)			
	Effectiveness		Efficiency		Effectiveness		Efficiency		Effectiveness		Efficiency	
	SR↑	CR↑	ME↑	AE↑	SR↑	CR↑	ME↑	AE↑	SR↑	CR↑	ME↑	AE↑
CHOP	**0.67**	**0.85**	**1.15**	0.91	**0.75**	**0.92**	**1.31**	0.92	**0.61**	**0.83**	**1.03**	0.80
CHOP *w/o* D_{basis}	0.47	0.74	1.00	**1.00**	0.50	0.73	**1.00**	**1.00**	0.44	0.76	1.00	**1.00**
CHOP *w/o* $Q_{\text{basis}}\&D_{\text{basis}}$	0.33	0.57	1.00	**1.00**	0.50	0.59	1.00	**1.00**	0.22	0.56	1.00	**1.00**

Table 3: Ablation study on CHOP-ZH comparing the full method with two variants: one excluding the documentation D_{basis} (CHOP *w/o* D_{basis}) and the other excluding both the basis subtask Q_{basis} and D_{basis} (CHOP *w/o* $Q_{\text{basis}}\&D_{\text{basis}}$). Experiments are conducted on three app sets: All (10 Apps), In-domain (4 Apps, where Q_{basis} is collected), and Out-of-domain (6 Apps). The best results are bolded, second-best underlined.

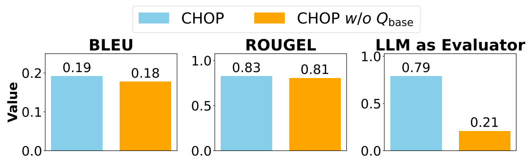


Figure 3: Subtask quality comparison with and without basis subtask on matching and LLM-based evaluation.

struction, we compare two versions of the subtask plan—one generated with the basis subtask and one without—based on three criteria: completeness (whether the subtasks collectively fulfill the instruction), efficiency (whether redundant or irrelevant steps are avoided), and effectiveness (whether each step is executable and clearly articulated). This evaluation reflects both semantic soundness and practical executability. To reduce token order and position bias (Dai et al., 2024), we randomly shuffle the order of comparison and report the win rates. The full evaluation prompt and format are provided in Appendix D.

As shown in Figure 3, both evaluation methods consistently show that adding the basis subtask constraint leads to higher-quality subtask plans.

Case Study. The plan agent is not only tasked with generating basis subtasks but also has the flexibility to create custom subtasks when the basis subtask is unavailable. As demonstrated in Appendix E, we present two examples showing the task and its corresponding subtasks. These examples highlight that, in addition to effectively selecting basis subtasks, our method CHOP can also generate high-quality custom subtasks that effectively complement the basis subtasks. In addition, we also demonstrate with two examples that adding the constraint of basis subtasks can address the issues of ineffective and inefficient subtasks.

4.4 RQ3: Conditions for Improvement

Improvement on Various App. **RQ3** analyzes which tasks benefit most from the basis subtask. We first calculate the **CR** metric for all methods across 10 different application categories. As shown in Figure 4, our method consistently achieves a high **CR** across various applications. In contrast, other methods like AppAgent struggle with app types such as Shopping and Map due to XML parsing issues, while our vision-based method bypasses this problem.

Improvement on Complex Instruction. We also measure **SR** on instructions of varying complexity, defined by step count. As shown in Fig-

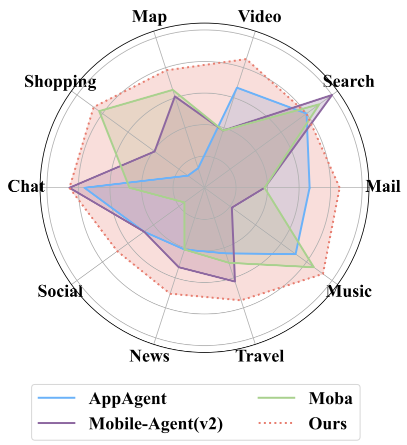


Figure 4: Performances of CHOP with other methods.

