DEEP PERSONALIZATION: CONTINUOUS CONTEXT OBSERVATION FOR LOCAL PROACTIVE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

As AI agents become increasingly capable, users must still recognize when and how to ask for help, leaving many opportunities for useful support unrealized. Proactive agents can close this gap by recognizing when support may be useful and initiating interaction. However, effective proactivity requires understanding users' real-time activities and personalizing intervention timing so that assistance is offered when it is likely to be helpful rather than disruptive. We introduce **deep personalization**, a closed-loop process that connects continuous context observation with continual agent adaptation: an agent observes user activity, selectively retains long-term context, and adapts intervention decisions from implicit feedback. We initiate deep personalization in **Coco**, a proactive desktop agent. At Coco's core is an Observer Agent that monitors desktop activity, maintains compact long-term user modeling context, and learns when assistance is desirable from implicit signals such as engagement, dismissal, explicit requests, and repeated work patterns. We further develop efficient on-device strategies including change-aware frame selection and observation-first scheduling, enabling the Observer Agent to run on local AI compute platforms without streaming raw activity data off-device. To study deep personalization at scale without collecting sensitive real-user activity, we introduce a data-synthesis pipeline that transforms existing computer-use trajectories into **OSWorld-Persona**, a benchmark with persona-conditioned intervention labels. On OSWorld-Persona, deep personalization improves macro F1 across three model backbones from 0.12-0.19 under direct prompting to 0.35-0.48 after adaptation, a 2.4-3.0$\times$ relative gain. In a week-long user study, we observe that Coco improves its macro-averaged F1 score from 0.199 to 0.515. These results suggest that on-device deep personalization can make proactive agents more timely and increasingly aligned with each user's everyday work.

1 INTRODUCTION

Large language models (LLMs) and LLM-based agents are becoming increasingly capable, transforming how people work (Shao et al., 2024; Liu et al., 2025; Sager et al., 2026). Yet most agents today remain reactive: users must first recognize when AI could help, then initiate interaction through explicit instructions. This potentially limits the benefits of increasingly capable agents to those who already know when and how to engage them (Zhang et al., 2026a; Angrisani et al., 2026). Proactive agents offer a different paradigm, in which agents recognize opportunities for support and initiate interaction itself. This vision has a long history in intelligent interfaces (Maes, 1994; Horvitz et al., 1998) and is recently re-emerging with modern LLM-based agents (Lu et al., 2025; Yang et al., 2026; Hu et al., 2026; Shaikh et al., 2025b). However, proactivity requires an agent to decide not only what it can do, but whether its help is wanted at a particular moment. An intervention that is useful for one user may be redundant or disruptive for another, and even the same user may prefer different levels of assistance across tasks and situations (Horvitz, 1999).

We argue that addressing this problem requires moving beyond personalization based on static profiles or explicit configuration. What proactivity requires is **deep personalization**: a close-loop process in which an agent continuously observes user activities, selectively retains user context, makes personalized intervention decisions, and adapts itself from implicit user feedback. Such an agent builds its understanding of the user from everyday interaction—what they are working on, how they

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

work, what assistance they value, and when they prefer to be left alone—and keeps revising it as it observes new activity and sees how its own interventions land.

We present **Coco**, a personalized proactive agent that realizes deep personalization through three specific technical contributions. First, at its core is an Observer Agent that continuously processes desktop activities, maintains a compact representation of the user's ongoing work, and predicts whether proactive assistance would be valuable (§3.1). Second, a long-term memory module that accumulates context across observations, maintaining a hidden state that is updated with each new observation and used to condition downstream assistance (§3.2). Third, a closed-loop adaptation mechanism that treats user behavior such as acceptance, rejection, explicit requests, and repetitive-work patterns as implicit supervision signals to continually refine the Observer Agent via evolutionary prompt optimization and LoRA fine-tuning (§3.3). Together, these components enable Coco to learn each user's preferred intervention timing without manual annotation. As there is an inherent tension between personalization and data exposure, we develop an efficient deployment strategy combining change-aware frame selection with observation-first scheduling (§4) to keep deep personalization on device. We demonstrate that the Observer Agent can run entirely on a single consumer device, such as an RTX 5090 or DGX Spark, with latency suitable for processing user activity over multi-second windows.

To evaluate personalized proactive agents, we introduce **OSWorld-Persona**, a new benchmark for deep personalization that we construct via a novel data synthesis pipeline. Rather than collecting real user data which raises privacy concerns and limits reproducibility, we leverage existing computer-use benchmarks and use VLM-based user simulators conditioned on distinct personas to generate personalized labels for intervention timing appropriateness. Experiments on OSWorld-Persona show that personalized adaptation is essential for proactive support: direct prompting rarely identifies useful intervention opportunities (macro recall 0.08-0.19 and F1 below 0.19), while personalization via evolutionary prompt optimization or LoRA training raises macro F1 by 2.4-3.0$\times$ across base models and reaches up to 0.77 F1 on one of the personas. We further evaluate Coco on real user workloads in a week-long user study with 10 participant. Coco improves its macro-averaged F1 score from 0.199 to 0.515, demonstrating the effectiveness of deep personalization in the wild.

In summary, our contributions include:

- We introduce Coco, a personalized proactive agent built with deep personalization. Its Observer Agent continuously observes the user's digital context, learns when assistance is desirable from implicit feedback, and provides relevant context to the Execution Agent.
- We develop an efficient on-device deployment strategy that runs the Observer Agent locally for both time-sensitive screen understanding and personalization workloads, demonstrating that local compute can support privacy-preserving personalization in a large agentic system.
- We introduce a data curation pipeline for deriving synthetic personalization benchmarks from computer-use datasets and use it to curate OSWorld-Persona. We further instantiate Coco as a desktop app and validate its effectiveness in a week-long user study.[1]

2 Learning Personalized Agents from Continuous Observation

We study the problem of learning a personalized AI agent for an individual user. While various formulations and experimental setups have been proposed for personalized AI agents (Yang et al., 2026; Liu et al., 2026; Lyu et al., 2026; Liang et al., 2026), the core problem can be stated as follows: given an AI agent with minimal prior knowledge, enable it to learn appropriate behavior from the user and their interactions (Maes, 1994; Xu et al., 2026). Formally, let s_t denote the environment state at timestep t and h_t denote the agent's internal state. The agent's behavior is abstracted as a policy $\pi(a_t \mid s_t, h_t)$ that selects action a_t based on the environment state and internal state. Given feedback $r_t = \mathcal{R}(a_t)$ obtained from user u, personalization aims to find a learning algorithm $\mathcal{L}$ that yields the optimal user-conditioned policy from the interaction history:

$$\pi_u^* = \mathcal{L}(\tau^{(u)}) \propto \mathbb{E}_{\tau^{(u)}\sim\pi}\left[\Sigma_{t=1}^{T}\gamma^{t-1}r_t\right], \tag{1}$$

where $\tau^{(u)} = (s_1, h_1, a_1, ..., s_T, h_T, a_T)$ is the trajectory induced by π for user u, and $\gamma \in (0, 1]$ is a discount factor.

[1]The code and OSWorld-Persona data are available at `https://anonymous.4open.science/r/coco-iclr-annonymous-repo-7F7E`.

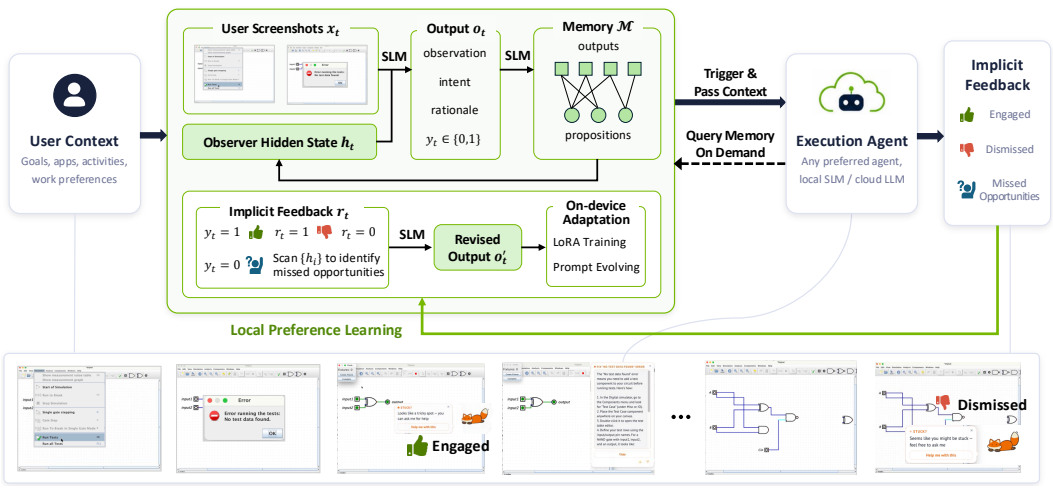


Figure 1: **Overview of Coco, a personalized proactive assistant.** Coco realizes deep personalization by closing the loop of continuous context observation, long-term user modeling and self-adaptation from implicit user feedback with an Observer Agent. Demo with Observer Agent deployed on-device: `https://anonymous.4open.science/r/coco-iclr-annonymous-repo-7F7E/coco-dgx-spark-demo.mov`

A central challenge in this formulation is obtaining informative s_t and r_t without repeatedly asking the user to explicitly specify their context or provide supervision. We consider **continuous context observation** as a natural source of both. For example, in digital environments, the agent can observe a representation of what the user themselves sees and does (*i.e.*, desktop screenshots and interaction history), and use this stream to infer ongoing activities and maintain h_t. The same interaction stream can provide feedback after the agent acts: editing an AI-generated draft, engaging with a proactive suggestion, dismissing it, or subsequently requesting assistance can all provide evidence about whether the agent's behavior matched the user's needs. Continuous observation therefore creates a ***closed-loop* deep personalization process** in which user activity provides context for an agent decision, and subsequent activity provides supervision for improving future decisions.

3 Method

We present **Coco**, a proactive assistant built on continuous context observation to personalize intervention timing. At its core, an Observer Agent transforms high-frequency desktop observations into compact structured representations and predict whether proactive assistance would be valuable (§3.1). Coco achieves deep personalization through long-term user modeling (§3.2) and continual observer adaptation from implicit feedback (§3.3). Figure 1 illustrates the system overview.

3.1 Observer Agent

Deep personalization requires the agentic system to maintain a coherent, evolving understanding of the user across time. Central to our personalization framework is the Observer Agent $\mathcal{O}$, which serves two tightly coupled roles: continuously distilling high-frequency desktop observations into compact structured representations of user activity, and deciding when proactive assistance would be valuable. Following the formulation in §2, let s_t denote the user-observable environment state, *i.e.*, desktop screenshots together with their timestamps, and h_{t-1} denote the personalized context accumulated before timestep t. At timestep t, the Observer Agent processes (s_t, h_{t-1}) and produces a structured observation o_t and an intervention decision y_t:

$$(y_t, h_t) = \mathcal{O}(s_t, h_{t-1}), \tag{2}$$

where $y_t \in \{0, 1\}$ denotes whether proactive assistance should be offered. When $y_t = 1$, Coco invokes the downstream Execution Agent; when $y_t = 0$, it continues passive observation.

