SeaPilot: Mobile Agent with Self-refining Environment Alignment

Zhigang Zuo[1], **Senyao Li**[1], **Yufeng Jiang**[1], **Haozhao Wang**[1], **Yichen Li**[1], **Wenchao Xu**[2], **Jingcai Guo**[3], **Ruixuan Li**[1*]

[1]School of Computer Science and Technology, Huazhong University of Science and Technology
[2]Hong Kong University of Science and Technology
[3]Hong Kong Polytechnic University

{zzg1018,hz_wang,rxli}@hust.edu.cn

Abstract

By combining cloud-side reasoning with edge-side observation and execution, cloud-edge collaboration has emerged as a promising paradigm for mobile UI agents. However, existing approaches face a fundamental trade-off between cost and accuracy. *On one hand*, step-by-step cloud interaction ensures high accuracy by uploading UI screens at every step and leveraging the cloud's powerful reasoning capabilities, but this inevitably incurs significant privacy costs for edge data and high computational expenses on the cloud. *On the other hand*, one-shot cloud planning drastically reduces the demand for edge privacy data and cloud compute by generating a complete plan based solely on the initial UI screen. Yet, this often leads to a substantial drop in accuracy, as the cloud agent may rely on invalid environment assumptions such as unseen app capabilities or UI flows. In this paper, we identify that the root cause of this trade-off lies in the *environment assumption gap*: the cloud agent lacks prior knowledge of the environment information required for edge-side task execution, and is thus forced to choose between planning with real-time information and planning based on speculative assumptions. To tackle this dilemma, we propose **SeaPilot**, which proactively provides the cloud agent with the necessary environmental context at the initial task submission stage, thereby achieving the dual benefits of both step-by-step and one-shot methods. To realize this, SeaPilot employs an iterative self-refining mechanism that progressively acquires feedback knowledge from execution failures across diverse tasks, enabling it to learn how to supply precise environmental information for each task in advance. Empirically, SeaPilot improves accuracy by **46.7%** over EcoAgent, a representative one-shot method, reduces cloud-token usage by $\mathbf{23.6}\times$ compared with step-by-step methods, and reduces privacy cost by **89.6%**, achieving a better cost–accuracy trade-off. The code is available at `https://github.com/zuoyerumeng/SeaPilot`.

1 Introduction

Mobile agents aim to automate smartphone tasks from natural-language instructions. Recent LLM-powered agents have shown strong capabilities in interpreting goals, observing mobile interfaces, and executing multi-step tasks [21, 18, 19, 26, 7, 14, 3, 1]. However, cloud-based agents incur latency, token cost, and privacy risks from frequent UI uploads [26, 14], while fully on-device agents remain

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

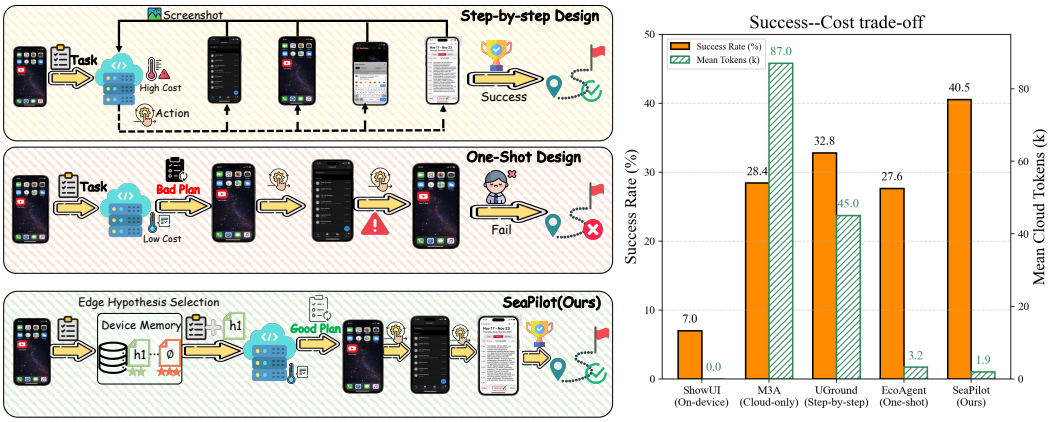


Figure 1: **Motivation.** Comparison of mobile agent paradigms and success and cost trade-offs. *Left*: step-by-step, one-shot, and SeaPilot-style cloud-edge designs; *Right*: success rate and mean cloud tokens across representative deployment settings.

limited in long-horizon reasoning and failure recovery [9, 10, 22]. This motivates cloud–edge mobile agents, which combine cloud-side reasoning with edge-side observation, execution, and verification.

As illustrated on the left side of Figure 1, existing cloud–edge mobile agents generally follow two paradigms. The *first* is step-by-step cloud interaction [5, 3], where the edge repeatedly sends observations to the cloud and receives the next action or subgoal. This design allows the cloud planner to stay aligned with the latest edge-side UI state and adapt its decisions during execution, greatly guaranteeing the accuracy. However, such adaptivity comes at the cost of frequent UI uploads, high cloud token consumption, increased latency, and greater privacy exposure. The *second* paradigm is one-shot cloud planning [25, 8], where the cloud generates a complete plan or a set of abstract milestones at the beginning, and the edge executes, verifies, and locally corrects them. While this design substantially reduces cloud interaction and better preserves privacy, it sacrifices real-time alignment between cloud-side planning and the actual edge environment and thus, inevitably suffers from accuracy degradation. Together with the cost–accuracy comparison on the right side of Figure 1, these two paradigms reveal a fundamental trade-off in cloud–edge mobile agents: step-by-step interaction provides strong environment adaptivity but incurs high cost, whereas one-shot planning reduces cost but weakens alignment with the edge environment.

Considering this, we in this paper aim to achieve both benefits of the step-by-step and one-shot methods. Specifically, we formalize that the trade-off arises from the *environment assumption gap* (EAG), i.e., the cloud agent lacking prior knowledge of the environment information required for edge-side task execution, and thus being forced to choose between planning with real-time information and planning based on speculative assumptions. Motivated by this, the key of the solution lies in equipping the cloud agent with sufficient environmental context to successfully accomplish the task. In essence, if a one-shot method were provided in advance with all the information typically gathered during step-by-step execution, it could formulate an accurate plan, thereby simultaneously ensuring low cost and high accuracy. Unfortunately, determining the requisite information prior to task execution is significantly challenging.

To address this challenge, we propose **SeaPilot**, a failure-driven framework for *Self-refining Environment Alignment* in cloud–edge mobile agents. The core idea of SeaPilot is to turn EAG-related failures into cross-task feedback for refining future cloud-side planning. Specifically, SeaPilot converts environment mismatches exposed by failure into reusable, device-local environment knowledge, represented as structured *environment hypotheses*. The induced hypothesis is stored locally on the edge device and updated through future execution outcomes. Before each subsequent cloud-planning request, the edge retrieves identical or semantically similar hypotheses and uses Thompson Sampling over their outcome statistics to select hypotheses that are both reliable and exploratory. The selected hypotheses are then provided to the cloud planner, allowing future plans to be aligned with the actual edge environment without repeatedly uploading intermediate UI states.