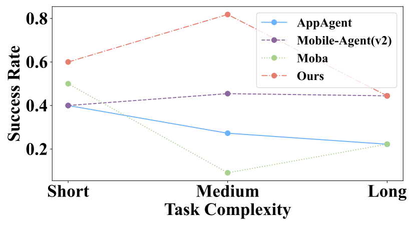


Figure 5: **SR** of different methods across tasks of varying complexities, where complexity is defined by task length, with segments based on consecutive echo points.

ure 5, we group instructions into three length segments. The results show that our method performs particularly well with short and medium-length instructions, with the largest improvement seen in medium-length tasks. However, the improvement is smaller for both short and long instructions. For short instructions, the bottleneck seems to lie outside task planning, likely in visual capabilities. For long instructions, the challenge is the higher requirement for successful subtask decomposition, but our method still outperforms others.

Error on Different Types. As shown in Table 4, we analyze failure reasons for various methods following the settings in (Lai et al., 2024). Both AppAgent and Moba depend on XML files, so XML parsing errors lead to failures, while text-based output parsing errors also contribute. We categorize these as "XML/Model Output Parse Error." AppAgent is most affected by XML parsing, highlighting the need for image-only solutions. Mobile-Agent(v2) and Moba show high "Misinterpretation of Task Context" rates, pointing to planning-level issues. In contrast, our approach has a low rate of this error, indicating that the basis subtask improves planning.

Improvement under Different Backbone Models. To test robustness across model capacities, we replace the backbone with a smaller model, GPT-4o-mini, and evaluate on medium-length CHOP-En instructions. As shown in Table 5, our method consistently outperforms others with both GPT-4o and GPT-4o-mini. Notably, our SR drops by only 0.2 with the smaller model, less than AppAgent and MobA. While Mobile Agent(v2) shows a similar drop, it still underperforms overall. These results confirm the robustness and generalizability of our subtask design.

Error Type	AppAgent	Mobile-Agent(v2)	Moba	Ours
Hallucinations	4.8%	5.9%	9.1%	0.0%
Poor Graphical Recognition	9.5%	29.4%	9.1%	54.6%
Misinterpretation of Task Context	23.8%	47.1%	63.6%	45.5%
Exceeds Max Iterations	4.8%	17.7%	4.6%	0.0%
XML/Model Output Parse Error	57.1%	0.0%	13.6%	0.0%

Table 4: Error distribution in mobile operating assistant.

Model	SR↑	CR↑	ME↑	AE↑	AAC↓
GPT-4o					
AppAgent	0.40	0.64	0.80	0.80	1.25
Mobile Agent (v2)	0.50	0.73	0.80	0.82	3.65
MobA	0.30	0.50	0.99	0.99	1.04
Ours	**0.70**	**0.89**	**1.20**	**0.94**	**0.85**
GPT-4o-mini					
AppAgent	0.20	0.56	0.72	0.72	1.35
Mobile Agent (v2)	0.40	0.65	0.75	0.75	3.83
MobA	0.20	0.45	0.98	0.98	1.10
Ours	**0.50**	**0.80**	**1.10**	**0.89**	**0.91**

Table 5: Performance comparison under two different backbone models.

Case Study. Finally, we demonstrate that our method enables agents to follow a more structured execution pattern, reducing errors and improving efficiency by generating multi-step actions in a single call. This leads to smoother task execution and faster completion times. A detailed explanation and figures can be found in Appendix F.

5 Conclusion

We present CHOP, a mobile operating assistant that enhances task execution by leveraging basis subtasks extracted from high-frequency human-executed sequences. CHOP identifies these basis subtasks through four key steps: verb extraction, synonym clustering, summarization, and frequency filtering. By integrating basis subtasks into the planning process, CHOP ensures that generated subtasks are both executable and aligned with key task pathways, leading to improved task effectiveness and efficiency. Experimental results on English and Chinese datasets demonstrate significant gains in execution quality over existing methods, highlighting CHOP as a robust solution.