We parameterize $\mathcal{O}$ as a multimodal language model capable of processing at least text and image inputs. Given a single prompt, $\mathcal{O}$ produces an output o_t consisting of four structured output fields:

(1) *observation*, a compact incremental update distilling valuable information from the user's recent computer-use activity; (2) *user intent*, an inference of the user's likely immediate goal; (3) *rationale*, a brief explanation of the proactive support decision grounded in the previous internal state and new screenshots; and (4) y_t, the binary support decision. Each o_t is written to long-term memory $\mathcal{M}$ and consolidated into higher-level information about the user (§3.2). Memory retrieval then constructs the context h_t used at the next timestep, closing the personalized context-update loop:

$$h_t = \text{Retrieve}(\mathcal{M}, o_t, t). \tag{3}$$

When $y_t = 1$, the Execution Agent receives h_t as initial personalized context and may retrieve additional information from $\mathcal{M}$ before generating assistance. The user's subsequent reaction provides implicit feedback r_t, which is used to continually refine $\mathcal{O}$ to personalize its behavior (§3.3).

3.2 Context Management and Long-Term User Modeling

Continuous observation provides Coco with rich evidence about a user's ongoing activities, but retaining the complete observation stream in the Observer Agent's context is neither efficient nor necessary. Coco therefore maintains a long-term memory $\mathcal{M}$ that organizes the Observer Agent's textual outputs and extracts higher-level patterns about the user.

Proposition-Observation Bipartite Graph We organize $\mathcal{M}$ as a bipartite graph with two sets of vertices: timestamped Observer outputs $\{o_1, \ldots, o_T\}$ and higher-level user propositions $\{p_1, \ldots, p_N\}$. We derive the latter using the General User Model (GUM) (Shaikh et al., 2025b). Using the same model that powers $\mathcal{O}$, GUM periodically synthesizes propositions from sequences of Observer Agent outputs to capture recurring patterns in the user's activities. Each proposition p_i represents an inferred property of the user, while edges connect p_i to the observations that provide evidence for it. When a new proposition is synthesized, Coco retrieves the top-k most similar existing propositions and determines whether the new evidence should create a new proposition or revise an existing one. When a proposition is revised, its supporting observation set is updated accordingly.

Context Management Memory serves two roles: maintaining compressed hidden state for the Observer Agent and providing long-term context to the Execution Agent. To retrieve content from $\mathcal{M}$, we focus on two logical primitives: temporal retrieval and semantic retrieval. After producing o_t, Coco retrieves the k most recent Observer outputs together with the k propositions most semantically relevant to o_t. Their combination forms the accumulated context h_t. $\mathcal{M}$ also bridges the Observer Agent and the Execution Agent invoked on demand. When $y_t = 1$, Coco provides h_t as initial context for proactive assistance and exposes $\mathcal{M}$ as a searchable tool that the Execution Agent can query. Rather than relying on a fixed retrieval policy or requiring the Observer Agent to determine all relevant context in advance, the Execution Agent can decide what information is needed and retrieve it from $\mathcal{M}$ through multi-turn tool calls. This separation resembles the interaction between a file system and an agent that selectively searches it in modern coding-agent systems.

3.3 Continual Adaptation via Self-Reflective Feedback

Long-term memory personalizes the context available to the Observer, but does not by itself change how $\mathcal{O}$ interprets that context or decides when to intervene. Coco therefore continually adapts the Observer Agent using implicit feedback derived from the user's responses to proactive suggestions and subsequent computer-use behavior.

Identifying Implicit Feedback For timesteps at which Coco intervenes ($y_t = 1$), we derive feedback r_t from the user's subsequent response. We assign $r_t = 1$ when the user engages with the proactive assistance and $r_t = 0$ when the user rejects it, providing direct evidence about whether the intervention was appropriate. However, relying solely on these signals risks a degenerate policy in which $\mathcal{O}$ learns to intervene less frequently and receives fewer signals. To recover such missed opportunities, we employ the same on-device model in $\mathcal{O}$ to retrospectively scan $\{h_1, \ldots, h_T\}$ and identify cases in which the user subsequently exhibits evidence of needing assistance based on heuristics, such as repeatedly performing the same operation, becoming stuck, or explicitly invoking AI (see full prompts in Appendix B.3). When such evidence follows a timestep with $y_t = 0$, we treat the preceding decision as a missed intervention opportunity and assign negative feedback to that decision. We acknowledge that this secondary signal identifies potential support opportunities rather than confirmed user preferences; nonetheless, it provides the exploration signal and subsequent user feedback under similar circumstances can further calibrate it.

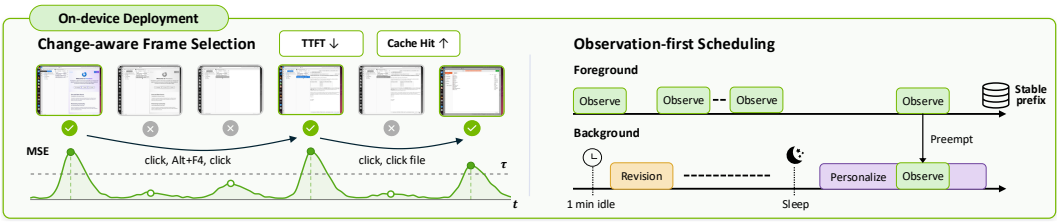


Figure 2: **Efficient on-device deployment of the Observer Agent.** Coco uses change-aware frame deduplication to reduce time to first token (TTFT), together with workload-aware scheduling that prioritizes observation requests and performs personalization updates during idle periods.

Observer Adaptation Once we infer r_t, we recover the corresponding Observer Agent input (s_t, h_{t-1}) and output o_t, and use the on-device model for self-reflection. When $r_t = 0$, the model produces a revised output o'_t; when $r_t = 1$, we set $o'_t = o_t$. The resulting dataset $\mathcal{D} = \{(s_t, h_{t-1}, o'_t)\}$ in a certain period of time is used to update $\mathcal{O}$. While Coco is agnostic to the particular learning algorithm applied to $\mathcal{D}$, we instantiate two low-resource adaptation strategies: (1) **LoRA Training (Coco-LoRA)**: examples in $\mathcal{D}$ are periodically shuffled and used to update the LoRA adapter of the multimodal language model that powers $\mathcal{O}$ (Hu et al., 2022). (2) **Prompt Evolution (Coco-PE):** following ACE (Zhang et al., 2026b), we maintain a compact bullet-point instruction memory that is iteratively refined through generation, reflection, and curation. At each batch, the current memory is used to generate predictions over $\mathcal{D}$. Incorrectly predicted examples are selected and augmented with temporally adjacent correctly predicted examples to provide contrastive evidence. These examples are then passed through reflection and curation to identify useful behavioral rules and update the Observer Agent prompt. Appendix B.4 provides the details.

4 Efficient On-device Deployment

To preserve user privacy, Coco's Observer Agent preferably runs on-device, ensuring raw screen content never leaves the user's machine. Unlike conventional LLM serving in data centers, the Observer Agent faces a distinct workload: it must process high-frequency screen captures in real time and concurrently handle personalization updates. As illustrated in Figure 2, we address these challenges through frame deduplication to reduce redundant token computation and priority scheduling.

Change-aware Frame Selection Consecutive screen captures are often nearly identical, particularly during typing and scrolling. However, even small visual differences can lead to different multimodal tokens due to the way multimodal language models encode images. Given Observer Agent input s_t, rather than feeding all screenshots to the multimodal model, we first align the user mouse and keyboard actions with each screenshot. Then we retain the initial and final frames but keeps an intermediate frame only when its RGB mean-squared error from the most recently retained frame exceeds a threshold τ. Actions associated with discarded frames are forwarded to the next retained frame, preserving interaction semantics while avoiding redundant visual tokens. On a workload replayed on DGX Spark, this strategy raises the KV cache hit rate from 26.7% to 45.2% by removing redundant visual tokens that would otherwise evict reusable cache entries. This is especially important for on-device deployment under memory constraints.

Observation-first Scheduling Coco treats continuous observation and user-facing inference as foreground workloads, while personalization runs opportunistically in the background. Lightweight revision jobs become eligible after one minute of user inactivity, whereas full personalization is deferred until a longer idle period or an explicit sleep mode where the user turns off the continuous observation. Personalization executes as a bounded, checkpointed, low-priority process and is preempted whenever interactive inference requires the model. Besides preventing personalization from delaying observation, separating these heterogeneous workloads improves cache locality: consecutive Observer requests share a stable prompt prefix, allowing their KV-cache entries to remain resident instead of being frequently displaced by personalization prompts.

In real-world deployment, we combine these strategies with speculative decoding through block diffusion prediction (Chen et al., 2026) to further speed up inference. Appendix D provides the on-device deployment details and workload profiling statistics.

5 EXPERIMENT

5.1 OSWORLD-PERSONA: REPURPOSING COMPUTER-USE DATA FOR PROACTIVE AGENT

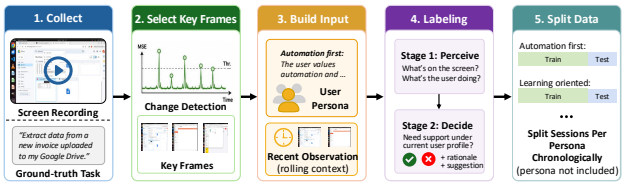


Figure 3: **Synthetic data pipeline for personalized proactive agent studies.** We use this pipeline to construct OSWorld-Persona from OSWorld v2 trajectories.

Existing datasets for studying proactive agents are largely confined to text-only environments and do not capture the complexity of real-world workflows. To bridge this gap, we propose a general data synthesis pipeline that can leverage any computer-use dataset. The pipeline consists of five stages: First, we collect screen recordings from a computer-use dataset of choice; second, from these recordings, we follow §4 to extract key frames that capture meaningful state transitions; third, we build input instances by pairing key frames in a rolling context with different user personas that describes the user's preferences and behavior patterns; fourth, the combined inputs are feed into a two-stage labeling process, where a VLM leverages the persona description to generate personalized assistance labels; finally, the labeled sessions are split chronologically into train and test sets per user.

We instantiate this pipeline on OSWorld 2.0 (Yuan et al., 2026), as it particularly curates long-horizon tasks that closely resemble everyday computer activities in both personal and professional scenarios. We consider 4 different personas ranging from wanting more support from AI ("automation first") to only preferring proactive support when stuck ("learning oriented"), and employ Kimi-K2.6 to simulate the user's decision based on the given persona. For quality control, we apply a self-consistency check at temperature 0.4, retaining an example only when two additional runs reproduce the original `need_support` label, and apply human validation by letting human to review cases where `need_support` is true and annotating context that informs potential proactive support. This yields **OSWorld-Persona**, comprising 6,558 key frames and 3,259 proactive support annotations from 108 sessions. Appendix C includes data synthesis prompts and detailed statistics.

5.2 EVALUATION SETUP

Models We consider the following multimodal models that can be deployed locally on consumer hardware or AI PCs to cover different model architectures: (1) Qwen-3.5-9B, dense VLM with 9B activation parameters, (2) Qwen-3.5-35B-A3B, mixture-of-experts VLM with 3B activation parameters, (3) Nemotron-3-Nano-Omni, mixture-of-experts omni models with 3B activation parameters.

Baselines We compare Coco observer adaptation method with (1) Vanilla, that prompts the off-the-shelf model to make the binary support decision; (2) Induced Persona, that induces compact user profile based on the training data (Zhang, 2024); (3) Oracle Persona, that includes the oracle persona used in data synthesis (not available in actual deployment). We further compare Coco context management with (1) relying on the model context and (2) Mem0 (Chhikara et al., 2025) that uses LLM to extract salient information from observations and retrieve similar memories based on vector embeddings. For both Coco and Mem0, we use Qwen-3.5-9B to construct the memory, and use the answer model to decide the retrieval queries. We limit the number of retrieval call to 5.