SeaPilot combines the advantages of one-shot and step-by-step agents by maintaining sparse cloud interaction while reusing knowledge from past failures to improve edge-environment alignment. This allows cloud-side planning to avoid repeated strategic errors and become increasingly environment-

aware over recurring tasks. Empirically, SeaPilot achieves a favorable cost–success–privacy trade-off: it uses $\mathbf{23.6}\times$ fewer cloud tokens than UGround, improves task success by **46.7%** over EcoAgent, and improves privacy mitigation by **89.6%** over CORE, with ablations validating its key design choices. Our contributions are summarized as follows:

- We characterize the cost–alignment trade-off in cloud–edge mobile agents and formalize the **Environment Assumption Gap**.
- We introduce **SeaPilot**, a failure-driven framework that converts execution failures into device-local environment hypotheses and reuses them to align future cloud-side planning with the actual edge environment.
- We demonstrate that SeaPilot improves task success while preserving the low cloud-cost advantage of one-shot planning.

2 Preliminaries

2.1 Mobile Task Automation

Mobile task automation aims to complete a user-specified smartphone task through interactions with mobile applications. Given a natural-language instruction x, a mobile agent observes the current UI state o_t and predicts an executable action a_t at each step:

$$a_t = \pi(x, o_{\leq t}, a_{<t}), \tag{1}$$

where $o_{\leq t}$ and $a_{<t}$ denote the observation and action histories, respectively. The action can be a mobile operation such as tapping, typing, scrolling, or navigating back. Executing a sequence of actions produces a trajectory τ and a task outcome $y \in \{0, 1\}$ indicating whether the task is successfully completed.

Depending on where the policy is deployed, mobile agents can be implemented as cloud-based, on-device, or cloud–edge systems. Since cloud-based agents incur high interaction costs and on-device agents are constrained by local model capacity, cloud–edge collaboration is becoming a practical paradigm for mobile agent deployment.

2.2 Cloud–Edge Mobile Agent Paradigms

Cloud–edge mobile agents combine cloud-side planning with edge-side observation, execution, and verification. Existing systems typically follow two paradigms [8]. In step-by-step cloud interaction [3, 5], the edge repeatedly queries the cloud during execution, allowing the cloud planner to adapt to the latest UI state but causing cloud-side cost to grow with task length. In contrast, one-shot cloud planning [25, 8] queries the cloud only at the beginning to generate a complete plan or a set of high-level milestones, while the edge is responsible for step-level execution and local correction.

We focus on the one-shot, low-interaction setting because it substantially reduces cloud communication but is more vulnerable to mismatch between cloud-side planning assumptions and the actual edge environment. Given the instruction x and limited cloud-visible context z_c, the cloud planner Ψ_c generates a sequence of high-level milestones:

$$G_0 = \{(g_1, s_1), \ldots, (g_K, s_K)\} = \Psi_c(x, z_c), \tag{2}$$

where g_k is the k-th abstract milestone and s_k denotes its expected state or completion condition. The edge executor then attempts to complete these milestones in the true device environment E_d:

$$(\tau, y) = \Pi_e(G_0, E_d), \tag{3}$$

where τ is the execution trajectory and y is the final task outcome.

3 Problem Formulation

Based on the one-shot low-interaction cloud–edge setting introduced above, we formalize the mismatch between cloud-side planning assumptions and the true edge-side environment. **Environment Assumption Gap.** In one-shot cloud planning, the cloud planner does not observe the device-specific

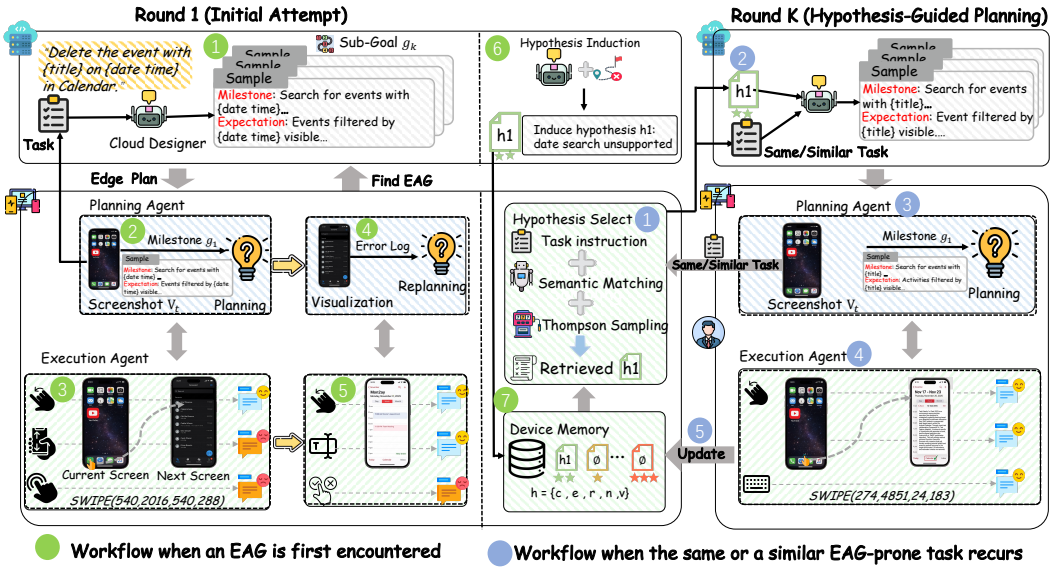


Figure 2: **Overview of SeaPilot.** The cloud may initially generate an infeasible milestone that is misaligned with the edge environment. Local replanning mitigates execution errors but cannot resolve milestone-level infeasibility. SeaPilot diagnoses such failures and converts them into reusable environment hypotheses for future cloud planning.

environment as it evolves during execution. As a result, it generates milestones according to an implicit environment model $\hat{E}_c$, which may differ from the true device environment E_d. The true environment E_d captures app capabilities, UI flows, app versions, installed applications, and user-specific configurations.

Let $\mathcal{F}(E)$ denote the set of milestones feasible in environment E. We say that a milestone (g_k, s_k) suffers from an *Environment Assumption Gap* if

$$(g_k, s_k) \in \mathcal{F}(\hat{E}_c) \quad \text{but} \quad (g_k, s_k) \notin \mathcal{F}(E_d). \tag{4}$$

That is, the cloud planner assumes that certain app capabilities, UI flows, or interaction paths are available, while the actual edge device does not support them. Unlike action-level execution errors, such failures indicate that the high-level milestone itself is strategically infeasible under the true device environment.

Objective. The goal is to improve task success over repeated tasks on the same edge environment under limited cloud interaction. For tasks $x_t \sim \mathcal{D}$, let $y_t \in \{0, 1\}$ denote whether task x_t is successfully completed. We formulate the objective as

$$\max \ \mathbb{E}_{x_t \sim \mathcal{D},\, E_d}[y_t] \quad \text{s.t.} \quad N_{\text{replan}}(t) \le R, \quad |G_h^t| \le M, \tag{5}$$

where $N_{\text{replan}}(t)$ denotes the number of cloud replanning operations for task t, and G_h^t denotes the hypothesis set selected from device-local memory. R and M are predefined deployment parameters that limit cloud replanning frequency and the number of selected hypotheses, respectively.

4 Method

We propose **SeaPilot**, a failure-driven framework that converts execution failures into reusable device-local environment hypotheses to improve future cloud-side planning.

