CLAW-ANYTHING: BENCHMARKING ALWAYS-ON PERSONAL ASSISTANTS WITH BROADER ACCESS TO USER'S DIGITAL WORLD

Anonymous authors
Paper under double-blind review

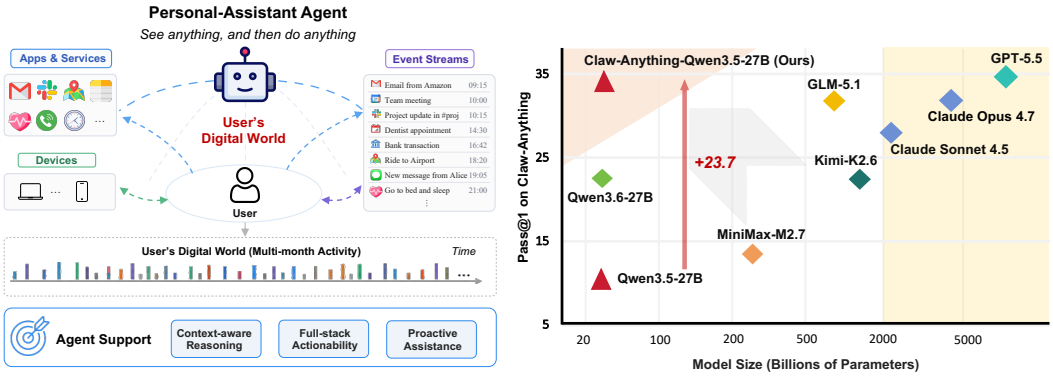


Figure 1: Overview of Claw-Anything and its empirical value. Left: Claw-Anything gives an always-on personal assistant broader access to the user's digital world, spanning services, devices, and long-horizon event streams, thereby expanding the range of tasks it can complete. Right: Enabled by our data pipeline, our model achieves the best Pass@1 among open-weight models. The yellow region represents closed-source models, and the horizontal axis does not correspond to model size.

ABSTRACT

Large language model agents are increasingly envisioned as always-on personal assistants with access to anything relevant in the user's digital world. Yet current systems operate over only narrow slices of that world, limiting context-sensitive reasoning and effective assistance. Existing benchmarks similarly provide only partial user state and therefore fail to capture performance in this broader setting. To address this gap, we introduce Claw-Anything, a benchmark that expands agent context along three dimensions: long-horizon activity histories, interdependent backend services, and integrated GUI and CLI interaction across multiple devices. To instantiate this setting, we simulate months of user activity through multi-round event injection, producing complex world states and realistic noise, including irrelevant events and conflicting signals. Agents must reason over rich contextual environments while remaining robust to such noise. This expanded scope also enables evaluation of heartbeat-conditioned proactive assistance, requiring agents to infer what to recommend at a fixed check time. Experiments show that GPT-5.5 achieves only 34.5% Pass@1, underscoring a gap between current agent capabilities and the demands of always-on personal assistance. Alongside the benchmark, we release an automated data-generation pipeline that yields 2,000 training environments; fine-tuning Qwen3.5-27B on successful trajectories improves Pass@1 by 23.7 percentage points, demonstrating the pipeline's utility for generating training data.

1 INTRODUCTION

Recent agent systems, such as the OpenClaw series (OpenClaw, 2026; HKUDS Teams, 2026a; NanoClaw Teams, 2026) and Hermes Agent (NousResearch, 2026), are moving beyond one-shot task solving toward always-on personal assistance. Deployed within users' digital environments and

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.

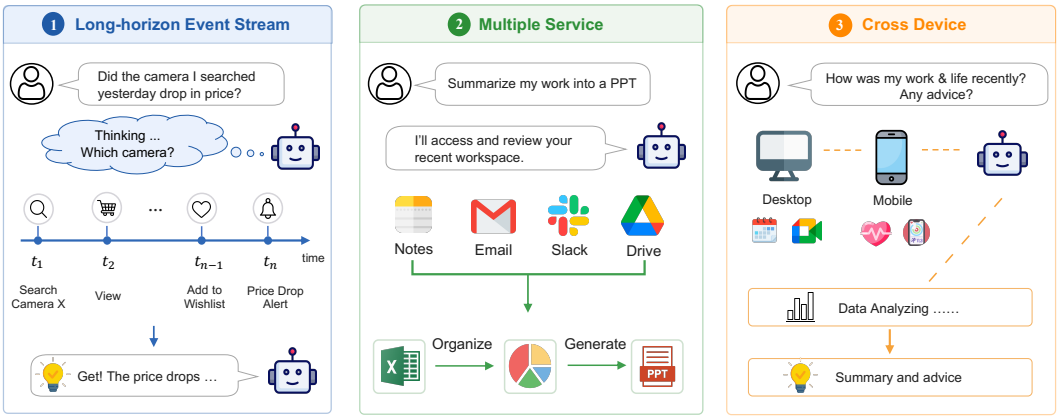


Figure 2: Three dimensions along which Claw-Anything expands agent context. **Left:** Long-horizon event streams provide a more complete view of the user's digital activity and support inference over evolving context. **Middle:** Access to multiple backend services enables cross-service coordination within a unified workflow. **Right:** Access across devices allows the agent to integrate distributed information and actions, broadening the range of tasks it can complete.