Datasets and Metrics We use **OSWorld-Persona** constructed in §5.1 to evaluate the proactive support quality. Following Lu et al. (2025), we report: (1) **Recall**, the fraction of genuine needs the agent identifies; (2) **Precision**, the fraction of offered interventions the user accepts; (3) **Accuracy**, overall correctness of the agent's predictions; (4) **False-Alarm**, the fraction of interventions offered when none was needed; and (5) **F1-Score** provides a balanced measure of the effective of the agent's proactive behavior. To evaluate our context management strategy, we adopt standard memory benchmarks including **LoCoMo** (Maharana et al., 2024), **LongMemEval** (Wu et al., 2025), **BEAM-100K**, and **BEAM-1M** (Tavakoli et al., 2026). These benchmarks evaluate context management quality on a large number of conversation histories and use **LLM-as-a-judge Accuracy**. We use `gpt-5.5` as the judge model and follow the same judging protocol in Chhikara et al. (2025)[2].

[2] `https://github.com/mem0ai/memory-benchmarks`

Model	Method	Macro Average					F1-Score Per Persona ↑			
		Precision ↑	Recall ↑	Acc. ↑	FA ↓	F1 ↑	Automation First	Eager Hands-on	Learning-oriented	Reserved Draft
Qwen-3.5-9B	Vanilla	0.4231	0.1739	0.6498	**0.0468**	0.1861	0.0371	0.1387	0.3919	0.1765
	Induced Persona	0.3729	0.2221	0.6380	0.1055	0.2094	0.0785	0.1993	0.2583	0.3014
	Oracle Persona	0.4758	0.4345	0.7295	0.1599	0.3932	0.2096	0.7291	0.3099	**0.3243**
	Coco-PE	0.4845	**0.4933**	**0.7759**	0.2492	**0.4832**	**0.5233**	0.8088	0.3888	0.2121
	Coco-LoRA	**0.5052**	0.4334	0.7753	0.2343	0.4551	0.3126	**0.8408**	**0.4264**	0.2404
Qwen-3.5-35B-A3B	Vanilla	0.3884	0.1863	0.6350	**0.0857**	0.1827	0.1020	0.1507	0.1900	**0.2878**
	Induced Persona	0.4373	0.2886	0.6490	0.0864	0.2408	0.1184	0.2158	**0.3440**	0.2850
	Oracle Persona	0.4448	0.3192	0.6527	0.1188	0.2713	0.2671	0.3810	0.1769	0.2602
	Coco-PE	**0.5678**	**0.3785**	**0.7846**	0.1298	**0.4334**	**0.4514**	**0.7369**	0.2603	0.2850
	Coco-LoRA	0.4087	0.3335	0.6784	0.1697	0.3135	0.1911	0.5385	0.2563	0.2682
Nemotron-3-Nano-Omni	Vanilla	0.4050	0.0815	0.6485	**0.0359**	0.1175	0.0362	0.0951	0.1525	0.1863
	Induced Persona	0.4260	0.1965	0.6494	0.0714	0.2192	0.0657	0.1991	**0.3043**	**0.3075**
	Oracle Persona	0.4222	0.2218	0.6927	0.1285	0.2810	0.1721	0.5247	0.1515	0.2759
	Coco-PE	0.4784	0.3025	0.6922	0.1233	0.3211	**0.3229**	0.4737	0.1928	0.2949
	Coco-LoRA	**0.4923**	**0.3152**	**0.7857**	0.2237	**0.3485**	0.1771	**0.8609**	0.1635	0.1923

Table 1: **Results on OSWorld-Persona.** Compared to baselines, Coco-PE and Coco-LoRA achieve higher F1 scores across models and personas, but exhibit higher False-Alarm (FA) rates, as vanilla models rarely trigger proactive support. Results are averaged over three random seeds.

6 Results and Analysis

6.1 Main Results

Observer Adaptation Effectiveness As shown in Table 1, Vanilla observers rarely identify support opportunities, yielding macro recall of only 0.082-0.186 and F1 scores below 0.19. Providing persona information alone improves performance, but remains insufficient: Induced Persona reaches F1 scores of 0.209-0.241, while Oracle Persona reaches 0.271-0.393. In contrast, Coco's adaptation strategies raise the best macro F1 to 0.483 on Qwen-3.5-9B (2.6× over vanilla), 0.433 on Qwen-3.5-35B-A3B (2.4×), and 0.349 on Nemotron-3-Nano-Omni (3.0×), even outperforming the corresponding Oracle Persona baseline. Crucially, these gains are not merely a byproduct of intervening more often as both precision and accuracy improve. The false alarm rate of the adapted observer is slightly higher than the vanilla observer as the adapted observer shift from near-silence toward active intervention. Coco-PE is the stronger method on the Qwen models, whereas Coco-LoRA is better on Nemotron-3-Nano-Omni.

Context Management Effectiveness We ablated context management component in the Observation Agent by comparing against two restricted variants: (1) **Coco (No Context)**, which removes the accumulated activity context (h_t) from the Observer Agent input entirely, and (2) **Coco (Decision Only)**, which lets the Observer Agent to output the intervention decision alone rather than the structured output including the observation and user intent. Results in Table 2 show that the richer signal provided by context management is important for personalization adaptation. Concretely, under Coco-PE, removing the accumulated context reduces F1 from 0.483 to 0.388 on Qwen-3.5-9B, from 0.433 to 0.398 on Qwen-3.5-35B-A3B, and from 0.3211 to 0.3040 on Nemotron-3-Nano-Omni-30B-A3B. Under Coco-LoRA, removing context also substantially reduces F1, particularly on Qwen-3.5-9B, where F1 drops from 0.455 to 0.285. These results indicate that accumulated context is important for personalized proactive assistance and is not merely a prompting artifact. At the same time, by comparing against the results of Coco (Decision Only), we find letting the Observer Agent output the observation not only allows it to accumulate context but also helps the prediction of the proactive decision.

Method	Variant	Qwen-3.5-9B	Qwen-3.5-35B-A3B	Nemotron-3-Nano-Omni
Coco-PE	Full	**0.4832**	**0.4334**	**0.3211**
	No Context	0.3878	0.3983	0.3040
	Decision Only	0.3045	0.1993	0.2529
Coco-LoRA	Full	**0.4551**	**0.3135**	**0.3485**
	No Context	0.2851	0.2951	0.3247
	Decision Only	0.4286	0.3204	0.2450

Table 2: **F1 scores of different Observer Agent formulations.** The full Observer Agent achieves the best F1 under both Coco-PE and Coco-LoRA.

Answer Model	Memory	LoCoMo	LongMemEval	BEAM-100K	BEAM-1M
GPT-5.5	/	0.9364	**0.9440**	0.5811	0.5121
	Mem0	0.7734	0.7220	0.6750	0.6257
	Coco	**0.9513**	0.8220	**0.7500**	**0.6786**
Claude-Sonnet-4.6	/	0.9162	**0.9040**	0.5394	0.3840
	Mem0	0.7240	0.7180	0.7275	0.7171
	Coco	**0.9448**	0.7520	**0.8375**	**0.8329**
Kimi-K3	/	0.9227	**0.9200**	0.6010	0.3951
	Mem0	0.7955	0.7260	0.5966	0.5799
	Coco	**0.9565**	0.8340	**0.7194**	**0.7253**

Table 3: **Memory benchmark results.** For Mem0 and Coco, Qwen-3.5-9B constructs the memory.

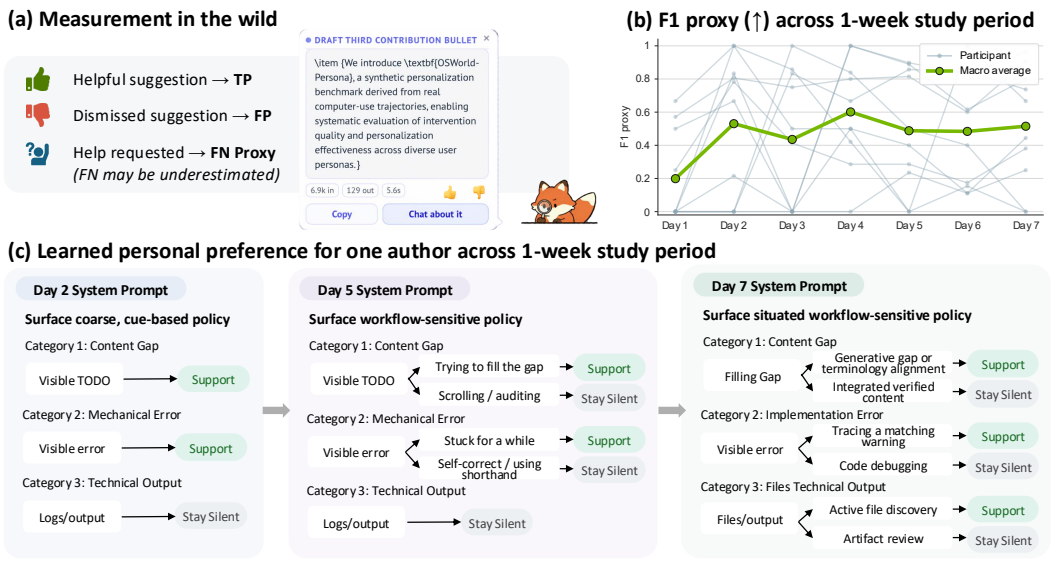


Figure 4: **Deep personalization in the wild.** In a week-long study with 10 participants who used Coco during their everyday computer activities, Coco's macro F1 proxy rose from 0.199 to 0.515.

We further compare the Coco's context management strategy with other alternative approaches on standard memory benchmarks. As shown in Table 3, across three different answer models, Coco consistently outperforms Mem0 on every benchmark and achieves the best results on LoCoMo, BEAM-100K, and BEAM-1M. The advantage is most pronounced on the longer BEAM settings: for example, with Claude Sonnet 4.6, Coco improves accuracy from 0.539 to 0.838 on BEAM-100K and from 0.384 to 0.833 on BEAM-1M relative to using no external memory. Coco similarly raises BEAM-1M performance from 0.512 to 0.679 with GPT-5.5 and from 0.395 to 0.725 with Kimi-K3. The vanilla long-context models perform best on LongMemEval as conversations in this benchmark fit within the context windows; but it does work well when the memory goes longer.

6.2 User Study

To assess whether Coco's on-device deep personalization yields useful and well-timed assistance in real-world settings, we instantiate Coco as a desktop assistant and conduct a user study in which 10 participants used Coco during their everyday computer activities for one week. The Observer Agent runs Qwen3.5-9B on a single RTX 5090 GPU, and is adapted using Coco-PE, the best-performing method for this backbone according to our benchmark results. During the study, we explain Coco's proactive functionality to participants and provide minimal instructions, allowing them to use their computers as normal. The only instructions are to use the thumb-up/thumb-down buttons and to request help when desired through Coco's text or audio interface. Appendix F provides further details on Coco desktop assistant implementation.

The Observer Agent's inputs, outputs, and curated memory remain on the user's device. With informed consent, we collect only three types of interaction events as illustrated in Figure 4 (a): (1) a thumb-up on a proactive suggestion, treated as a true positive; (2) a thumb-down or dismissing the suggestion without revealing the full content, treated as a false positive; (3) a direct text or audio request when Coco remains silent, treated as a proxy for a false negative[3]. Figure 4 (b) reports the F1 proxy over the one-week study period. While fluctuations exist for individual users on certain days, the macro-averaged F1 proxy improves steadily over time, rising from 0.199 to 0.515 during the study period. The results also reflect a cold-start effect: on Day 1, 6 out of 10 users have an F1 proxy of zero because Coco produces no helpful proactive suggestions. After adaptation, all six begin receiving suggestions they rate as helpful.