4.1 Overview

SeaPilot represents reusable environment knowledge as structured environment hypotheses. Each hypothesis is represented as $h = (c, e, r)$, where c denotes its applicable scope, e denotes the inferred environment constraint, and r denotes the recommended planning adaptation. Unlike trajectory

memory or action replay [23, 4], an environment hypothesis abstracts a reusable mismatch between cloud-side planning assumptions and the true device environment.

Based on this representation, SeaPilot consists of three key components: **failure-driven hypothesis induction**, **device-local hypothesis memory**, and **uncertainty-aware hypothesis selection**. For each task instruction x_t, the edge device maintains a local hypothesis memory $\mathcal{M}_d$ associated with device d. Before cloud planning, the edge selects a compact hypothesis set $G_h^t \subseteq \mathcal{M}_d$ and provides it to the cloud planner as environment guidance. Following the one-shot cloud planning process in Eq. (2), SeaPilot augments the cloud planner input with the selected hypothesis set and generates milestones as $G_t = \Psi_c(x_t, z_c^t, G_h^t)$. The milestones are then executed locally by the edge-side executor through observation, grounding, execution, and self-correction.

During execution, SeaPilot updates selected hypotheses based on task outcomes. When failures are diagnosed as EAG-related, the cloud induces a new hypothesis for strategic replanning, which is then stored locally for future use.

4.2 Failure-Driven Hypothesis Induction and Replanning

When the edge executor fails to complete a milestone (g_k, s_k) within the local correction budget, SeaPilot does not directly perform ordinary trajectory-based replanning. Since action-only traces are often insufficient to distinguish local execution errors from invalid cloud-side assumptions, SeaPilot augments failed trajectories with compact textual UI descriptions.

$$H_{\text{fail}}^k = \langle x_t, (g_k, s_k), \{(a_i, \tilde{o}_i)\}_{i=1}^{n_k} \rangle \,, \tag{6}$$

where a_i denotes the i-th attempted action and $\tilde{o}_i$ denotes a compact textual description of the visible UI elements and screen state at that step, typically constrained to approximately 50–100 tokens.

Given this UI-grounded failure context, the cloud diagnoses whether the failure is related to an Environment Assumption Gap:

$$z_k = D_\Omega(H_{\text{fail}}^k), \quad z_k \in \{\text{EAG}, \text{Non-EAG}\}, \tag{7}$$

where D_Ω denotes the cloud-side diagnostic function. Equation (4) provides a conceptual definition of EAG rather than an explicitly computed feasibility test. In practice, D_Ω infers whether a failed milestone relies on an invalid environment assumption from the failed plan and observed execution evidence. If $z_k = \text{Non-EAG}$, SeaPilot treats the failure as a local execution issue, such as perception, grounding, or action-selection errors, and performs ordinary replanning without creating a new hypothesis. If $z_k = \text{EAG}$, the failure is attributed to an invalid cloud-side assumption about the target environment. SeaPilot then induces a structured environment hypothesis $\bar{h} = I_\Omega(H_{\text{fail}}^k)$, where I_Ω is the cloud-side hypothesis induction function and $\bar{h}$ follows the form $h = (c, e, r)$.

SeaPilot uses the induced hypothesis to guide strategic replanning from the current device state:

$$G_t' = \Psi_c(x_t, z_c^t, G_h^t \cup \{\bar{h}\}, \tilde{o}_{\text{cur}}), \tag{8}$$

where $\tilde{o}_{\text{cur}}$ denotes a compact textual description of the current screen. In this process, the failure trajectory supports diagnosis and hypothesis induction, while the current screen and induced constraint guide replanning, preventing the cloud planner from repeating the same invalid assumption.

4.3 Device-Local Hypothesis Memory and Update

Each edge device maintains a local memory of environment hypotheses, $\mathcal{M}_d = \{h_i\}_{i=1}^{N_d}$. SeaPilot stores this memory on the edge rather than in the cloud to preserve device-specific knowledge and enable direct outcome-based updates. The edge device directly observes execution outcomes, allowing it to update the usage and success statistics of selected hypotheses after each task. By maintaining hypotheses locally, SeaPilot sends only a compact selected set to the cloud planner, avoiding additional cloud-side search, computation, and token overhead.

Once SeaPilot diagnoses a failure as EAG-related, the cloud induces an actionable hypothesis $\bar{h} = (c, e, r)$ for hypothesis-guided replanning. The hypothesis is then stored in device-local memory as $h_i = (c_i, e_i, r_i, n_i, v_i)$, where (c_i, e_i, r_i) are its semantic fields, n_i is the number of times it is selected and used, and v_i is the number of successful task executions among those uses. A newly

stored hypothesis starts with $n_i = 0$ and $v_i = 0$, and its reliability is estimated from future execution outcomes.

When a hypothesis h_i is selected and used for cloud planning or replanning, SeaPilot updates its usage count by setting $n_i \leftarrow n_i + 1$. After the task terminates, SeaPilot updates the success count according to the final task outcome. Let $y_t \in \{0, 1\}$ denote whether task t is successfully completed. For each selected hypothesis $h_i \in G_h^t$, SeaPilot updates its success count by setting $v_i \leftarrow v_i + y_t$. By maintaining hypotheses locally and sending only a compact selected set to the cloud, SeaPilot preserves device-specific knowledge while keeping the cloud–edge interaction lightweight.

4.4 Uncertainty-Aware Hypothesis Selection

Before each cloud planning request, the edge device selects a compact hypothesis set from its local memory $\mathcal{M}_d$ as environment guidance. SeaPilot performs selection in two stages: semantic retrieval and uncertainty-aware sampling.

First, SeaPilot retrieves hypotheses whose applicable scopes are semantically related to the current task instruction. It reuses the edge-side textual agent as a lightweight retriever ϕ and computes the similarity between the task instruction x_t and each hypothesis scope c_i. The candidate set is defined as

$$\mathcal{R}_t = \mathrm{TopK}_{h_i \in \mathcal{M}_d} \, \mathrm{sim}_\phi(x_t, c_i). \tag{9}$$

This step filters out hypotheses that are unlikely to apply to the current task.

Second, SeaPilot selects hypotheses from $\mathcal{R}_t$ using an uncertainty-aware utility estimate. For each candidate h_i, it models the utility with a Beta posterior based on its usage count n_i and success count v_i, and selects the hypotheses with the highest sampled utilities:

$$G_h^t = \mathrm{TopM}_{h_i \in \mathcal{R}_t}(\theta_i), \quad \theta_i \sim \mathrm{Beta}(v_i + \alpha_0,\ n_i - v_i + \beta_0), \tag{10}$$

where α_0 and β_0 are prior parameters, and M is the number of hypotheses provided to the cloud planner. This Thompson-Sampling-based [17] strategy favors historically successful hypotheses while still exploring new or under-tested ones through posterior uncertainty. Only the finally selected hypotheses are included in the cloud planner's prompt, while unselected hypotheses remain in local memory without affecting the current planning process.

5 Experiments

5.1 Experimental Setup

Implementation details. We implement SeaPilot in the one-shot cloud–edge mobile-agent setting. The cloud planner Ψ_c uses GPT-4o, while the edge side includes a Qwen3-VL-2B planning agent and an execution agent based on Qwen2.5-3B for SeaPilot or Qwen3-4B for SeaPilot-Pro. All edge-side models are quantized and deployed on a local server with an NVIDIA RTX 4070 TiS GPU. Unless otherwise specified, we set the maximum number of cloud replanning rounds to $R = 1$, retrieve $K = 5$ hypotheses, and provide $M = 1$ hypothesis to the cloud planner. Sensitivity analyses are reported in the appendix.