equipped with long-term memory and background execution, these systems are expected to provide continuous, context-sensitive support over time. Yet user intent and activity are inherently distributed across heterogeneous digital artifacts, including historical events, backend services, and multiple devices. Effective assistance therefore requires broad access to the user's digital world, so that an agent can both perceive relevant state and act on it in a closed loop.

Motivated by this shift, we argue that the effectiveness of personal assistants depends fundamentally on their operational scope: the set of digital states they can observe and the actions they can execute. As shown in Figure 1, expanding this scope enlarges both the task space an agent can address and the context over which it can reason, enabling coordination across otherwise disconnected parts of the user's digital world. Similar patterns appear in other areas of AI: coding agents require access to the full codebase and executable environment to resolve realistic bugs (Jimenez et al., 2024; Zhu et al., 2026a; Teams, 2026), while autonomous vehicles depend on broad sensor coverage for safe operation (Sun et al., 2020). Consistent with this trend, recent systems increasingly expose richer digital interfaces to agents. Open-source projects such as CLI-Anything (HKUDS Team, 2026) and Gym-Anything (Aggarwal et al., 2026), as well as commercial platforms such as Google Workspace (Google Workspace, 2026) and Feishu (Larksuite, 2026), provide unified interfaces or programmable endpoints, making diverse software systems accessible to agents. These developments indicate that widening an agent's operational scope is critical for enabling it to perform complex tasks across the real-world digital environment.

However, current evaluation paradigms remain poorly aligned with this objective. Existing benchmarks (Zhang et al., 2026; Ding et al., 2026; kilo.ai, 2026; Evolvent AI, 2026; Qwen Team, 2026) typically expose only narrow, static slices of user state, omitting long-horizon activity, cross-service dependencies, and interaction across devices. As a result, they provide limited evidence about how agents perform when operating in richer, more realistic digital environments. To address this gap, we introduce Claw-Anything, a benchmark for evaluating personal-assistant agents under substantially broader access to the user's digital world.

As illustrated in Figure 2, Claw-Anything expands agent context along three dimensions: i) long-horizon event streams that connect past and present through months of fine-grained activity records; ii) diverse, interdependent backend services spanning the principal digital spaces users inhabit; and iii) multiple devices with heterogeneous interfaces, including both GUI and CLI interaction. In this setting, the agent must integrate fragmented information and coordinate actions across time, services, and devices. The expanded context scope also enables evaluation of proactive assistance (Tang et al., 2026; Yang et al., 2026), requiring the agent to anticipate user needs and provide timely recommendations from context rather than merely react to explicit requests.

Constructing such environments at scale is challenging: it requires modeling extended time horizons, numerous services, and multiple devices while preserving realism and cross-component consistency.

Table 1: Comparison of representative digital-agent benchmarks and Claw-Anything across event streams, device interfaces, service coverage, and proactivity. "Event Stream" denotes records of user activity in the digital environment; "Device Interfaces" denotes the interaction surfaces in each task; "Services" gives the average and maximum number of services used per task; and "Context Length" is the length of textualized static states and dynamic event streams. "Proactive" denotes assistance without an explicit task-specific goal; in Claw-Anything, this is recommendation/discovery after a fixed heartbeat, not intervention-timing evaluation.

Benchmark	Event Stream	Device Interfaces	# Services (avg. / max.)	Proactive	Context Length (words)	# Instances (Eval)	# Instances (Train)
ClawBench (Zhang et al., 2026)	✗	CLI	1.6 (5)	✗	2.2k	313	0
WildClawBench (Ding et al., 2026)	✗	CLI	0.5 (3)	✗	2.6k	60	0
PinchBench (kilo.ai, 2026)	✗	CLI	0.1 (3)	✗	1.7k	53	0
ClawMark (Evolvent AI, 2026)	✗	CLI	3.9 (5)	✗	2.0k	100	0
QwenClawBench (Qwen Team, 2026)	✗	CLI	0.3 (6)	✗	12.1k	100	0
Claw-Eval (Ye et al., 2026)	✗	CLI	1.3 (6)	✗	5.3k	300	0
Claw-Anything (ours)	✓	**CLI + GUI**	**10.1 (18)**	✓	**191.7k**	**200**	**2000**