Because participants' observation data remain private, we qualitatively examine an author's week-long deployment separately, reviewing the local adaptation data $\mathcal{D} = \{(s_t, h_{t-1}, o'_t)\}$ and the resulting evolved system prompt. Figure 4 (c) illustrate three categories of learned decision boundaries on

[3]False negatives may be underestimated, as users may desire assistance without explicitly prompting Coco.

different days. Interestingly, the learned boundaries in the system prompt shows a trend of progressive refinement. By Day 2, the deep personalization procedure learns coarse cues, such as a visible "TODO" often signals an opportunity to help, whereas technical output often calls for silence. Subsequent feedback allows Coco to learn more nuanced understanding of the user's typical workflows. For example, on Day 5, Coco learns to assist when the user is actively filling a "TODO", but to remain silent when the user is scrolling through or auditing the full document. By Day 7, the prompt further distinguishes between filling a generative gap, where assistance is welcome, and integrating verified content, where the user prefers doing it themselves. These refinements demonstrate how deep personalization learns situated preferences beyond the static profiles or explicit settings used by conventional personalization approaches.

7 Related Work

Proactive Computing Proactive computing enables systems to anticipate needs and act before explicit requests, which pushes the limit of computing and elicits new needs that can lead to increased human productivity and quality of life (Tennenhouse, 2000). It has been applied to information retrieval (Rhodes & Maes, 2000), recommendation (Resnick & Varian, 1997; Sarwar et al., 2001; Covington et al., 2016), programming assistance (Pu et al., 2025; Liu et al., 2024), and ubiquitous computing (Weiser, 1991). In modern LLM systems, proactive agents remain nascent. Lu et al. (2025) formulate the proactive agent problem and define evaluation metrics, and subsequent work studies proactivity in mobile agent and conversational settings (Yang et al., 2026; Hu et al., 2026). Our work focuses on personalization as proactive assistance is valuable only when it surfaces the right thing at the right moment, a judgment that is inherently subjective (Horvitz, 1999).

Data Source for Personalization Personalization commonly draws on user messages, writing, and other digital traces (Zhang et al., 2025b; Salemi et al., 2024), although these signals can be noisy. More explicit approaches solicit pairwise preferences (Li et al., 2024), demonstrations (Shaikh et al., 2025a), or direct edits to prompts and personas (Müller et al., 2026). A second family infers signal from natural interaction instead of soliciting it. For example, Sharma et al. (2022) adapt a robot planning policy from human corrections, and Gao et al. (2024) recover latent preferences from user edits to LLM agent output. These methods scale better than direct elicitation but often tied to a specific task. A third option infers preferences from user activity. In Shaikh et al. (2025b), general preference propositions are induced from raw observations of computer use, but these propositions are open-ended and are not confirmed by the user. Building on the last two directions, we propose a deep personalization framework that learns user preferences through continuous context observation.

Personalized Agent Adaptation In agentic systems, personalization affects the full decision pipeline rather than only response generation (Zhang et al., 2025b; Xu et al., 2026). Unlike adaptation with verifiable objectives and ground-truth evaluators (Yuksekgonul et al., 2026; Novikov et al., 2025), personalized adaptation must learn latent, evolving preferences from a single user's feedback, often while keeping sensitive data on device. One line of work applies parameter-efficient fine-tuning to user data (Tan et al., 2024; Wang et al., 2026), or specializes adapters for individual components of the agentic system such as memory (Bini et al., 2025). With the growing instruction-following capabilities of LLMs, recent work also induces personalized patterns into retrievable skills (Xia et al., 2026). While prior work in agentic settings largely rely on large cloud-based models, we show that on-device-scale models can also provides a knob for agent personalization.

8 Conclusion

We present Coco, a personalized proactive agent that realizes deep personalization by closing the loop of continuous context observation, long-term user modeling and self-adaptation from implicit user feedback with an Observer Agent. By efficiently deploying the Observer Agent locally, we further demonstrate that privacy-preserving personalization is not only principled but practical on a single consumer device. To support rigorous evaluation of personalized proactive agents, we introduce OSWorld-Persona, a new synthetic benchmark constructed via data synthesis pipeline that enables evaluation of deep personalization without releasing real user data. Experiments on OSWorld-Persona and a week-long user study show that personalization is essential for proactive support, and that Coco is able to adaptive to individual needs in the wild.

AI USE STATEMENT

In this work, LLMs are used to assist with code development and also used as components of the data synthesis pipeline for support-decision labeling. We manually check all model-generated content. LLMs are not used for research ideation, result interpretation, experimental design, or generating claims about system performance. We also use Coco, the local proactive assistant introduced in this work, to support daily research work and section drafting during the writing process. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This paper proposes deep personalization, a closed-loop process connecting continuous context observation with continual agent adaptation. We instantiate this framework as Coco, a proactive agent that improves suggestion timing and reducing disruptive interruptions. We see personalized proactive agents holding particular promise for democratizing AI assistance: many people do not yet recognize when AI could help them (Savoldi et al., 2025; Zhang et al., 2025a), and a system that observes context and intervenes at the right moment can bridge the gap.

While our experiments demonstrate its effectiveness, they also surface an inherent tension fundamental to personalized AI research more broadly: the more user context and feedback an agent can observe and retain, the better it personalizes; yet broader observation increases exposure to sensitive information and risks revealing personal habits and preferences to malicious parties. Our deep personalization approach directly addresses this tension. By separating the Observer Agent from the broader system, we confine continuous desktop observation and adaptation to a self-contained, on-device component. We further contribute efficient on-device deployment strategies and demonstrate that real users can be served using local compute, including a single consumer RTX 5090 GPU or DGX Spark. Although the deep personalization algorithm is not tied to any particular system instantiation, we view on-device deployment as more than an engineering optimization, but as a principled privacy safeguard. By keeping observation, context retention, and adaptation local, the system reduces the need to route sensitive user data through third-party providers. We hope this encourages future work on personalized agents to prioritize privacy-preserving, on-device adaptation, enabling agents to better align with individual needs and preferences while minimizing unnecessary data exposure.

We nonetheless acknowledge several residual risks. First, continual adaptation may reinforce undesirable behavioral patterns or overfit to transient user behavior. Practical deployments should therefore provide user-accessible controls to inspect, edit, correct, or reset learned context. Second, the same observational capability that enables personalization could be misused as a surveillance mechanism if deployed without appropriate safeguards. We strongly caution that any deployment of such systems must require explicit, informed, and revocable user consent, with clear disclosure of what is observed, what is retained, and how the learned context and preference is used.

This paper does not release any user-activity dataset. Instead, we introduce a data synthesis pipeline that leverages existing computer-use datasets to study personalized proactive agents without exposing real user activity traces. Our one-week user study was conducted with full informed consent from all participants. During the study, we only collected aggregate interaction statistics, including the number of helpful proactive suggestions, unhelpful proactive suggestions, and user-requested assistance events. The Observer Agent was deployed locally using a self-hosted Qwen3.5-9B model on an RTX 5090 GPU, so participant desktop observations were processed on-device rather than sent to third-party model providers.

REFERENCES

Marco Angrisani, Maria Casanova, Nathanael J Fast, Jimmy Narang, and Juliana Schroeder. Gaps in large language model awareness, usage, and perceptions in the united states: Evidence from a nationally representative longitudinal survey. PNAS nexus, 5(2):pgag007, 2026.

Massimo Bini, Ondrej Bohdal, Umberto Michieli, Zeynep Akata, Mete Ozay, and Taha Ceritli. Memlora: Distilling expert adapters for on-device memory systems. arXiv preprint arXiv:2512.04763, 2025.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In Forty-third International Conference on Machine Learning, 2026. URL https://openreview.net/forum?id=Oz335dV48X.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. arXiv preprint arXiv:2504.19413, 2025.

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In Proceedings of the 10th ACM conference on recommender systems, pp. 191–198, 2016.

Ge Gao, Alexey Taymanov, Eduardo Salinas, Paul Mineiro, and Dipendra Misra. Aligning llm agents by learning latent preference from user edits. Advances in neural information processing systems, 37:136873–136896, 2024.

Eric Horvitz. Principles of mixed-initiative user interfaces. In Proceedings of the SIGCHI conference on Human Factors in Computing Systems, pp. 159–166, 1999.

Eric Horvitz, Jack Breese, David Heckerman, David Hovel, and Koos Rommelse. The lumière project: Bayesian user modeling for inferring the goals and needs of software users. In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, UAI'98, pp. 256–265, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 155860555X.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In International Conference on Learning Representations, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Haoyi Hu, Qirong Lyu, Xianghan Kong, Weiwen Liu, Jianghao Lin, Zixuan Guo, Yan Xu, Yasheng Wang, Weinan Zhang, and Yong Yu. Anticipate and learn: Unleashing idle-time compute in proactive agents. arXiv preprint arXiv:2605.25971, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In Proceedings of the 29th symposium on operating systems principles, pp. 611–626, 2023.

Xinyu Li, Ruiyang Zhou, Zachary C Lipton, and Liu Leqi. Personalized language modeling from personalized human feedback. arXiv preprint arXiv:2402.05133, 2024.

Kaiqu Liang, Julia Kruk, Shengyi Qian, Xianjun Yang, Shengjie Bi, Yuanshun Yao, Shaoliang Nie, Mingyang Zhang, Lijuan Liu, Jaime Fernández Fisac, et al. Learning personalized agents from human feedback. arXiv preprint arXiv:2602.16173, 2026.

Ailin Liu, Yesmine Karoui, Fiona Draxler, Frauke Kreuter, and Francesco Chiossi. Sensing what surveys miss: understanding and personalizing proactive llm support by user modeling. In Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems, pp. 1–21, 2026.

Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. Large language model-based agents for software engineering: A survey, 2025. URL https://arxiv.org/abs/2409.02977.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. In International Conference on Learning Representations, volume 2024, pp. 47832–47850, 2024.

Yaxi Lu, Shenzhi Yang, Cheng Qian, Guirong Chen, Qinyu Luo, Yesai Wu, Huadong Wang, Xin Cong, Zhong Zhang, Yankai Lin, et al. Proactive agent: Shifting llm agents from reactive responses to active assistance. In International Conference on Learning Representations, volume 2025, pp. 47431–47457, 2025.

Yibo Lyu, Gongwei Chen, Rui Shao, Weili Guan, and Liqiang Nie. Personalalign: Hierarchical implicit intent alignment for personalized gui agent with long-term user-centric records. In Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 36074–36089, 2026.

Pattie Maes. Agents that reduce work and information overload. Communications of the ACM, 37 (7):30–40, 1994.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 13851–13870, 2024.

Elias Müller, Kirsten Greiner, Adrian Wegener, Florian Onur Kuhlmeier, and Alexander Maedche. Personalizing human-llm interactions through mixed profiling. In Proceedings of the Extended Abstracts of the 2026 CHI Conference on Human Factors in Computing Systems, pp. 1–6, 2026.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. arXiv preprint arXiv:2506.13131, 2025.

Kevin Pu, Daniel Lazaro, Ian Arawjo, Haijun Xia, Ziang Xiao, Tovi Grossman, and Yan Chen. Assistance or disruption? exploring and evaluating the design and trade-offs of proactive ai programming support. In Proceedings of the 2025 CHI conference on human factors in computing systems, pp. 1–21, 2025.

Paul Resnick and Hal R Varian. Recommender systems. Communications of the ACM, 40(3):56–58, 1997.

Bradley J Rhodes and Pattie Maes. Just-in-time information retrieval agents. IBM Systems journal, 39(3.4):685–704, 2000.

Pascal J Sager, Benjamin Meyer, Peng Yan, Rebekka von Wartburg-Kottler, Layan Etaiwi, Aref Enayati, Gabriel Nobel, Ahmed Abdulkadir, Benjamin F Grewe, and Thilo Stadelmann. A comprehensive survey of agents for computer use: Foundations, challenges, and future directions. Journal of Artificial Intelligence Research, 85, 2026.

Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. Lamp: When large language models meet personalization. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 7370–7392, 2024.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th international conference on World Wide Web, pp. 285–295, 2001.