Datasets. We conduct the main evaluation on AndroidWorld [14], which includes 116 tasks across 20 Android applications and provides programmatic validators for reproducible evaluation. All experiments are run on a Pixel 6 emulator with Android API level 33. To account for environment stochasticity, we conduct three independent experimental runs and report the average results. We also report supplementary results on AndroidLab [24]; since it relies on human verification rather than programmatic validators, large-scale automated evaluation is less practical.

Metrics. We evaluate each method from four aspects: effectiveness, cloud-side efficiency, privacy mitigation, and latency.

Task success rate. This metric measures the percentage of tasks successfully completed according to the benchmark validator.

Cloud-side efficiency. We measure cloud-side efficiency using the mean number of cloud calls (MC) and mean cloud tokens (MT) per task, which reflect cloud interaction frequency and token-level communication cost, respectively.

Table 1: **Main Performance on AndroidWorld.** We report task success rate (SR) and mean cloud tokens (MT). For compactness, Q2VL denotes Qwen2-VL, Q3VL denotes Qwen3-VL, Q2.5 denotes Qwen2.5, and Q3 denotes Qwen3.

Method	Architecture	Cloud Model	Edge Model	SR ↑	MT ↓
Pure device baselines					
OS-Atlas [22]	Single-Agent	–	OS-Atlas-Pro-4B	4.3%	0
ShowUI [9]	Single-Agent	–	ShowUI-2B	7.0%	0
InfiGUIAgent [10]	Single-Agent	–	InfiGUIAgent-2B	9.0%	0
Pure cloud baselines					
AppAgent [26]	Single-Agent	GPT-4o	–	11.2%	∼15k
M3A [14]	Multi-Agent	GPT-4o×2	–	28.4%	∼87k
Cloud–edge collaborative					
UGround [5]	Step-By-Step	GPT-4o×2	UGround-2B	32.8%	∼45k
CORE [3]	Step-By-Step	GPT-4o	Gemma-2-9B-IT	26.7%	∼11.3k
EcoAgent [25]	One-shot	GPT-4o	OS-Atlas-4B + Q2VL-2B	27.6%	∼3.2k
AdecPilot [8]	One-shot	GPT-4o	Q3-4B + Q3VL-2B	34.5%	∼1.9k
AdecPilot w/o Alignment	One-shot	GPT-4o	Q3-4B + Q3VL-2B	25.8%	∼2.3k
SeaPilot(Ours)	**One-shot**	**GPT-4o**	**Q3VL-2B + Q2.5-3B**	**36.2%**	**∼2.0k**
SeaPilot-Pro(Ours)	**One-shot**	**GPT-4o**	**Q3VL-2B + Q3-4B**	**40.5%**	**∼1.9k**

Privacy mitigation. Following prior work [3], we quantify privacy mitigation using the uploaded UI element reduction rate (RR). For a fair comparison, we only consider decision rounds where different methods operate on identical UI screens. Let $E_{\text{GPT-4o}}$ and E_{SeaPilot} denote the numbers of UI elements uploaded by the GPT-4o cloud-agent baseline and SeaPilot, respectively. The reduction rate is defined as:

$$RR = \frac{E_{\text{GPT-4o}} - E_{\text{SeaPilot}}}{E_{\text{GPT-4o}}}. \tag{11}$$

A higher RR indicates that fewer UI elements are exposed to the cloud, thereby reducing potential visual privacy leakage.

Latency. Latency reports the average end-to-end task completion time, capturing the practical runtime cost introduced by cloud interaction, local reasoning, and failure diagnosis.

5.2 Analysis of Cloud Tokens and Success Rate

SeaPilot further improves task success while preserving the low cloud-cost advantage of one-shot planning. Compared with UGround [5], SeaPilot-Pro uses about **23.6**× fewer cloud tokens and improves the success rate by **23.5%**. Compared with EcoAgent [25], SeaPilot-Pro achieves a **46.7%** relative improvement in success rate. Moreover, SeaPilot-Pro improves the success rate over AdecPilot w/o Alignment [8] by **56.9%** , without relying on manually injected environment information. These results show that SeaPilot effectively mitigates the *Environment Assumption Gap* in one-shot planning by automatically inducing and reusing edge-side environment hypotheses, achieving better task success under low cloud cost.

5.3 Effect of Accumulated Environment Hypotheses

For each independent run, SeaPilot accumulates environment hypotheses over five rounds of task execution. In each round, task instances are re-generated with newly sampled parameters, while the device-local hypothesis memory is continuously updated. The No Hypothesis setting differs from SeaPilot only in disabling the reuse of accumulated environment hypotheses. The SeaPilot results in Table 1 correspond to the five-round setting in Table 2. Table 2 evaluates the effect of accumulated environment hypotheses. Across both edge models, increasing hypothesis experience consistently improves task success while reducing execution steps and cloud-token consumption. For Qwen2.5-3B, SR increases from 24.14% without hypotheses to **36.21%** after 5 rounds, while Steps decrease from 9.89 to **9.58** and MT is reduced from 2187 to **2007**. For Qwen3-4B, SR improves from 28.45% to **40.52%**, with Steps reduced from 10.43 to **9.94** and MT reduced from 2111 to

Table 2: **Effect of accumulated environment hypotheses.** We compare an empty memory with hypothesis memories accumulated after different numbers of execution rounds. Results are reported as mean ± standard deviation.

Hypothesis Experience	Qwen2.5-3B			Qwen3-4B		
	SR ↑	Steps ↓	MT ↓	SR ↑	Steps ↓	MT ↓
No Hypothesis	24.14 ±1.29	9.89 ±0.44	2187 ±92	28.45 ±0.87	10.43 ±0.53	2111 ±89
1-Round Hypothesis	28.45 ±0.43	9.85 ±0.56	2131 ±94	33.62 ±0.86	10.16 ±0.47	2055 ±83
3-Round Hypothesis	31.03 ±0.87	9.71 ±0.39	2084 ±81	35.34 ±0.87	10.01 ±0.42	1982 ±97
5-Round Hypothesis	**36.21 ±0.86**	**9.58 ±0.47**	**2007 ±79**	**40.52 ±0.43**	**9.94 ±0.37**	**1934 ±85**

Table 3: Failure analysis over 60 AndroidWorld tasks. Failure types are not mutually exclusive.

Failure Type	Count	Description
Environment Assumption Gap	21	The cloud-side milestone relies on an invalid assumption about the actual device environment.
Grounding Failure	29	The edge executor fails to map the high-level instruction to the correct UI element.
Execution Failure	6	The correct UI target is identified, but the low-level action fails due to timing, precision, or interaction errors.
System Failure	11	The failure is caused by missing or incomplete UI-tree metadata from the accessibility interface.

1934. These results show that accumulated hypotheses provide effective cross-task environment feedback. By transforming past failures into reusable edge-side environment knowledge, SeaPilot progressively mitigates the Environment Assumption Gap, leading to higher one-shot planning success with lower execution and cloud-token costs. To further distinguish cross-task knowledge transfer from task-template adaptation, we conduct two stricter controlled evaluations, including same-template hypothesis exclusion and fully disjoint task templates, as detailed in Appendix E.