Limitations

We believe the proposed CHOP method represents a significant step forward in advancing GUI agent research in the LLM era. However, several limitations remain that should be addressed in future work. First, the current evaluation process relies on manual assessments, which results in a relatively small dataset. Future research should aim to develop an automated evaluation pipeline to handle large-scale data and provide more stable and reproducible results. Second, our work currently focuses on the issues between the planning agent and the action agent in a multi-agent architecture, without exploring the potential challenges between the action agent and the grounding model. Future efforts should investigate how to better enable the action agent to effectively utilize the grounding model. Finally, the current architecture enhances VLM's planning capabilities in GUI scenarios through prompts, as searching for planning data is computationally expensive. However, fine-tuning directly on data offers a more reliable approach. Future research should explore the use of synthetic data for fine-tuning to further strengthen VLM's planning capabilities.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.

Chongyang Bai, Xiaoxue Zang, Ying Xu, Srinivas Sunkara, Abhinav Rastogi, Jindong Chen, et al. 2021. Uibert: Learning generic multimodal representations for ui understanding. *arXiv preprint arXiv:2107.13731*.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *NIPS*, pages 1877–1901.

Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A Plummer. 2022. A dataset for interactive vision-language navigation with unknown command feasibility. In *European Conference on Computer Vision*, pages 312–328. Springer.

Andrea Burns, Kate Saenko, and Bryan A Plummer. 2024. Tell me what's next: Textual foresight for generic ui representations. In *ACL (Findings)*.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2024. Chateval: Towards better llm-based evaluators through multi-agent debate. In *ICLR*.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. 2024. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *ICLR*.

Carlos G Correa, Mark K Ho, Frederick Callaway, Nathaniel D Daw, and Thomas L Griffiths. 2023. Humans decompose tasks by trading off utility and computational cost. *PLoS computational biology*, 19(6):e1011087.

Sunhao Dai, Chen Xu, Shicheng Xu, Liang Pang, Zhenhua Dong, and Jun Xu. 2024. Bias and unfairness in information retrieval systems: New challenges in the llm era. In *SIGKDD*, pages 6437–6447.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. 2024a. Metagpt: Meta programming for a multi-agent collaborative framework. In *ICLR*.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. 2024b. Cogagent: A visual language model for gui agents. In *CVPR*, pages 14281–14290.

Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, et al. 2024. Os agents: A survey on mllm-based agents for general computing devices use.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, et al. 2024. Autowebglm: Bootstrap and reinforce a large language model-based web navigating agent. *arXiv preprint arXiv:2404.03648*.

Sunjae Lee, Junyoung Choi, Jungjae Lee, Munim Hasan Wasi, Hojun Choi, Steve Ko, Sangeun Oh, and Insik Shin. 2024. Mobilegpt: Augmenting llm with human-like app memory for mobile task automation. In *MobiCom*, pages 1119–1133.

Gang Li and Yang Li. 2023. Spotlight: Mobile ui understanding using vision-language models with a focus. In *ICLR*.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: communicative agents for" mind" exploration of large language model society. In *NIPS*, pages 51991–52008.

Toby Jia-Jun Li, Amos Azaria, and Brad A Myers. 2017. Sugilite: creating multimodal smartphone automation by demonstration. In *CHI*, pages 6038–6049.

Toby Jia-Jun Li, Lindsay Popowski, Tom Mitchell, and Brad A Myers. 2021. Screen2vec: Semantic embedding of gui screens and gui components. In *CHI*, pages 1–15.

Toby Jia-Jun Li, Marissa Radensky, Justin Jia, Kirielle Singarajah, Tom M Mitchell, and Brad A Myers. 2019. Pumice: A multi-modal agent that learns concepts and conditionals from natural language and demonstrations. In *UIST*, pages 577–589.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. 2024. Showui: One vision-language-action model for generalist gui agent. In *NeurIPS 2024 Workshop on Open-World Agents*.

Zhe Liu, Cheng Li, Chunyang Chen, Junjie Wang, Boyu Wu, Yawen Wang, Jun Hu, and Qing Wang. 2024. Vision-driven automated mobile gui testing via multimodal large language model. *arXiv preprint arXiv:2407.03037*.