Beatrice Savoldi, Giuseppe Attanasio, Olga Gorodetskaya, Marta Marchiori Manerba, Elisa Bassignana, Silvia Casola, Matteo Negri, Tommaso Caselli, Luisa Bentivogli, Alan Ramponi, et al. Generative ai practices, literacy, and divides: An empirical analysis in the italian context. arXiv preprint arXiv:2512.03671, 2025.

Omar Shaikh, Michelle Lam, Joey Hejna, Yijia Shao, Hyundong Cho, Michael Bernstein, and Diyi Yang. Aligning language models with demonstrated feedback. In International Conference on Learning Representations, volume 2025, pp. 20498–20525, 2025a.

Omar Shaikh, Shardul Sapkota, Shan Rizvi, Eric Horvitz, Joon Sung Park, Diyi Yang, and Michael S Bernstein. Creating general user models from computer use. In Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology, pp. 1–23, 2025b.

Yijia Shao, Yucheng Jiang, Theodore Kanell, Peter Xu, Omar Khattab, and Monica Lam. Assisting in writing wikipedia-like articles from scratch with large language models. In Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pp. 6252–6278, 2024.

Pratyusha Sharma, Balakumar Sundaralingam, Valts Blukis, Chris Paxton, Tucker Hermans, Antonio Torralba, Jacob Andreas, and Dieter Fox. Correcting robot plans with natural language feedback. arXiv preprint arXiv:2204.05186, 2022.

Zhaoxuan Tan, Qingkai Zeng, Yijun Tian, Zheyuan Liu, Bing Yin, and Meng Jiang. Democratizing large language models via personalized parameter-efficient fine-tuning. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pp. 6476–6491, 2024.

Mohammad Tavakoli, Alireza Salemi, Carrie Ye, Mohamed Abdalla, Hamed Zamani, and J Ross Mitchell. Beyond a million tokens: Benchmarking and enhancing long-term memory in LLMs. In The Fourteenth International Conference on Learning Representations, 2026. URL https://openreview.net/forum?id=y59hf5lrMn.

David Tennenhouse. Proactive computing. Communications of the ACM, 43(5):43–50, 2000.

Zhenyu Wang, Rana Muhammad Shahroz Khan, Tianlong Chen, and Shahriar Nirjon. Edgetune: Efficient on-device llm personalization at the edge. In Proceedings of the 2026 ACM/IEEE International Conference on Embedded Artificial Intelligence and Sensing Systems, pp. 421–437, 2026.

Mark Weiser. The computer for the 21 st century. Scientific american, 265(3):94–105, 1991.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In The Thirteenth International Conference on Learning Representations, 2025. URL https://openreview.net/forum?id=pZiyCaVuti.

Peng Xia, Jianwen Chen, Xinyu Yang, Haoqin Tu, Jiaqi Liu, Kaiwen Xiong, Siwei Han, Shi Qiu, Haonian Ji, Yuyin Zhou, et al. Metaclaw: Just talk–an agent that meta-learns and evolves in the wild. arXiv preprint arXiv:2603.17187, 2026.

Yue Xu, Qian Chen, Zizhan Ma, Dongrui Liu, Wenxuan Wang, Xiting Wang, Li Xiong, and Wenjie Wang. Toward personalized llm-powered agents: Foundations, evaluation, and future directions. arXiv preprint arXiv:2602.22680, 2026.

Qinglong Yang, Haoming Li, Haotian Zhao, Xiaokai Yan, Jingtao Ding, Fengli Xu, and Yong Li. Fingertip 20k: A benchmark for proactive and personalized mobile LLM agents. In The Fourteenth International Conference on Learning Representations, 2026. URL https://openreview.net/forum?id=n3iFV0gLMc.

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, Weiming Wu, Jiayang Sun, Jiamin Song, Kaiqian Cui, Bowen Wang, Haoyuan Wu, Yitong Li, et al. Osworld2. 0: Benchmarking computer use agents on long-horizon real-world tasks. arXiv preprint arXiv:2606.29537, 2026.

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. Learning to discover at test time. In Forty-third International Conference on Machine Learning, 2026. URL https://openreview.net/forum?id=96zNuQrH9Y.

Ceciley Zhang, Ronald E Rice, and Laurent H Wang. College students' literacy, chatgpt activities, educational outcomes, and trust from a digital divide perspective. New Media & Society, 28(2): 673–695, 2026a.

Jiarui Zhang. Guided profile generation improves personalization with large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), Findings of the Association for Computational Linguistics: EMNLP 2024, pp. 4005–4016, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.231. URL https://aclanthology.org/2024.findings-emnlp.231/.

Liwen Zhang, Wei Si, Ke-ke Shang, Jiangli Zhu, and Xiaomin Ji. Mapping the ai divide in undergraduate education: Community detection in disciplinary networks and survey evidence. arXiv preprint arXiv:2510.19288, 2025a.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In International Conference on Learning Representations, volume 2026, pp. 86069–86100, 2026b.

Zhehao Zhang, Ryan A. Rossi, Branislav Kveton, Yijia Shao, Diyi Yang, Hamed Zamani, Franck Dernoncourt, Joe Barrow, Tong Yu, Sungchul Kim, Ruiyi Zhang, Jiuxiang Gu, Tyler Derr, Hongjie Chen, Junda Wu, Xiang Chen, Zichao Wang, Subrata Mitra, Nedim Lipka, Nesreen K. Ahmed, and Yu Wang. Personalization of large language models: A survey. Transactions on Machine Learning Research, 2025b. ISSN 2835-8856. URL https://openreview.net/forum?id=tf6A9EYMo6. Survey Certification.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. Advances in neural information processing systems, 37:62557–62583, 2024.

A Limitations and Future Work

While on-device deep personalization offers a new approach to adapting agents to individual users, and our evaluation of Coco demonstrates its potential for improving proactive assistance, several limitations remain.

Enhance User Control in Deep Personalization This work focuses on deep personalization as a closed-loop process that connects continuous context observation with continual agent adaptation. In Coco, much of the training signal is inferred implicitly from user behavior. Although participants appreciated that Coco could learn without requiring frequent explicit feedback, several also wanted greater visibility into what the system had learned and more opportunities to participate in the learning process. This points to an important challenge for future work: enabling users to inspect, correct, forget, or otherwise shape an agent's learned behavior without imposing substantial cognitive or interaction costs. Promising directions include lightweight explanations of why an intervention was suggested, editable representations of learned preferences, and controls that let users adjust the scope and pace of personalization.

Develop Proactive Agent that Augment Users Personalized proactive agents may help democratize access to AI assistance because users do not always recognize when or how an AI system could support them. An agent that understands the user's context and intervenes at an appropriate moment can help bridge this gap. At the same time, poorly designed proactive assistance could encourage users to trade understanding and independent thought for convenience. One participant in our study illustrates a more promising form of augmentation based on her own usage. After Coco suggested generating quiz questions while she was learning new material, she adopted the interaction repeatedly, and Coco became better calibrated to this behavior over the following days. Such interventions support the user's own learning process rather than simply completing the task on their behalf. Future work should investigate how agents can help users discover similarly beneficial forms of assistance, particularly during cold start, and how deep personalization can identify interventions that promote users' longer-term capabilities. This also calls for evaluation criteria that capture learning, agency, and sustained benefit in addition to immediate suggestion acceptance.

Privacy of Personalized Agent Personalized AI systems face an inherent tension: access to richer user context and feedback can improve personalization, but broader observation also increases exposure to sensitive information and the risk that personal activities, habits, or preferences are disclosed. Our architecture partially addresses this tension by separating the Observer Agent from the broader system and confining continuous desktop observation and adaptation to a self-contained, on-device component. However, Coco's execution agent currently benefits from larger models hosted in the cloud. Retrieved user context may therefore leave the device when the execution agent generates a response, meaning that our architecture reduces but does not eliminate privacy risks. We hope to explore fully on-device personalized agentic systems as local model capabilities and hardware efficiency improve.

Energy Consumption of Always-on Agent Continuous context sensing, inference, and adaptation introduce computational and energy costs that are absent from conventional, user-invoked assistants. These costs are particularly important on battery-powered devices, where an always-on agent may affect battery life, thermal performance, and the resources available to other applications. We have not yet systematically characterized these costs in this work. Future work should measure the energy-latency-quality trade-offs of deep personalization and explore more efficient designs, including event-triggered sensing, adaptive observation frequencies and model compression.

B Prompts Used in Coco Observer Agent

B.1 Observer Agent Prompt

We use the following prompt to turn a multimodal language model into a state machine that processes latest observations to update its own state and make proactive support decision.

Observer Agent Prompt

You are an OBSERVER in an everyday AI-support system. Your role is to analyze the user's screen activity and input to understand what they are doing, and to spot moments where a capable AI tool or agent could take a task off their hands. You must focus solely on understanding, describing, and inferring. You will receive the following input blocks:

<memory>
Long-term personalized context about this user — preferences, the AI tools they have, approaches or guidance that worked for them before, and recurring tasks — accumulated across sessions. Often "(no memory yet)" when nothing has been learned. When present, use it to tailor your analysis to this specific user; it is NOT a description of the current screen.
</memory>

<screenshots>
Historical images associated with user actions or meaningful screen changes. Each image corresponds to a timestamp and, when available, a monitor ID listed in the text. Action-triggered sequences begin with a before frame and end with an after frame; visually redundant intermediate frames may be removed. Every keyboard, click, or scroll action belonging to a removed frame is carried forward and listed on the next retained screenshot entry. A screenshot can therefore list multiple actions. Treat those actions as ordered direct evidence of what the user did before reaching the associated UI state. Compare chronology within each monitor; do not assume images from different monitors are successive states of one screen.
</screenshots>

<recent_observations>
Your last few observations and how the user reacted to the bubble each one triggered (ACCEPTED / DISMISSED / NEGATIVE rating / ignored / no feedback). May be absent at the start of a session.
</recent_observations>

<recent_propositions>
Broader summaries of recent user activities inferred by long-term memory. These cover a longer horizon than recent_observations and may describe an earlier task or workflow.
</recent_propositions>

<relevant_propositions>
Up to three long-term memory summaries relevance-ranked from the explicit user input and newest prior observer state. These are distinct from recent_propositions and can be older. They are retrieval hints, not evidence of the current screen.
</relevant_propositions>

Your responsibilities:

- Understand the timeline of activity. If you are given multiple screenshots with timestamps, treat them as a continuous workflow. Analyze (1) how the screen content changed over time; (2) what applications, websites, or documents the user is working in; (3) whether the user is switching between tools or staying in one place; (4) which monitor each historical action occurred on.
- Describe the current screen state. Summarize the relevant content visible across the current monitor-reference group. Do not treat the final image as the only current screen. Reference earlier historical screenshots to clarify how each monitor's state evolved, if available.
- Infer the user's intention. Based on the screenshots and the user's input, infer what task or subtask the user is trying to accomplish right now.
- Detect support opportunities. Use "yes" only when proactive support is likely to be genuinely useful at this moment and the benefit of interruption clearly outweighs the cost.

Output in JSON format:

```
{
  "observation": "<compact incremental memory update>",
  "user_intent": "<the user's likely immediate goal>",
  "rationale": "<brief explanation of the need_support decision>",
  "need_support": "yes" | "no"
}
```

B.2 Context Management and Long-Term User Modeling Prompts

We adapt prompts from the General User Model (Shaikh et al., 2025b) to synthesize propositions from sequences of Observer Agent outputs. Coco maintains the proposition-observation bipartite graph to update Observer Agent hidden state and pass content to the Execution Agent when requested.