5.4 Analysis of Failure Types

We analyze failed executions over 60 tasks and categorize them into four non-mutually-exclusive types: Environment Assumption Gap (EAG), grounding failure, execution failure, and system failure. As shown in Table 3, grounding failure is the most frequent category, while EAG appears in 21 tasks, indicating that cloud-side plans often fail because they rely on invalid assumptions about the edge environment. These results support our motivation for SeaPilot: beyond low-level execution errors, low-interaction cloud planning can fail when the planner lacks accurate device-local context. While grounding, execution, and system failures are mostly tied to edge-side perception, action precision, or incomplete accessibility metadata, EAG failures can be mitigated by converting invalid assumptions into reusable device-local environment hypotheses.

5.5 Performance and Privacy Analysis.

Figure 3 demonstrates the superior equilibrium of SeaPilot-Pro among task generalization, execution effectiveness, and privacy preservation. First, Figure 3(a) captures the domain-level success rates on the AndroidWorld [14] benchmark. It reveals that SeaPilot-Pro outlines the largest performance area, dominating both routine and complex applications compared to existing collaborative baselines. Second, Figure 3(b) expands this multifaceted evaluation to the AndroidLab [24] environment. This tracking confirms robust generalization capabilities, indicating SeaPilot-Pro consistently outperforms advanced architectures including GPT-4o and CORE across diverse domains. Finally, Figure 3(c) reflects the privacy and utility evaluation. It illustrates that SeaPilot-Pro achieves the highest task success rate of 40.5% while yielding a 73.2% reduction rate for uploaded UI elements. In contrast to UGround [5], SeaPilot-Pro achieves a **23.5%** relative improvement in success rate and drastically boosts visual privacy protection from zero to 73.2%. Consequently, these multidimensional results

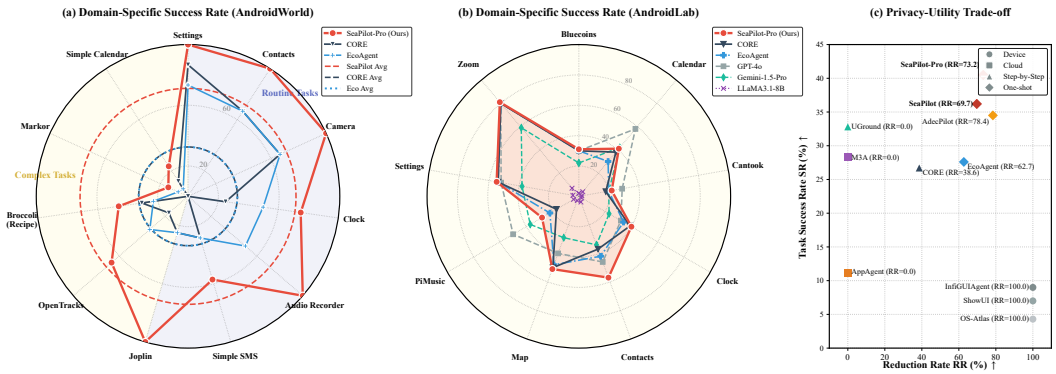


Figure 3: **Performance and Privacy.** (a) Completion rate within AndroidWorld domain breakdown. (b) Completion rate within AndroidLab domain breakdown. Baseline selection aligns with the benchmark's innate evaluation paradigm. (c) Privacy–utility trade-off on AndroidWorld.

Table 4: **Ablation on environment-hypothesis selection.** We compare the full SeaPilot selection strategy with variants that remove semantic filtering or uncertainty-aware sampling. Results are reported as mean ± standard deviation over three random seeds.

Selection Strategy	Qwen2.5-3B			Qwen3-4B		
	SR ↑	Steps ↓	MT ↓	SR ↑	Steps ↓	MT ↓
Full SeaPilot	**36.21** ±0.86	**9.58** ±0.47	**2007** ±79	**40.52** ±0.43	**9.94** ±0.37	**1934** ±85
w/o Semantic Filtering	24.57 ±1.72	10.02 ±0.81	2119 ±121	29.31 ±1.29	10.47 ±0.78	2386 ±114
w/o Thompson Sampling	24.14 ±1.28	10.11 ±0.74	2132 ±98	31.03 ±0.86	10.38 ±0.63	2335 ±101
Greedy Posterior Mean	35.34 ±0.43	9.59 ±0.42	2037 ±81	35.63 ±0.86	9.96 ±0.31	1984 ±76

validate that localizing reasoning through environment assumption decoupling effectively augments broad application adaptability and privacy without compromising system utility.

5.6 Ablation Study

Effect of Hypothesis Selection. Table 4 compares Full SeaPilot with three selection variants. **w/o Semantic Filtering** removes semantic filtering and selects from the whole hypothesis memory; **w/o Thompson Sampling** [17] directly uses the semantic Top-1 hypothesis; and **Greedy Posterior Mean** replaces Thompson Sampling with posterior-mean selection. The results show that Full SeaPilot achieves the best performance on both edge models. Removing semantic filtering leads to a clear drop in success rate, from **36.21%** to **24.57%** on Qwen2.5-3B and from **40.52%** to **29.31%** on Qwen3-4B, indicating that irrelevant hypotheses can mislead cloud planning. Removing Thompson Sampling also degrades performance, showing that semantic relevance alone is insufficient to estimate hypothesis reliability. Greedy Posterior Mean performs closer to Full SeaPilot but remains worse, suggesting that uncertainty-aware Thompson Sampling is more robust than deterministic exploitation.

6 Related Work

Mobile agents. LLM- and MLLM-based mobile agents have shown strong abilities in understanding instructions, interpreting mobile UIs, and executing multi-step tasks [18, 21, 19, 26, 7, 20, 2]. Existing methods can be broadly categorized by deployment setting into cloud-based, on-device, and cloud–edge agents [26, 14, 3]. Cloud agents provide strong reasoning but incur latency, token cost, and privacy risks from frequent UI uploads, while on-device agents reduce exposure but remain limited in long-horizon reasoning and failure recovery [9, 10, 22]. Cloud–edge agents seek a balance, but still face a trade-off between costly step-by-step interaction and less adaptive one-shot planning [5, 3, 25, 8]. Our work focuses on improving one-shot cloud–edge planning under this trade-off.

Agent memory and self-refinement. Recent agents use memory, reflection, and self-refinement to improve from past experience [15, 30, 13, 31, 6, 27]. They typically store trajectories, textual reflections, long-term memories, reusable skills, or procedural rules, and retrieve them to guide future planning [23, 4, 12, 28]. Other methods refine outputs, actions, or policies using self-feedback, execution results, or failed trials [11, 16, 29]. However, most prior work focuses on agent behavior refinement, while failures caused by mismatched environment assumptions remain less explored.

7 Limitations

SeaPilot primarily addresses failures caused by the Environment Assumption Gap (EAG), but does not directly resolve failures arising from edge-side visual grounding, low-level action execution, or incomplete accessibility metadata. Moreover, its effectiveness depends on the reliability of cloud-side failure diagnosis and hypothesis induction. Although execution feedback can reduce the influence of incorrect or outdated hypotheses, it does not guarantee their complete elimination. Evaluating SeaPilot under real-world environment evolution and improving long-term hypothesis memory management remain important directions for future work.