Dang Nguyen, Jian Chen, Yu Wang, Gang Wu, Namyong Park, Zhengmian Hu, Hanjia Lyu, Junda Wu, Ryan Aponte, Yu Xia, et al. 2024. Gui agents: A survey. *arXiv preprint arXiv:2412.13501*.

Runliang Niu, Jindong Li, Shiqi Wang, Yali Fu, Xiyu Hu, Xueyuan Leng, He Kong, Yi Chang, and Qi Wang. 2024. Screenagent: A vision language model-driven computer control agent. *arXiv preprint arXiv:2402.07945*.

OpenAI. 2021. Chatgpt. https://openai.com/research/chatgpt.

OpenAI. 2023. Gpt-4 technical report. Accessed on May 20, 2025.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318.

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 *UIST*, pages 1–22.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. From exploration to mastery: Enabling llms to master tools via self-driven interactions. In *The Thirteenth International Conference on Learning Representations*.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Towards completeness-oriented tool retrieval for large language models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 1930–1940.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2025. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy P Lillicrap. 2024. Androidinthewild: A large-scale dataset for android device control. In *NIPS*.

Liangtai Sun, Xingyu Chen, Lu Chen, Tianle Dai, Zichen Zhu, and Kai Yu. 2022. Meta-gui: Towards multi-modal conversational agents on mobile gui. *arXiv preprint arXiv:2205.11029*.

Bryan Wang, Gang Li, Xin Zhou, Zhourong Chen, Tovi Grossman, and Yang Li. 2021. Screen2words: Automatic mobile ui summarization with multimodal learning. In *UIST*, pages 498–510.

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. *arXiv preprint arXiv:2406.01014*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024b. A survey on large language model based autonomous agents. *FCS*, 18(6):186345.

Shuai Wang, Weiwen Liu, Jingxuan Chen, Weinan Gan, Xingshan Zeng, Shuai Yu, Xinlong Hao, Kun Shao, Yasheng Wang, and Ruiming Tang. 2024c. Gui agents with foundation models: A comprehensive survey. *arXiv preprint arXiv:2411.04890*.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. 2024. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024a. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Yuhao Yang, Yue Wang, Dongxu Li, Ziyang Luo, Bei Chen, Chao Huang, and Junnan Li. 2024b. Aria-ui: Visual grounding for gui instructions. *arXiv preprint arXiv:2412.16256*.

Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Qingwei Lin, Saravan Rajmohan, et al. 2024a. Large language model-brained gui agents: A survey. *arXiv preprint arXiv:2411.18279*.

Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, et al. 2024b. Ufo: A ui-focused agent for windows os interaction. *arXiv preprint 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. *arXiv preprint arXiv:2312.13771*.

Jiwen Zhang, Jihao Wu, Yihua Teng, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. 2024c. Android in the zoo: Chain-of-action-thought for gui agents. *arXiv preprint arXiv:2403.02713*.

Zhuosheng Zhang and Aston Zhang. 2023. You only look at screens: Multimodal chain-of-action agents. *arXiv preprint arXiv:2309.11436*.

Zichen Zhu, Liangtai Sun, Jingkai Yang, Yifan Peng, Weilin Zou, Ziyuan Li, Wutao Li, Lu Chen, Yingzi Ma, Danyang Zhang, et al. 2023. Cam-gui: A conversational assistant on mobile gui. In *MMSP*, pages 302–315. Springer.

Zichen Zhu, Hao Tang, Yansi Li, Kunyao Lan, Yixuan Jiang, Hao Zhou, Yixiao Wang, Situo Zhang, Liangtai Sun, Lu Chen, et al. 2024. Moba: A two-level agent system for efficient mobile task automation. *arXiv preprint arXiv:2410.13757*.

A Basis Subtask Details

Verb Extraction. To capture subtasks, we use the AITZ dataset (Zhang et al., 2024c), a subset of AITW (Rawles et al., 2024), covering four Apps. Each entry in the dataset contains an instruction and its step-by-step actions with the thought process. In AITW, raters annotate shorter sequences (at least $K \geq 3$ actions) as single-step demonstrations like "Add item to cart," which are considered subtasks. Since verbs can represent actions, we use *spaCy* for part-of-speech tagging, retaining only the verb to represent each instruction.