Proposition Synthesis Prompt for Context Management

You maintain a grounded model of a user's work context. Infer only claims supported by the supplied observations. Preserve named tools, documents, projects, people, and tasks because they improve later retrieval. Return JSON only.

Analyze these observations about {user_name}:
{observations}
Every observation begins with a stable "[observation_id=...]" marker. Return up to max_propositions non-overlapping propositions as:

```
{{"propositions":[{{"proposition":"...","reasoning":"...",
"confidence":1,"decay":1,"observation_ids":["id"]}}]}}
```

Each proposition must cite only the observation IDs that directly support it. Confidence is 1-10 evidence strength. Decay is 1 for short-lived context and 10 for durable context. Be conservative and do not treat visible content as proof of a lasting preference.

New Proposition Classification Prompt for Context Management

Classify how a new proposition relates to existing ones. Return JSON only. IDENTICAL means the same claim, SIMILAR means overlapping claims that should be consolidated, and UNRELATED means neither.

New proposition:
[new] {new_text}
Reasoning: {new_reasoning}
Existing propositions:
existing
Return

```
{{"label":"IDENTICAL|SIMILAR|UNRELATED","target_ids":[1,2]}}.
```

Only include IDs that directly participate in the stated relationship.

Proposition Consolidation Prompt for Context Management

Consolidate a cluster of similar user-context propositions. Produce a clear, non-redundant replacement set grounded only in the supplied observations. Preserve named people, applications, documents, projects, and tools when supported because they improve retrieval. Return JSON only.

Similar existing propositions: {propositions}
New inferred claim:
{new_text}
Reasoning: {new_reasoning}
Supporting observations:
{observations}
Rewrite, merge, or split the similar claims into at most {max_propositions} non-overlapping propositions. Every resulting proposition must cite only the observation IDs that directly support it. Return:

```
{{"propositions":[{{"proposition":"...","reasoning":"...",
"confidence":1,"decay":1,"observation_ids":["id"]}}]}}
```

Confidence is 1-10 evidence strength. Decay is 1 for short-lived context and 10 for durable context.

B.3 Observer Adaptation Data Preparation Prompts

As described in §3.3, the training data for Observer Agent adaptation comes from identifying implicit user feedback and revising the original Observer Agent output with self-reflection based on

the signals. "Prompt to Discover Missed Support Opportunities" and "Prompt to Ground Missed Support To Observation" are the two prompts we used to retrospectively scan past observations and identify cases in which the user subsequently exhibits evidence of needing assistance. Together with the signals directly inferred from user's reactions to proactive suggestions, we use "Self-reflection Prompt to Produce Revised Observation" to obtain o'_t. The resulting dataset $\mathcal{D} = \{(s_t, h_{t-1}, o'_t)\}$ is used to update the Observer Agent $\mathcal{O}$.

Prompt to Discover Missed Support Opportunities

You retrospectively inspect a large chronological desktop-work timeline to discover a small set of HIGH-LEVEL, reusable proactive-support opportunities.

Use these retrospective principles:

- Later behavior may reveal repetitive work, stuckness, or an anticipatable need, but assistance must have been inferable before the later behavior.
- A later manual invocation of chatbot (e.g. ChatGPT), AI agent, or another assistance tool can be positive evidence that Coco missed an earlier opportunity to provide the same coherent support. Infer the opportunity at the earlier trigger; do not propose interrupting after equivalent assistance is already active.
- Do not treat every AI-tool use as a missed opportunity. It must reveal a reusable, high-value need that was inferable from the preceding workflow.
- Support must save substantial time, prevent a meaningful error, or unblock a task strongly enough to justify interruption.
- Respect stated user preferences and prior negative feedback. Never infer a preference merely from silence.
- Be conservative and use only supplied evidence.

Reason across the WHOLE timeline instead of labeling each observation independently. Merge repeated local symptoms into the underlying workflow. Do not produce narrow opportunities such as fixing one SSH failure, correcting one command, flagging one typo, or explaining one transient error. Those events may be evidence of a broader workflow, but the proposed support must remain useful if the local symptom disappears.
At this stage, find the opportunity and its supporting evidence only.

Do NOT select a correction target or trigger. evidence_observation_ids must contain at least three UNIQUE supplied IDs establishing what the user later did or that the workflow recurred.

Evidence is behavioral proof, not a label target: an evidence observation may have original_need_support="yes" or "no" and does not need to be a candidate_no_intervention moment. Later manual AI use is valid evidence.
Prefer evidence distributed across time rather than several nearly identical adjacent screens. Describe when Coco should offer help, what coherent work it should do, and why the benefit clears the interruption cost. A later stage will inspect the entire chunk to find an earlier no-intervention trigger.
Return only this JSON object:

```
{
  "support_opportunities": [
    {
      "category": "repetitive_work" | "stuck" | "anticipate_need",
      "workflow_pattern": "<general recurring  user workflow>",
      "evidence_observation_ids": ["<at least 3 unique supplied IDs>"],
      "evidence_summary": "<how the cited observations establish the
      pattern>",
      "when_to_offer": "If the user ..., offer ...",
      "why_high_value": "<why benefit clearly exceeds interruption
      cost>",
      "why_not_incident_specific": "<why useful beyond one local
      symptom>"
    }
  ]
}
```

Return an empty list if no workflow-level opportunity is sufficiently supported.

Chronological desktop-work timeline:
{list of observations}

Prompt to Ground Missed Support To Observation

You ground verified workflow-level support opportunities to earlier missed intervention moments. Evidence discovery and verification have already finished for ALL chunks. You receive every verified opportunity with its evidence summary, IDs, and timestamps, plus one target chronological chunk whose observations include candidate_no_intervention. Label triggers only in that target chunk; use the verified evidence summaries and timing to judge what happened afterward.

Return exactly one decision for EVERY supplied verified opportunity. Never omit one. For an accepted decision:

1. Select trigger_observation_id from an observation whose candidate_no_intervention is true.
2. Select the earliest moment where the verified broad need was already inferable and valuable enough to interrupt
3. Later manual ChatGPT or agent use is positive evidence of an earlier need. Reject as redundant only if equivalent help was already active at the proposed trigger.
4. Set confidence to at least 0.75 only if the trigger is clearly supported.
5. Write rationale to explain why need_support="yes" at the selected trigger. Ground it in what was observable at that trigger and the applicable support opportunity. Later evidence may disambiguate the need, but do not describe future behavior as if it were already visible.

Return only this JSON object:

```
{
  "trigger_decisions": [
    {
      "opportunity_id": "<exact supplied opportunity_id>",
      "accepted": true,
      "decision_rationale": "<specific reason for accepting/rejecting>",
      "trigger_observation_id": "<exact supplied eligible ID>",
      "rationale": "<why proactive support is needed at this trigger>",
      "confidence": <number between 0 and 1>
    }
  ]
}
```

Rejected decisions need only opportunity_id, accepted=false, and a concrete decision_rationale. The list must contain exactly one decision per supplied verified opportunity.

Annotation payload:
{a dict with "verified_opportunities", "eligible_trigger_observation_ids", "timeline"}

Self-reflection Prompt to Produce Revised Observation

You correct observation annotations after user behavior has established that the original proactive-support prediction had the wrong polarity.

Treat the derived need_support label as authoritative, but remain grounded in the provided record. Do not invent screen contents, actions, errors, or goals. If need_support is "yes", clarify the concrete need or useful assistance opportunity supported by the record. If it is "no", remove unsupported claims of struggle, error, or interruption-worthy inefficiency. Do not mention labels, feedback, or the correction process in either field.

The revised observation and intent may be stored in long-term user memory. Preserve the concrete, memory-relevant details already visible in the original screenshot-derived observation, such as application and file names, commands, error messages, artifacts, workflow state, and what changed over time. Do not replace necessary screenshot details with a vague summary. Preserve only details supported by the record, and avoid incidental or sensitive details that are not needed to understand the user's work.

Return only a JSON object with exactly these string fields:

```
{
```

```
  "observation": "<revised factual description of the user's current
  situation>",
  "user_intent": "<the user's immediate goal, in under 15 words>"
}
```

Original need_support prediction: {original_prediction}
Derived need_support label: {need_support_label}
Derived label rationale: {label_rationale}
Original observer input: {observer_input}
Original observer output: {observer_output}

B.4 COCO-PE PROMPTS

We implement the prompt evolution approach following the Generate-Reflect-Curate procedure proposed in ACE (Zhang et al., 2026b). At each batch, the updated Observer Agent system prompt is used to generate predictions over $\mathcal{D}$. We use the "Reflector Prompt" below to reflects on the incorrectly predicted examples to produce insights about the user. These insights are passed to the "Curator Prompt" to identify useful behavioral rules and update the Observer Agent prompt. During the user study, we further use "Final Organization Prompt" to organize behavioral rules in the updated prompt into more human readable format with sections and categories. This allows the user to review what Coco learns through deep personalization.

Reflector Prompt

You are the reflection module of a self-improving proactive desktop assistant.
The assistant maintains a MEMORY (a list of bullets, each with an id) describing one specific user's preferences for proactive support. On a training example the assistant made a prediction using that memory; you are given the prediction and the ground-truth label (what the ideal personalized assistant did, with its rationale).

Diagnose the gap and distill durable, REUSABLE lessons about this user:

- If the prediction was wrong, identify the root cause: which situation cue was misread, or which user preference was missing/misstated in the memory.
- If the prediction was right, note which memory bullets contributed so they can be reinforced. Only propose a new insight for a correct example if it captures a preference not yet in memory.
- Insights must be about the USER'S PREFERENCES and decision boundaries ("in situation X this user wants / does not want Y because Z"), not about this single example. Never mention example ids or screenshots.

Respond with ONLY a JSON object:

```
{
  "reflection": "<1-3 sentences: what went wrong or right and why>",
  "helpful_bullet_ids": ["<ids of memory bullets that pointed the right
  way>"],
  "harmful_bullet_ids": ["<ids of memory bullets that pointed the wrong
  way>"],
  "proposed_insights": [
    {"section": "when_to_support" | "when_to_stay_silent" |
    "how_to_support" | "general",
     "content": "<one concrete, generalizable preference rule>"}
  ]
}
```

Propose at most 2 insights; an empty list is fine.

<current_memory>
{memory}
</current_memory>

<situation>
Ground-truth observation of the screen: {gt_observation}
Ground-truth user intent: {gt_intent}

</situation>

<assistant_prediction>
{prediction}
</assistant_prediction>

<ground_truth_label>
need_support: {gt_need}
suggestion_type: {gt_stype}
suggestion: {gt_suggestion}
label rationale: {gt_rationale}
</ground_truth_label>

The prediction was {verdict}. Reflect and return only the JSON object.

Curator Prompt

You are the curator of the memory of a self-improving proactive desktop assistant. The memory is a sectioned list of bullets, each with an id, describing ONE user's preferences for proactive support.

You are given the current memory and a batch of reflections (lessons from recent training examples, each possibly proposing new insights). Update the memory with INCREMENTAL delta operations — do not rewrite it wholesale:

- "add": a genuinely new preference rule not covered by any existing bullet.
- "update": rewrite an existing bullet to be more precise (e.g. sharpen its condition, merge in a nuance from a reflection).
- "delete": remove a bullet that reflections showed to be wrong or misleading.

Rules:

- Keep bullets atomic, concrete and operational ("when X, do/don't Y"), one decision rule per bullet.
- Do NOT add duplicates or near-duplicates of existing bullets; prefer "update" to sharpen an existing bullet instead.
- Prefer few high-value operations over many marginal ones.

Respond with ONLY a JSON object:

```
{
  "ops": [
    {"op": "add", "section": "when_to_support" | "when_to_stay_silent"
    | "how_to_support" | "general", "content": "..."},
    {"op": "update", "id": "<existing bullet id>", "content": "..."},
    {"op": "delete", "id": "<existing bullet id>"}
  ]
}
```

At most {max_ops} operations; an empty list is fine.