8 Conclusion

We introduced SeaPilot, a failure-driven framework for self-refining environment alignment in cloud–edge mobile agents. SeaPilot addresses the Environment Assumption Gap in one-shot cloud planning by diagnosing failed executions, inducing device-local environment hypotheses, and reusing them for future similar tasks. Experiments show that SeaPilot improves task success while substantially reducing cloud-token usage and privacy exposure, achieving a better success–cost–privacy trade-off.

Acknowledgments and Disclosure of Funding

This work was supported by the National Key Research and Development Program of China under Grant 2024YFC3307900; the National Natural Science Foundation of China under Grants 62376103, 62302184, 62436003, 62472332, and 62672205; the Major Science and Technology Project of Hubei Province under Grants 2025BAB011, 2024BAA008, and 2024BAB039; the Hubei Science and Technology Talent Service Project under Grant 2024DJC078; the Science and Technology Innovation Talent Program under grant 2023CB005; and Wuhan Dameng Database Co., Ltd.

Competing Interests. The authors declare no competing interests.

References

[1] Yuxiang Chai, Shunye Tang, Han Xiao, Weifeng Lin, Hanhao Li, Jiayu Zhang, Liang Liu, Pengxiang Zhao, Guangyi Liu, Guozhi Wang, et al. A3: Android agent arena for mobile gui agents with essential-state procedural evaluation. *arXiv preprint arXiv:2501.01149*, 2025.

[2] Gaole Dai, Shiqi Jiang, Ting Cao, Yuanchun Li, Yuqing Yang, Rui Tan, Mo Li, and Lili Qiu. Advancing mobile gui agents: A verifier-driven approach to practical deployment. *arXiv preprint arXiv:2503.15937*, 2025.

[3] Gucongcong Fan, Chaoyue Niu, Chengfei Lyu, Fan Wu, and Guihai Chen. Core: Reducing ui exposure in mobile agents via collaboration between cloud and local llms. *arXiv preprint arXiv:2510.15455*, 2025.

[4] Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. *arXiv preprint arXiv:2508.06433*, 2025.

[5] Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*, 2024.

[6] Wenjia Jiang, Yangyang Zhuang, Chenxi Song, Xu Yang, Joey Tianyi Zhou, and Chi Zhang. Appagentx: Evolving gui agents as proficient smartphone users. *arXiv preprint arXiv:2503.02268*, 2025.

[7] Sunjae Lee, Junyoung Choi, Jungjae Lee, Munim Hasan Wasi, Hojun Choi, Steve Ko, Sangeun Oh, and Insik Shin. Mobilegpt: Augmenting llm with human-like app memory for mobile task automation. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pages 1119–1133, 2024.

[8] Senyao Li, Zhigang Zuo, Haozhao Wang, Junyu Chen, Zhanbo Jin, and Ruixuan Li. Administrative decentralization in edge-cloud multi-agent for mobile automation. *arXiv preprint arXiv:2604.07767*, 2026.

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

[10] Yuhang Liu, Pengxiang Li, Zishu Wei, Congkai Xie, Xueyu Hu, Xinchen Xu, Shengyu Zhang, Xiaotian Han, Hongxia Yang, and Fei Wu. Infiguiagent: A multimodal generalist gui agent with native reasoning and reflection. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1035–1051, 2026.

[11] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

[12] Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025.

[13] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023.

[14] Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024.

[15] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

[16] Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. *Advances in neural information processing systems*, 36:58202–58245, 2023.

[17] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[18] Bryan Wang, Gang Li, and Yang Li. Enabling conversational interaction with mobile ui using large language models. In *Proceedings of the 2023 CHI conference on human factors in computing systems*, pages 1–17, 2023.

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

[20] Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. Mobile-agent-e: Self-evolving mobile assistant for complex tasks. *arXiv preprint arXiv:2501.11733*, 2025.

[21] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. Autodroid: Llm-powered task automation in android. In *Proceedings of the 30th annual international conference on Mobile computing and networking*, pages 543–557, 2024.

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

[23] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

[24] Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. Androidlab: Training and systematic benchmarking of android autonomous agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2144–2166, 2025.

[25] Biao Yi, Xavier Hu, Yurun Chen, Shengyu Zhang, Hongxia Yang, Fan Wu, and Fei Wu. Ecoagent: An efficient edge-cloud collaborative multi-agent framework for mobile automation. *arXiv e-prints*, pages arXiv–2505, 2025.

[26] Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–20, 2025.

[27] Guibin Zhang, Muxin Fu, and Shuicheng Yan. Memgen: Weaving generative latent memory for self-evolving agents. *arXiv preprint arXiv:2509.24704*, 2025.

[28] Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, et al. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026.

[29] Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li, Yueting Zhuang, and Weiming Lu. Agent-pro: Learning to evolve via policy-level reflection and optimization. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5348–5375, 2024.

[30] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642, 2024.

[31] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 19724–19731, 2024.

Appendix Contents

A Hyperparameter Sensitivity Analysis 14

B Algorithm Details 15

C Environment Hypothesis Examples 16

D Latency and cloud workload 16

E Cross-Task Transfer of Environment Hypotheses 17

F Reliability of EAG Diagnosis and Hypothesis Induction 18

G Prompt Templates 18

A Hyperparameter Sensitivity Analysis

We further analyze the sensitivity of SeaPilot to the two key hyperparameters in uncertainty-aware hypothesis selection: the semantic retrieval size K and the number of selected hypotheses M. The retrieval size K controls the number of semantically related hypotheses retained from the device-local memory, while M controls how many hypotheses are finally provided to the cloud planner after uncertainty-aware sampling. Unless otherwise specified, the default setting is $K = 5$ and $M = 1$.

A.1 Effect of Retrieval Size K

We first study the effect of the retrieval size K by fixing $M = 1$ and varying $K \in \{3, 5, 7\}$. The results are shown in Table 5. Across both backbone models, $K = 5$ achieves the highest task success rate, indicating that a moderate retrieval size provides the most reliable set of candidate hypotheses. A smaller retrieval size may miss useful hypotheses from the device-local memory, while a larger retrieval size may introduce weakly related or noisy candidates. Although the effects on token consumption, execution steps, and replanning frequency are not strictly monotonic across models, $K = 5$ provides the most consistent trade-off between retrieval recall and selection reliability.

Table 5: Sensitivity analysis of the semantic retrieval size K with $M = 1$.

K	Qwen2.5-3B				Qwen3-4B			
	SR ↑	MT ↓	Step ↓	Replan ↓	SR ↑	MT ↓	Step ↓	Replan ↓
3	26.72	2234	9.92	0.56	30.17	1994	10.48	0.50
5	36.21	2007	9.58	0.50	40.52	1934	9.94	0.43
7	25.00	2084	9.03	0.50	30.17	2232	10.36	0.50

A.2 Effect of Selected Hypothesis Number M

We then study the effect of the selected hypothesis number M by fixing $K = 5$ and varying $M \in \{1, 2, 3\}$. The results are shown in Table 6. Increasing M consistently reduces the task success rate for both backbone models, suggesting that providing more hypotheses does not necessarily improve planning performance. Although each environment hypothesis is short and the token overhead of the initial planning request is limited, additional hypotheses may introduce stale, redundant, or weakly relevant guidance, which can distract the cloud planner from the most reliable environmental cues. The effects on MT, execution steps, and replanning frequency are mixed and not strictly monotonic; in particular, a lower number of execution steps under larger M may reflect earlier task failures rather than higher execution efficiency. Overall, these results suggest that selecting a small number of reliable hypotheses is preferable to passing multiple potentially noisy hypotheses to the planner.