Synonym Clustering. Although verb extraction groups similar actions, synonyms with different expressions often serve the same function (e.g., "search news" vs. "lookup news".) Merging them reduces computational cost when generating subtasks (Wu et al., 2024). To cluster words by semantic similarity, we use *WordNet*[1] to group them into synonym sets (synsets.) Words are clustered based on shared synsets, reflecting their semantic similarity. After manual review, we retained verbs that represent meaningful actions and merged their corresponding action sequences.

Summarization. In GUIs, consistent logic is applied across software to enhance user experience. For example, "Search" in browsers and email Apps follows similar steps: "1. Click search box, 2. Enter content, 3. Click search button." Thus, action sequences within the same basis subtask should have similar representations. We standardize these sequences for the downstream action agent to improve performance. Specifically, for each basis subtask, we use GPT-4 to summarize its corresponding action sequences with the prompt: "Please summarize the following action sequence into a standardized process and specify boundary conditions."

Frequency Filtering. To reduce inference latency and mitigate performance degradation from long input sequences, we filter out low-frequency basis subtasks. As the more frequently used subtasks in the AITZ dataset are likely to appear on the critical path of task execution, we rank all candidate basis subtasks by their frequency and retain the top 10 most common ones. As shown in Table 6, these subtasks collectively account for 81% of user actions in the dataset. This selection ensures broad coverage while maintaining high efficiency and generalizability to unseen applications. Less frequent subtasks outside the top 10 are handled by GPT-4 through on-the-fly generation.

[1]https://github.com/argilla-io/spacy-wordnet

Table 6: Top 10 Most Frequent Basis Subtasks in AITZ

Rank	Basis Subtask	Frequency (%)
1	Interact	18.4
2	Search Item	15.2
3	Send Text Message	10.8
4	Open Section	8.3
5	View Content	6.7
6	Modify Settings	5.9
7	Check Notifications	4.5
8	Share Content	4.1
9	Manage Collections	3.7
10	Create or Edit Entry	3.4
Total (Top 10)		**81.0**

B Test Set Details

To conduct an in-depth comparison of the ability of our method and other assistants to handle complex user instructions and task execution efficiency on mobile devices, we evaluate them on two real-life scenario test datasets, namely, CHOP-En and CHOP-ZH.

The CHOP-En dataset consists of 30 instructions used to assess the performance of assistants in real-world mobile applications with a diverse set of English tasks. This dataset is collected following the setup of the dataset used in Mobile Agent(v2) (Wang et al., 2024a), where 10 widely used applications in China are selected, covering various everyday scenarios. For each application, three tasks of different levels of difficulty were included: easy, medium, and difficult. The easy-level instructions explicitly specify the app to be used and typically require fewer than five steps to complete. Medium-level instructions necessitate more actions to be executed, while difficult-level instructions are presented in natural language without specifying the app to be used.

The CHOP-ZH dataset consists of 200 human-curated and annotated Chinese instructions. The dataset is constructed by selecting 10 applications that cover a broad range of daily usage scenarios. For each application, annotators who are in-house data labelers first provide 20 instructions based on daily tasks and execute them on mobile phones. Before execution, annotators are asked to create a subtask plan for each task and describe their thought process before performing each action. Addition-

Dataset Name	CHOP-En	CHOP-ZH (Sampled)	CHOP-ZH (Full)
#Instructions	30	30	200
#Task Steps	5.57	5.53	
Language	English	Chinese	Chinese
Screen Image	×	✓	✓
Plan Thought	×	✓	✓
Action Thought	×	✓	✓

Table 7: Dataset details, including instruction count, task steps, and availability of supporting data.

ally, we anonymized all the data by replacing all personal information with placeholders. Compared to similar English task sets (Zhang et al., 2023; Wang et al., 2024a), the CHOP-ZH dataset is the first real-life **Chinese** test set designed for mobile devices. Additionally, while these datasets only provide instructions and corresponding actions for each step, the CHOP-ZH dataset offers a comprehensive task plan. This allows us not only to assess the overall performance of the architecture based on task execution but also to evaluate the plan agent's ability to decompose tasks, providing a more targeted evaluation. Due to the high cost of GPT-4o, we sample 3 instructions per app and assign them difficulty levels (easy, medium, hard) as in CHOP-En. The test instructions and CHOP-ZH details are in Table 7.