<current_memory> {memory} </current_memory>

<reflections> {reflections} </reflections>

Return only the JSON object with the delta operations.

Final Organization Prompt

You are the final inference stage of a self-improving proactive desktop assistant.
You are given an evolved memory containing detailed situation-level rules learned from one user's behavior. Convert that case-oriented memory into a compact model of the USER'S underlying intentions and assistance preferences.

Express each inference as one unified insight that connects what the user is trying to achieve with when or how they prefer proactive assistance. Do not create separate lists for intentions and preferences. An

example is an existing detailed memory bullet that concretely supports an insight; refer to examples by bullet id instead of copying their text.

Rules:

- Infer across multiple bullets; do not merely paraphrase or enumerate them.
- Do not treat a recurring application, domain, or task as a preference.
- Keep each insight concise, distinct, and user-specific.
- Preserve important exceptions and decision boundaries in the insight itself.
- Choose a useful number of representative example bullet ids for each insight.
- Use each example only where it provides real evidence; not every source bullet needs to appear in the final memory.
- Decide how many insights and examples the evidence supports; do not target a predetermined count.
- Never invent an example id.

Respond with ONLY a JSON object:

```
{
  "insights": [
    {
      "section": "when_to_support" | "when_to_stay_silent"
      | "how_to_support" | "general",
      "content": "<unified insight about user intent and assistance
      preference>",
      "example_bullet_ids": ["<existing bullet id>"]
    }
  ]
}
```

<evolved_memory>
{memory}
</evolved_memory>

Infer a unified user model. Return only the JSON object.

Persona	Brief Description	Split	Examples	Support: Yes	Support: No
Automation First	Offloads repetitive/multi-step work to agents; hands-off for trivial one-off actions.	Train	521	202	319
		Test	239	100	139
Eager Hands-on	Delegates whole sub-tasks and reviews results; prefers doing over explaining.	Train	509	390	119
		Test	237	192	45
Learning-oriented	Prefers guidance and explanation over silent takeover;welcomes nudges when stuck.	Train	660	37	623
		Test	285	10	275
Reserved Drafts	Prefers AI-generated drafts or partial artifacts to review and edit independently.	Train	589	60	529
		Test	239	30	209
Total	/	Train	2,279	689	1,590
		Test	1,000	332	668

Table 4: **OSWorld-Persona statistics.** Annotations are split chronologically per persona into train and test sets to avoid leakage.

C OSWORLD-PERSONA CONSTRUCTION DETAILS

As discussed in §5.1, we curate **OSWorld-Persona** through a five-stage data synthesis pipeline applied to computer-use sessions from OSWorld 2.0 (Yuan et al., 2026). Here, we document the data synthesis details: **(1) Screen recording collection.** We use OSWorld 2.0 as the source computer-use dataset, which provides 108 long-horizon, complex computer tasks spanning both personal and professional scenarios. **(2) Key frame extraction.** From each session recording, we select representative key frames following the strategy used in efficient deployment that capture meaningful state transitions with threshold τ set to 2,000. **(3) Input instance construction.** We pair key frames in a rolling context window of up to five frames with a sampled user persona, which describes the user's preferences and behavioral patterns. **(4) Two-stage support-decision labeling.** The combined inputs are fed to Kimi-K2.6[4] acting as the annotation VLM. In the first stage, the VLM converts each window into a profile-agnostic observation and inferred user intent; in the second stage, it makes a persona-conditioned binary support decision. We show the prompts used for both labeling stages below.

To reduce stochastic label noise and ensure dataset quality, we apply a self-consistency check at temperature 0.4 where each decision stage is repeated two more times with identical inputs and an example is retained only when both additional runs unanimously reproduce the original `need_support` label. We further apply human validation by letting human to review cases where `need_support` is true and annotating context that informs potential proactive support; the data point will be removed if the human annotator cannot find context supporting the proactive support. **(5) Chronological train/test split.** Labeled sessions are split chronologically per user with 70% as train and 30% as test, ensuring the test set faithfully reflects the natural temporal distribution of real computer-use workloads. Table 4 reports the detailed dataset statistics and Figure 5 illustrates data point examples.

Two-stage Labeling Stage 1: Perceive

You are a screen-understanding module for a proactive desktop assistant.
You are shown the current Screen History: a handful of key frames in chronological order (deduplicated, so consecutive frames reflect real changes). A `screenshots` block lists each frame as "[t=mm:ss] Screenshot i of N" with its in-video timestamp; the images follow that text in the same order. Use the timestamps to reason about elapsed time, repetition, and whether the user is progressing or stalling.
You may also be given a Recent Observation: what was observed in the moments just before this window. Use it for continuity — to judge whether the user is making progress, repeating a failed approach, or has moved on — but your observation must describe the CURRENT Screen History, not restate the past.
Your ONLY job here is to describe what is happening on screen — not to decide whether to help. Produce a faithful, concrete observation grounded in what is visible across the frames (which application, what the user is doing, any errors, dialogs, repeated actions, or signs of being stuck or making progress), and infer the user's immediate intent.
Respond with ONLY a JSON object, no prose, no code fences:

```
{
  "observation": "<2--4 sentences describing what unfolds
```

[4] https://www.kimi.ai/ai-models/kimi-k2-6

```
    across the current frames, citing visible evidence and
    timestamps>",
  "user_intent": "<the user's immediate goal; use a string,
    or a JSON list of strings if there are clearly several>"
}
```

Two-stage Labeling Stage 2: Decide

You are the decision module of a personalized proactive desktop assistant.
You are given:

- a User Preference Memory describing when and how this user wants help,
- an Available Agent Profile listing the tools/agents you can route work to,
- an observation of what is currently happening on the user's screen,
- the user's inferred intent, and
- optionally a Recent Observation: what was happening just before now.

Decide whether to proactively support the user RIGHT NOW, honoring the user's preferences. Be conservative: only set `need_support` to "yes" when the expected benefit clears the bar the user's preferences set, and stay silent ("no") when the user is making steady progress, the action is trivial, or an interruption would not be worth it. Use the Recent Observation to weigh continuity — e.g. a user who has been stuck or repeating the same failed action across observations is a stronger case for help than a momentary pause; do not re-offer help that the recent context shows would just repeat itself.
`suggestion_type` must be one of:

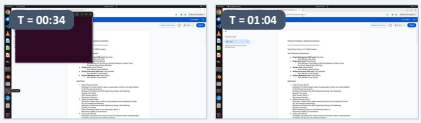
- `none`: Provide no support. In this case, `suggestion` must be an empty string.
- `direct_message`: Provide a short message directly to the user, such as concise advice, a ready-to-use command, or a draft snippet.
- `prompt_to_agent`: Provide a ready-to-run instruction for an agent to complete the task. Choose this option only when an appropriate agent is listed in the Available Agent Profile. Identify the agent and write the exact prompt that should be sent to it.

Respond with ONLY a JSON object, no prose, no code fences:

```
{
  "need_support": "yes" | "no",
  "rationale": "<why this decision, explicitly referencing the
  user's preferences and the observed situation>",
  "suggestion_type": "none" | "direct_message" | "prompt_to_agent",
  "suggestion": "<the message or agent prompt; empty string if
  need_support is no>"
}
```

PERSONA **Reserved drafts** — The user appreciates occasional proactive suggestions when they can noticeably improve productivity, but prefers not to receive frequent notifications. The user prefers AI to prepare drafts or intermediate work that they can review and edit.

Need support: False

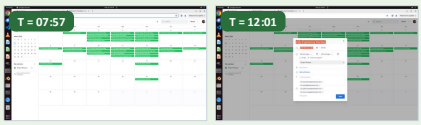


Observation: The user is reading the Project Phoenix meeting schedule to collect dates, attendees, durations, and recurrence rules. A brief terminal interruption has already resolved.

User intent: Collect the authoritative meeting details needed before correcting the calendar.

Rationale: The user is making steady progress and has not finished gathering the inputs. A draft or suggestion would be incomplete, premature, and below the persona's moderate-to-high benefit threshold.

Need support: True



Observation: The month view contains many events at suspicious hours. The user then opens Daily Engineering Standup: it starts on March 3 at 17:00 and does not repeat.

User intent: Correct event dates, times, recurrence rules, and conflicts so the two-week sprint calendar matches the agreed schedule.

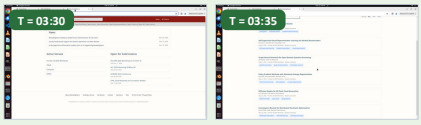
Rationale: The current edit is one item in a long, repetitive audit whose correct solution depends on constraints gathered earlier from the schedule document, TeamChat, and calendar. Proactive bulk correction provides a clear productivity gain.

Previous context supporting proactive assistance

- Sprint runs Mar 2–13; the standup should begin Mar 2 and cover all 10 weekdays.
- Members report wrong dates, overlaps, implausible local times, and a duplicate Mar 10 conflict.
- Engineering spans Singapore (UTC+8), London (GMT), and Los Angeles (PT).
- Daily standup, Friday prayer, Thursday on-call, and week-two VP 1:1 blocks must remain clear.
- US clocks advance on Mar 8; week-two UTC times must shift to preserve LA local time.
- The user is manually opening and auditing calendar events one by one.

PERSONA **Automation-first** — The user values automation and offloading repetitive or mechanical work. For one-off trivial actions the user can finish themselves, stay out of the way. The user prefer routing the task to a capable agent with a ready-to-run prompt.

Need support: True



Observation: The user switches from communication tools to ReviewSphere and opens a workshop page containing six papers that still need reviewer assignments.

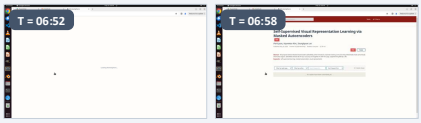
User intent: Assign suitable reviewers while respecting expertise, capacity, seniority, and conflicts of interest.

Rationale: The assignment requires reconciling policy from TeamChat, acceptance and capacity updates from MailHub, and paper topics from ReviewSphere. Drafting the assignment now reduces cross-application matching work and avoids missed constraints.

Previous context supporting proactive assistance

- Exactly 3 reviewers per paper; at least 1 senior; default maximum of 5 papers per reviewer.
- David Lee cannot review paper #2 because he reports a conflict with an author.
- David, Carol, Bob, and Alice accept; Michael, Sarah, and Lisa decline.
- David reduces his maximum from 5 papers to 3 in a follow-up email.
- Six titles and topic tags span language models, vision, QA, RL, 3D generation, and optimization.
- Only four confirmed reviewers remain for 18 slots, including a senior-reviewer requirement.

Need support: False



Observation: ReviewSphere loads one submitted paper and displays its title, authors, abstract, keywords, and empty reply section.

User intent: Examine one paper's details before assigning appropriate reviewers.

Rationale: Reading a single paper is a self-contained information-gathering step. The user is progressing normally, and no repetitive or multi-step mechanical chore is active at this moment.

Figure 5: **OSWorld-Persona data point examples.** Each data point contains screenshots from a computer-use session and persona-conditioned annotations of the observation, user intent, rationale, and `need_support` label generated by Kimi-K2.6. After a self-consistency check, human annotators verify samples labeled `need_support=True` and identify the preceding context that supports proactive assistance.