Table 6: Sensitivity analysis of the selected hypothesis number M with $K = 5$.

M	Qwen2.5-3B				Qwen3-4B			
	SR ↑	MT ↓	Step ↓	Replan ↓	SR ↑	MT ↓	Step ↓	Replan ↓
1	36.21	2007	9.58	0.50	40.52	1934	9.94	0.43
2	25.00	2037	8.86	0.49	29.31	2130	10.33	0.50
3	24.14	2049	8.81	0.49	29.31	2069	9.67	0.44

B Algorithm Details

Algorithm 1 SeaPilot for Cloud–Edge Mobile Task Automation

Require: Task instruction x_t, device-local hypothesis memory $\mathcal{M}_d$, cloud planner Ψ_c, edge executor Π_e, cloud diagnostic function D_Ω, hypothesis inducer I_Ω, retrieval size K, selected hypothesis number M

Ensure: Execution trajectory τ_t and task outcome y_t

Initialize the execution trajectory $\tau_t \leftarrow \emptyset$

Retrieve semantically related hypotheses from $\mathcal{M}_d$:

$$\mathcal{R}_t = \mathrm{TopK}_{h_i \in \mathcal{M}_d} \, \mathrm{sim}_\phi(x_t, c_i)$$

For each $h_i \in \mathcal{R}_t$, sample its utility:

$$\theta_i \sim \mathrm{Beta}(v_i + \alpha_0, \; n_i - v_i + \beta_0)$$

Select the guidance hypothesis set:

$$G_h^t = \mathrm{TopM}_{h_i \in \mathcal{R}_t}(\theta_i)$$

for each selected hypothesis $h_i \in G_h^t$ **do**
  Update usage count: $n_i \leftarrow n_i + 1$
end for

Generate cloud-side milestones:

$$G_t = \Psi_c(x_t, G_h^t)$$

while task is not terminated **do**
  Execute milestones on the edge device:

$$(\tau', y_t, H_{\text{fail}}^k) \leftarrow \Pi_e(G_t, E_d)$$

  Append trajectory: $\tau_t \leftarrow \tau_t \cup \tau'$
  if $y_t = 1$ **then**
    break
  end if
  if $D_\Omega(H_{\text{fail}}^k) = \mathrm{EAG}$ **then**
    Induce a structured environment hypothesis:

$$\bar{h} = I_\Omega(H_{\text{fail}}^k)$$

    Store the accepted hypothesis as $\bar{h} = (c, e, r, n, v)$ with $n \leftarrow 0$ and $v \leftarrow 0$
    Update local memory: $\mathcal{M}_d \leftarrow \mathcal{M}_d \cup \{\bar{h}\}$
    Replan with the updated environment guidance:

$$G_t \leftarrow \Psi_c(x_t, G_h^t \cup \{\bar{h}\})$$

  else
    Apply edge-side self-correction or request cloud replanning without inducing a new environment hypothesis
  end if
end while

for each selected hypothesis $h_i \in G_h^t$ **do**
  Update success count: $v_i \leftarrow v_i + y_t$
end for

return τ_t, y_t

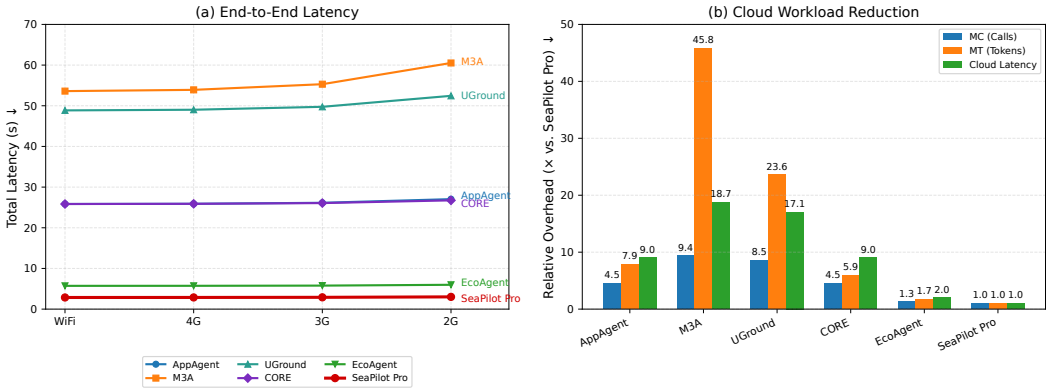


Figure 4: **Privacy and latency.** (a) End-to-end latency under different network bandwidths. (b) Normalized cloud-side workload relative to SeaPilot-Pro.

C Environment Hypothesis Examples

To make the learned environment hypotheses more concrete, we provide several representative examples in Table 7. Each hypothesis is represented as a triplet $h = (c, e, r)$, where c specifies the applicable scope, e describes the inferred environment constraint, and r suggests the corresponding planning adaptation. These hypotheses are device-local and are intended to capture reusable environment knowledge rather than task-specific action traces.

Table 7: Representative examples of environment hypotheses learned from failures.

Applicable Scope c	Environment Constraint e	Planning Adaptation r
Simple Calendar Pro Q&A tasks	The in-app search function indexes event titles only; queries by date or time fields return no results on this device.	Navigate directly to the target date in the calendar view; do not use the search entry for date- or time-based retrieval.
Markor note deletion tasks	A single tap on a file row opens the note editor rather than entering selection mode; delete actions only become available after a long-press.	Never plan a click-based deletion path.
OpenTrack activity search tasks	The in-app search function does not support time-based filters or temporal queries; searching for timestamps or dates returns no relevant activities.	Search using the exact activity name or task title; do not attempt to use dates or time ranges in the search input for retrieval.
Third-party app launch tasks	Typing the app name into the Google Search bar or launcher search does not reliably surface the installed app icon on this device state.	Swipe down to open the App Drawer and tap the icon.

D Latency and cloud workload

Appendix Figure 4 further explains the latency benefit of SeaPilot-Pro. SeaPilot-Pro consistently achieves the lowest end-to-end latency across all network conditions, increasing only from 2.86s under WiFi to 3.01s under 2G. The workload breakdown shows that this advantage mainly comes from reduced cloud usage. Compared with SeaPilot-Pro, M3A requires $9.4\times$ more cloud calls, $45.8\times$ more cloud tokens, and $18.7\times$ higher cloud-side latency, while UGround requires $8.5\times$ more cloud

Table 8: **Effect of excluding same-template hypotheses.** We report task success rate (SR) and the performance change in percentage points (pp).

Method	Original SR	w/o Same-Template	Δ (pp)
SeaPilot	36.2%	35.3%	-0.9
SeaPilot-Pro	40.5%	40.1%	-0.4

Table 9: **Cross-task generalization with disjoint task templates.** Task success rate (SR, %) on 40 held-out tasks using memories constructed from 76 separate tasks.

Method	No Memory	1 Round	3 Rounds	5 Rounds
SeaPilot	26.25	31.25	35.00	**41.25**
SeaPilot-Pro	32.50	38.75	41.25	**48.75**

calls and $23.6\times$ more cloud tokens. This confirms that SeaPilot-Pro reduces latency by lowering cloud workload rather than relying on favorable network conditions.