To enhance clarity, we provide additional details about the human annotation process. All annotations were conducted by in-house data labelers with undergraduate-level education, specifically recruited for this task. Prior to executing each instruction, annotators developed a subtask plan outlining the necessary steps, guided by predefined examples to ensure consistency. To maintain data quality, a dedicated review team manually inspected the annotations and removed incomplete instructions or those that could not be executed by the agent.

C Baseline Details

To provide a comprehensive evaluation, we also implement several baseline methods for comparison with our method to demonstrate its effectiveness and efficiency. These methods include the Human Baseline as well as some sophisticated agent-based automation approaches.

Human Baseline records the process of a human completing the instructions and is considered the golden solution for solving each task, as it reflects the best method based on human performance.

AppAgent (Zhang et al., 2023) introduces a framework with two phases: exploration and deployment. In the exploration phase, an agent learns app functions through self-learning or observation of humans, storing the knowledge in app-specific documents. During deployment, the agent uses these documents, along with the view hierarchy and screenshots, to plan and select actions. Each interactive element is labeled with bounding boxes and a unique index, improving the agent's accuracy in task execution.

Mobile Agent(v2) (Wang et al., 2024a) is a multi-agent system for mobile device operation assistance, comprising planning, decision, and reflection agents. The system takes screenshots as input and utilizes additional modules such as the OCR model and qwen-vl-plus API, enabling more effective action generation in complex mobile operation tasks.

Moba (Zhu et al., 2024) utilizes a two-level agent architecture with a Global Agent (GA) and a Local Agent (LA) to enhance mobile task automation. The GA interprets user commands and manages task planning, while the LA executes specific actions on the screen. The system takes as input both visual information and XML view hierarchy data to understand the mobile interface. For action execution, it employs a combination of OCR for text recognition and target localization to guide the selection of interactive elements.

D LLM-Based Evaluation Prompt

We use GPT-4o to compare two subtask plans for each instruction based on a structured prompt. The evaluation focuses on completeness, efficiency, and clarity. The full prompt is shown below:

Listing 1: Prompt used for LLM-based evaluation

```
Now I will give you an instruction and its
    corresponding two step-by-step solution
    strategies. I need your help to determine
    which strategy is better. There are three
    criteria for evaluation.
1. Whether it is complete and whether the
    instruction can be fully executed according
    to the step-by-step strategy.
2. Whether it is redundant, that is, whether
    there are useless steps in the generated
    strategy.
3. Whether it is clear, that is, whether the
    generated steps clearly express the intended
     purpose.

This is the instruction: {instruction}.
This is Plan One: {sub1}.
This is Plan Two: {sub2}.

The output format is:
### Reflection ###: Reflection on Two Solutions.
### Select ###: Option X.
```

Task	Subtasks
Search for videos about Stephen Curry on Bilibili and open 'Comments' to comment 'Oh, chef, your basketball spirit has always inspired me'	1. Find App (Bilibili) 2. Search Item (Stephen Curry videos) 3. Open Video (Stephen Curry) 4. Access Comments (Stephen Curry video) 5. Post Comment ('Oh, chef, your basketball spirit has always inspired me.')
Open the Calendar and look at today's date, then go to Notepad and create a new note to write 'Today is [today's date]'	1. Find App (Calendar) 2. Check Date (today's date) 3. Back Home 4. Find App (Notepad) 5. Create New Note (Today is [today's date])

Table 8: Two task examples with corresponding subtasks, with custom subtasks in red.