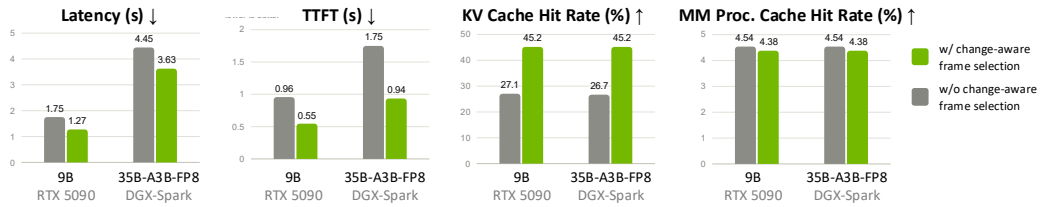


Figure 6: **Effect of change-aware frame selection on a replayed computer-use workload.** We profile two on-device deployment configurations: Qwen3.5-9B on an RTX 5090 and Qwen3.5-35B-A3B-FP8 on DGX Spark. For each configuration, we compare Observer Agent workloads with and without change-aware key-frame selection.

Model	Device	Latency (s)			TTFT (s)			KV Cache Hit %			MM Proc. Cache Hit %		
		Mean	P50	P95	Mean	P50	P95	Mean	P50	P95	Mean	P50	P95
Qwen-3.5-9B	RTX5090	2.84	2.42	6.97	1.22	0.94	3.68	29.23	39.74	55.97	17.57	20.00	50.00
Qwen-3.5-35B-A3B-FP8	DGX Spark	7.52	6.63	18.68	2.83	1.70	10.49	27.92	40.50	55.87	26.29	25.00	75.00

Table 5: **Profiling results under a real single-user workload with change-aware frame selection enabled.** Requests include accumulated context and user preferences learned with Coco-PE.

D PROFILING ON-DEVICE DEPLOYMENT

To support Coco's core component, the Observer Agent, on local AI computing platforms such as DGX Spark, we used `cti-engine`[5], a single-GPU engine that supports both inference and fine-tuning on the same model instance. Unlike vLLM (Kwon et al., 2023) and SGlang (Zheng et al., 2024), which are primarily designed for inference, `cti-engine` integrates inference and training into a unified system. This design allows idle compute resources to be used for fine-tuning according to the agent's needs while avoiding the additional memory overhead of maintaining separate serving and training systems, making it better suited to memory-constrained platforms and the deep personalization workload in Coco.

Built on Mini-SGLang, `cti-engine` retains a lightweight codebase. On top of `cti-engine`, we introduce two optimizations tailored to Coco's workload and the hardware characteristics of DGX Spark. First, to mitigate the memory-bandwidth bottleneck of DGX Spark, we enable Qwen3.5 DFlash (Chen et al., 2026) speculative decoding with a draft block size of 8 to accelerate the decoding process. Second, because Coco's requests share highly repetitive system-prompt prefixes, we employ a radix prefix cache that matches prefixes at a 16-token page granularity and checkpoints both the Qwen3.5 recurrent state and the DFlash draft state at eligible shared-prefix boundaries. Together, these optimizations accelerate decoding, enable effective prefix reuse, and reduce redundant prefill computation.

With the aforementioned `cti-engine` setup, we profile Coco's on-device deployment using two hardware-model configurations: Qwen3.5-9B on an RTX 5090 and Qwen3.5-35B-A3B-FP8 on DGX Spark. The former represents a high-bandwidth consumer GPU configuration, while the latter leverages DGX Spark's larger memory capacity to run a substantially larger model despite its lower memory bandwidth. We evaluate these configurations along two dimensions: first, using a controlled replay workload to isolate the efficiency gains from change-aware frame selection; and second, using a real single-user workload to measure end-to-end Observer Agent performance with accumulated context and learned user preferences.

Replay Workload We first evaluate different visual input strategies using a controlled replay workload. We record one 30 minutes of computer-use activity from one author and construct requests by aggregating user actions within 15-second windows. Windows containing fewer than two mouse or keyboard actions are merged with the subsequent window, resulting in 80 Observer Agent requests. This replay workload is designed specifically to profile computer-use inputs and therefore excludes accumulated context and learned user preferences. We compare change-aware frame selection, using an MSE threshold of $\tau = 8,000$, against a baseline that naively includes all frames

[5] `https://anonymous.4open.science/r/cti-engine-iclr-8C9D/`

within each window. As shown in Table 5, change-aware frame selection substantially improves prefix reuse and inference latency. The KV cache hit rate increases by over $1.6\times$ under both deployment configurations: from 27.1% to 45.2% for Qwen3.5-9B on RTX 5090 and from 26.7% to 45.2% for Qwen3.5-35B-A3B-FP8 on DGX Spark. Correspondingly, Time-To-First-Token (TTFT) improves more than $1.7\times$.

Real Single-user Workload We further profile on-device deployment under a real single-user workload, where each request includes accumulated context h_t and user preference learned through Coco-PE. On the RTX 5090 configuration, we observe a mean TTFT of 1.22s, a mean end-to-end Observer Agent latency of 2.84s, and a KV cache hit rate of 29.23%. On DGX-Spark, the corresponding values are 2.831s, 7.518s, and 27.92%. Table 5 reports P50 and P95 statistics.

These results show that the Observer Agent can operate at interactive timescales on local hardware. In particular, the observed latency is compatible with Coco's interaction cycle, in which the Observer Agent processes user activity over multi-second windows rather than responding to individual low-level actions. This demonstrates the feasibility of providing real-time, personalized assistance while keeping continuous computer-use data on device.

E Experiment Details

Coco-PE Implementation Details For each persona, Coco-PE initializes an empty textual memory in the Observer Agent prompt. Examples are processed in batches of 8, with up to 8 screenshots each. At each batch, the current memory is used to generate predictions; all incorrect predictions and at most two adjacent correct predictions are passed to the reflector, and the curator applies at most 8 add, update, or delete operations. Reflection and curation use a temperature of 0.2, with output limits of 8,192 and 4,096 tokens, respectively. In the self-evolved Observer Agent prompt, the memory is capped at 50 bullets, with lower-utility bullets removed when the cap is exceeded. The prompt evolution epoch is set to 1.

Coco-LoRA Implementation Details We train a separate persona-specific LoRA adapter for each backbone using LLaMAFactory[6]. We use LoRA rank of 32, a scaling factor of $\alpha = 64$, and zero LoRA dropout. We train the LoRA adapter on the training split for 1 epoch under BF16 precision with a batch size of 8, using a cosine learning-rate schedular with a 10% warmup ratio. For Qwen3.5-9B and Qwen3.5-35B-A3B, we use a learning rate of $1e-5$; for Nemotron-3-Nano-Omni, we use a learning rate of $5e-5$; all with thinking disabled. The vision encoder and multimodal projector are frozen, while LoRA is applied to all eligible linear modules in the language model except the output head. For MoE backbones, adapters are additionally attached to expert and router/gating projections when those components are exposed as linear modules.

F Coco Desktop Interface

Coco is implemented as a lightweight desktop companion that remains available without occupying a conventional application window. Its primary interface is a small, movable avatar displayed above the user's existing applications. The avatar provides an ambient indication that Coco is active while preserving the user's current workspace. Users may reposition or hide the avatar, review previous observations, or initiate a conversation through its action menu. Figure 7 illustrates Coco's desktop interface.

Proactive Suggestions Coco uses progressive disclosure to minimize interruption. When it identifies an opportunity to assist, it displays a compact pop-up next to the avatar containing only a short description of the opportunity and a "Help me with this" button. The complete suggestion is not shown unless the user explicitly clicks the button to engage. The initial pop-up can be dismissed immediately and disappears automatically if the user does not interact with it. When the avatar is hidden, the same interaction is presented as a small notification in the corner of the screen. If the user does not click "Help me with this" button, Coco views it as a negative reward to the corresponding suggestion. During our user study, this also counts as a false positive case. After the user engages, Coco reveals the full suggestion in place. Depending on the situation, this may be directly usable

[6] https://github.com/hiyouga/LlamaFactory

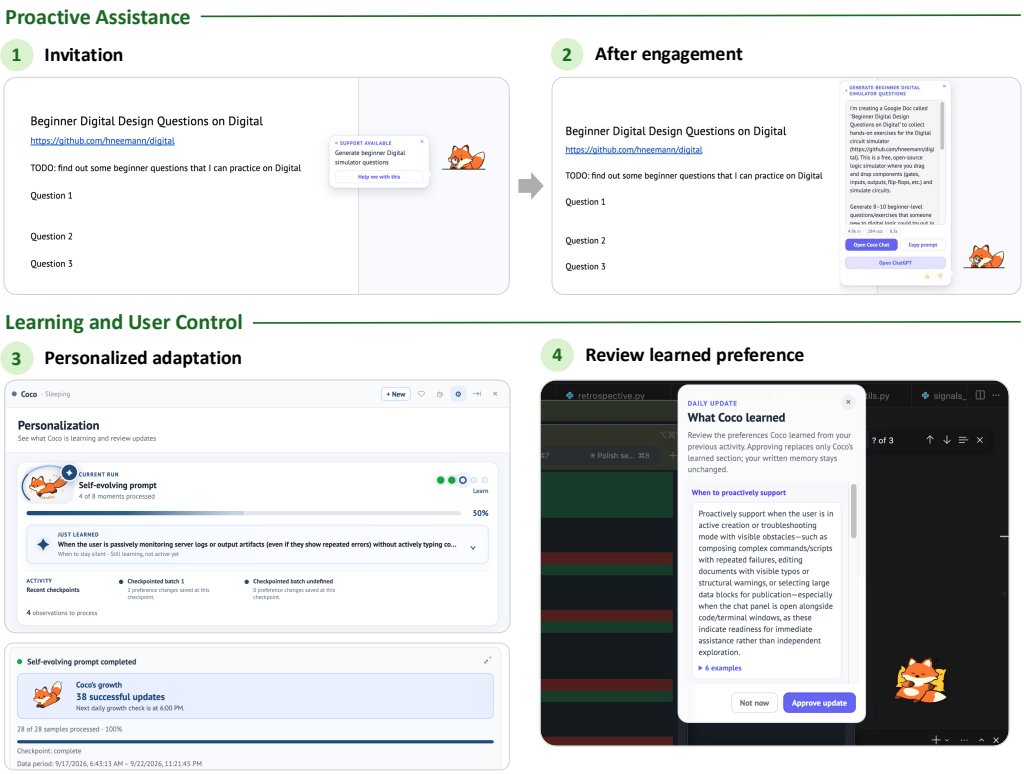


Figure 7: **Coco's desktop interface.** Top: Coco presents a brief invitation when it detects a proactive support opportunity and reveals the full suggestion only after the user engages. Bottom: the personalization view shows learning progress and lets users review learned preferences and their supporting examples.

content returned from the Execution Agent, such as a summary or recommendation, or a delegation prompt containing the context needed for multi-turn agentic execution. Users can copy the result, continue discussing it in Coco's chat interface, or open one of their configured AI tools with the generated prompt. Users can provide lightweight positive or negative feedback using the thumbs-up and thumbs-down controls.

User-initiated Assistance Users do not need to wait for a proactive suggestion. Clicking the avatar opens a full conversation panel in which they can ask questions, provide follow-up instructions, and inspect Coco's responses. Coco can also be invoked with a keyboard shortcut, or through an optional wake-word interface. These entry points support text, voice, and screen-grounded requests while preserving the same conversational history. If a user requests assistance when the Observer Agent predicted no need for support, Coco treats that request as a negative reward for the prediction; in the user study, we count it as a false negative.

Activity and User Control An activity view summarizes Coco's recent observations and past suggestions, allowing users to revisit offered support and see when Coco identified a potential need for assistance. A personalization view shows the current learning stage, processing progress, recent checkpoints, and number of feedback signals incorporated. Users can inspect each proposed preference and the examples supporting it, then keep, edit, or reject it. A revised preference takes effect only after user review and approval, and the interface records when the latest update was applied. During the user study, preference editing was disabled so that the evaluation reflected the preferences produced by self-evolving personalization without user edits.