E Cross-Task Transfer of Environment Hypotheses

To further verify that SeaPilot learns reusable device-environment knowledge rather than adapting to specific evaluation task templates, we conduct two controlled experiments: excluding same-template hypotheses and evaluating on fully disjoint task templates.

E.1 Excluding Same-Template Hypotheses

We first examine whether SeaPilot relies on hypotheses generated from the same task template. For each evaluation task, we exclude all hypotheses previously generated from that template during retrieval, allowing only hypotheses from other task templates to guide cloud planning. The cloud planner, edge models, and all other experimental settings remain unchanged.

As shown in Table 8, excluding same-template hypotheses reduces the success rates of SeaPilot and SeaPilot-Pro by only 0.9 and 0.4 percentage points, respectively. These small changes indicate that the performance gains do not primarily depend on hypotheses originating from the same task template, supporting the reuse of environment knowledge across different tasks.

E.2 Fully Disjoint Task Templates

We further evaluate cross-task generalization under a stricter protocol with fully separated task templates. Specifically, we divide the 116 AndroidWorld task templates into 76 memory-construction tasks and 40 held-out evaluation tasks. The two subsets contain no overlapping templates, and we manually inspect the task definitions to avoid highly semantically similar tasks across the subsets.

Environment hypotheses are accumulated exclusively from the 76 memory-construction tasks. The 1-, 3-, and 5-round settings correspond to memories obtained after the respective numbers of complete execution rounds on this subset. We then freeze the device-local memory and evaluate on the 40 held-out tasks. Execution outcomes, failures, and newly induced hypotheses from the held-out tasks are not written back to memory.

As shown in Table 9, both configurations consistently benefit from accumulated environment knowledge. With five-round memory, SeaPilot improves from 26.25% to 41.25% (+15.00 pp), while SeaPilot-Pro increases from 32.50% to 48.75% (+16.25 pp).

Since the evaluation templates are excluded from memory construction and the memory remains frozen during testing, these improvements provide further evidence that SeaPilot acquires transferable device-environment knowledge rather than merely adapting to previously encountered evaluation templates.

Table 10: **Reliability of EAG diagnosis and hypothesis induction.** Results are reported as percentages.

Metric	Result
Diagnosis Accuracy	86.0%
EAG Precision	87.5%
EAG Recall	84.0%
EAG F1	85.7%
Non-EAG False-Positive Rate	12.0%
Hypothesis-Induction Accuracy	79.4%

F Reliability of EAG Diagnosis and Hypothesis Induction

We evaluate the reliability of cloud-side EAG diagnosis and hypothesis induction using 30 manually verified failure cases, including 15 EAG cases and 15 Non-EAG cases caused by edge-side perception, grounding, or execution errors. Using GPT-4o, we independently query each case five times, yielding 150 diagnostic outputs.

As shown in Table 10, GPT-4o achieves 86.0% diagnosis accuracy and an EAG F1 score of 85.7%. Conditioned on correct EAG diagnosis, hypothesis-induction accuracy reaches 79.4%. These results indicate that cloud-side diagnosis and hypothesis induction are generally reliable, although incorrect predictions may still occur.

G Prompt Templates

We provide the prompt templates used by SeaPilot in Table-style boxes below. These templates are used for cloud-side milestone planning, failure diagnosis with hypothesis induction, and replanning. The templates shown here are simplified for readability and can be instantiated with task-specific context, execution traces, and device-local hypotheses.

Cloud Milestone Planning Prompt

System: You are a High-Level Strategic Planner for Android automation.
Goal: {task_instruction}
Your job is to break this goal into a list of **semantic milestones**.
environment guidance:
{selected_hypotheses}
For each milestone, define:

- `"instruction"`: the high-level goal of this step (what to achieve, not how).
- `"expectation"`: the visual state of the screen after this step is completed.

Rules:

- Do not specify specific UI elements such as "click the blue button".
- Focus purely on the state transition.

Output strictly in JSON format:

```
[
  {"instruction": "...", "expectation": "..."},
  ...
]
```

Hypothesis Induction and replanning Prompt

System: You are a High-Level Strategic Planner for Android automation.
Goal: {goal}
Failed Plan: {old_plan}
Execution Trace: {trace}

Your job is to diagnose why the previous plan failed and produce a revised plan.
Step 1 — Diagnose the failure: Read the `[Monitor Observation]` and `[Action]` records. Classify the failure as one of the following:

- **Environment Assumption Gap (EAG)**: the planner had a wrong assumption about the app, UI behavior, widget interaction, or app constraints.
- **Non-EAG Execution Error**: the failure was caused by transient issues, UI lag, navigation mistakes, or the agent getting lost.

Step 2 — Branching logic:
Case A: Non-EAG failure
If the failure is execution-side, skip reflection and rule induction. Directly issue a recovery plan.
Output strictly in the following format:

```
Diagnosis: No EAG
Reason: <one-sentence description of the transient or execution-side error>
New Plan: <revised JSON milestone list to recover>
```

Case B: EAG confirmed
If the failure is caused by a wrong environment assumption, identify the wrong assumption and induce a reusable rule.
For the induced rule, define:

- `"Scope"`: the task or app types where this rule applies.
- `"Constraint"`: the actual UI behavior or app constraint observed.
- `"Adaptation"`: how future plans should adapt to avoid the same failure.

Output strictly in the following format:

```
Diagnosis: EAG
Wrong Assumption: <specific incorrect assumption>
Rule.Scope: <task/app scope>
Rule.Constraint: <actual observed environment behavior>
Rule.Adaptation: <future planning adjustment>
Plan: <new JSON milestone list applying the rule>
```

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Sections Introduction explicitly present the main contributions of this work.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses the limitations of the proposed method in Section 7. Specifically, our method is designed to mitigate failures caused by invalid cloud-side environment assumptions, but it is not intended to address errors originating from the edge-side system itself, such as perception failures, execution inaccuracies, or incomplete accessibility metadata.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The main contribution of this work does not lie in theoretical analysis, and the paper does not introduce new theoretical results that require formal assumptions or proofs.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The implementation details and hyperparameter settings are comprehensively described in Section Experiments and Appendix.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code used in this work are publicly available via the link `https://github.com/zuoyerumeng/SeaPilot`.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: All training and testing details are clearly specified in Section Experiments and Appendix.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The reported error bars are obtained from three independent experimental runs.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: Section Experiments specifies that we run all experiments on RTX4070tis GPU.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: All research complies with NeurIPS Ethics Guidelines. No human subjects or sensitive personal data involved. Benchmark datasets used contain publicly available non-identifiable information.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: At this stage, we do not identify any direct societal impacts, either positive or negative. The work is methodological in nature and does not involve applications that could raise immediate ethical, social, or policy concerns.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This work does not introduce or release any new datasets or pre-trained models with a high risk of misuse. The contributions are limited to methodological developments and experimental evaluations based on publicly available datasets. Therefore, no specific safeguards are required for responsible release.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All datasets and assets used in this work are publicly available and properly credited.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: No new datasets or models released beyond methodological implementations.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: No human subject research conducted. All data sources consist of publicly available benchmark corpora.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: No human subject research conducted. All data sources consist of publicly available benchmark corpora.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: We utilized AI assistants solely for grammatical error correction, sentence polishing, and improving the fluency of the manuscript. All scientific claims, experimental designs, code implementations, and logical arguments are the original work of the authors.