Task	Subtasks
Share the latest video from Bilibili content creator Johnny with Bob on WeChat.	1. Open the Bilibili app or website. 2. Find the latest video from content creator Johnny. 3. Click the share button and select WeChat. 4. In the sharing interface, choose the contact Bob and send the video link.
	1. Find App (Bilibili). 2. Search Item (Johnny). 3. Open Section (Johnny). 4. Share Content (WeChat, Bob).
Could you please check my search history on Baidu?	1. Open the Baidu browser or Baidu app. 2. Log in to your Baidu account (if not already logged in). 3. Access the history option and open it to view your Baidu search history.
	1. Find App (Baidu) 2. Open Section (search history) 3. Check Notifications (search history)

Table 9: Task examples with corresponding subtasks, without the basis subtask restriction. Ineffective subtasks are in blue, and inefficiency is in orange.

E Subtask Case

In Table 8, we present two examples, each containing a task and the corresponding subtasks decomposed by the plan agent in CHOP. As shown, our output not only includes basis subtasks but also features custom subtasks, highlighted in red. This demonstrates that our method can compensate for cases where the basis subtask cannot handle certain tasks by generating custom subtasks, thereby improving the quality of the generated subtasks.

In Table 6, we further present two examples showing that our basis subtasks can address both ineffectiveness and inefficiency issues. Specifically, in the first example, the task highlighted in blue is too complex to be executed by the downstream action agent. Our method breaks this blue subtask into two basis subtasks, making them simpler to execute, thus solving the ineffective subtask. Additionally, our method ensures more appropriate subtask granularity, such as using a single subtask for the sharing action, while without the restriction, two steps would be required. In the second example, the subtask highlighted in orange does not affect the task progression. Our method resolves this inefficiency by introducing a subtask in the critical path, thereby avoiding the inefficient subtask.

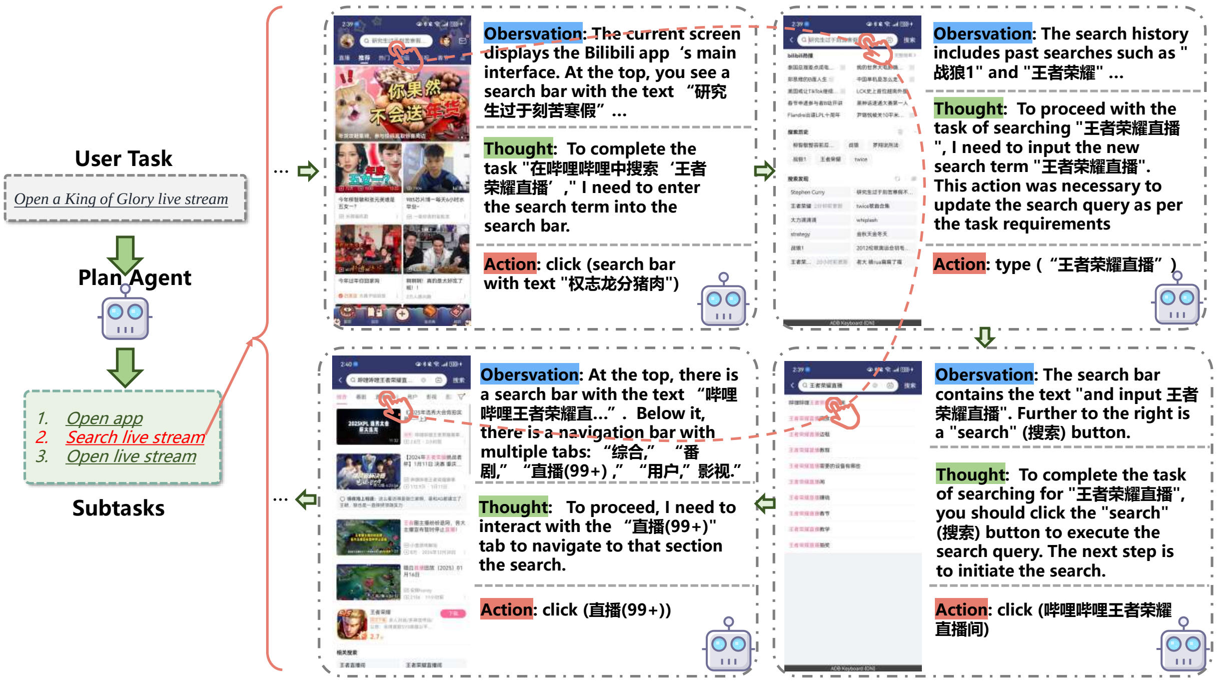
F Case Study

We present an example of the subtasks we executed in Figure 6. In this example, our method, due to the basis subtask, does not directly click "Live" on the

Basis Subtask	Explanation
Search Item (parameter)	Click on the search bar, type in the item name, and press enter. The parameter is the name of the item you want to search for. This action can be performed on any website with a search functionality. Output format is “Search Item (XXX)”.
Send Text Message (parameter)	This action involves typing a specific message into a designated text input area. The parameter is the content of the message to be sent. Output format is “Send Text Message (XXX)”.
Open Section (parameter)	Find and enter the specified section or feature in the application. The parameter is the name of the section, such as “Hot List”, “Messages”, “Settings”, etc.
View Content (parameter)	View the specified content in the application. The parameter describes the content to be viewed, such as “Latest News”, “Posts”, etc.
Interact (parameter1, parameter2)	Interact with the content in the application, such as “Like” or “Comment”. The parameter1 is the content to interact with, such as “Video” or “Song”. The parameter2 is the action of interaction, such as “Like content”, “Post a comment”, etc.
Manage Collections (parameter1, parameter2)	Manage personal collections or shopping carts, etc. The parameter1 includes actions such as “Add to Favorites”, “Delete”, and parameter2 includes items such as “Product”, “Video”, etc.
Share Content (parameter1, parameter2)	Share content from the application to other platforms or users. The parameter1 includes the sharing platform and parameter2 includes the recipient, such as “WeChat”, “Lucky”.
Check Notifications (parameter)	View notifications or messages in the application. The parameter is the section of the app, such as “System Notifications”, “Private Messages”, etc.
Modify Settings (parameter1, parameter2)	Modify the settings in the application. The parameter1 includes the setting item and parameter2 includes its changes, such as “Theme Skin”, “Notification Method”, etc.
Create or Edit Entry (parameter1, parameter2)	Create or edit entries in the application. The parameters include the entry type and name, such as “Playlist”, “Contact”, etc.

Table 10: Description of various basis subtasks and their explanations.

homepage to find relevant streams. Instead, it uses the “Search” basis subtask to perform the search. Although this approach may involve more steps than directly navigating to the live page, it is more structured and reliable, reducing the chances of execution errors. Additionally, since the “Search” process is relatively fixed, we can have the action agent generate the entire action sequence for the search subtask in one call, reducing the number of action agent invocations.



Obersvation: The current screen displays the Bilibili app ‘s main interface. At the top, you see a search bar with the text “研究生过于刻苦寒假” ...

Thought: To complete the task "在哔哩哔哩中搜索‘王者荣耀直播’," I need to enter the search term into the search bar.

Action: click (search bar with text "权志龙分猪肉")

Obersvation: The search history includes past searches such as "战狼1" and "王者荣耀" ...

Thought: To proceed with the task of searching "王者荣耀直播", I need to input the new search term "王者荣耀直播". This action was necessary to update the search query as per the task requirements

Action: type (“王者荣耀直播”)

Obersvation: The search bar contains the text "and input 王者荣耀直播". Further to the right is a "search" (搜索) button.

Thought: To complete the task of searching for "王者荣耀直播", you should click the "search" (搜索) button to execute the search query. The next step is to initiate the search.

Action: click (哔哩哔哩王者荣耀直播间)

Obersvation: At the top, there is a search bar with the text “哔哩哔哩王者荣耀直...”. Below it, there is a navigation bar with multiple tabs: “综合,” “番剧,” “直播(99+) ,” “用户,” 影视.”

Thought: To proceed, I need to interact with the “直播(99+)" tab to navigate to that section the search.

Action: click (直播(99+))

Figure 6: Subtask: Search live stream.