We therefore develop an automated pipeline that jointly synthesizes digital worlds and tasks. Starting from a minimal persona seed, an LLM-based simulator incrementally expands the user's digital world through multi-round event injection. At each step, it samples everyday events from a seed pool and updates both persistent world state and dynamic service traces, including sources such as email, calendars, and social platforms. Over time, the event history accumulates, the persona becomes more fully specified, and the environment acquires richer states and realistic noise, including irrelevant or contradictory events. Given the resulting digital world, the next event is instantiated as a persona-grounded task with an executable verifier, casting evaluation as completing the next step in an evolving digital life. Using this pipeline, we construct 200 human-verified evaluation tasks and 2,000 training environments, enabling Claw-Anything to function both as a benchmark and as scalable data infrastructure.

Experiments reveal a substantial gap between current capabilities and the demands of broad-access personal assistance. On Claw-Anything, GPT-5.5 achieves only 34.5% Pass@1. Several models that perform strongly on existing benchmarks also fail on ours, suggesting that Claw-Anything exposes failure modes underrepresented in prior evaluations and that current models remain unreliable even when given broader access to the user's digital world. As a complementary test of pipeline utility, fine-tuning Qwen3.5-27B on 1,500 successful trajectories improves Pass@1 from 9.8% to 33.5%, a gain of 23.7 percentage points. A separate evaluation shows modest gains on Claw-Eval, another personal-assistant benchmark (Appendix E.1).

Our primary contribution is **Claw-Anything**, a benchmark of 200 human-verified interactive task environments that jointly expands access across time, services, and devices, with heartbeat-conditioned proactivity as a distinct evaluation axis. We develop an automated construction pipeline; the benchmark-validity and human-evaluation studies are detailed in Appendix B. Our evaluation exposes substantial capability and reliability gaps, with GPT-5.5 achieving 34.5% Pass@1. As a secondary contribution, we demonstrate that the same pipeline supplies useful training data through 2,000 training environments and a fine-tuning study with both in-benchmark and external evaluation.

2 Related Work

Benchmarks for Personal Assistants. As claw-style agents have rapidly gained momentum, a growing family of benchmarks has emerged to measure their capabilities. ClawBench (Zhang et al., 2026) broadens coverage across a large set of standardized digital tasks, WildClawBench (Ding et al., 2026) moves evaluation into more realistic open environments, PinchBench (kilo.ai, 2026) centers on practical personal-productivity scenarios, ClawMark (Evolvent AI, 2026) studies longer-horizon professional workflows, QwenClawBench (Qwen Team, 2026) emphasizes execution in realistic user-distributed CLI tasks, and Claw-Eval (Ye et al., 2026) advances evaluation methodology through rubric-based assessment for open-ended trajectories. Collectively, these benchmarks have advanced the study of planning, tool use, and grounded interaction for digital agents. Yet they largely cast the agent as a solver of localized tasks rather than an always-on assistant embedded in a user's broader digital world. Most remain confined to isolated, short-horizon, and relatively clean settings,

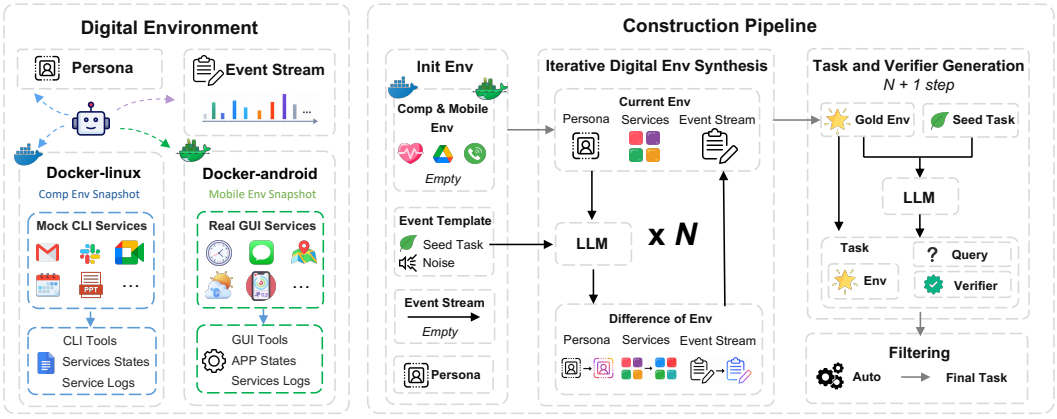


Figure 3: Claw-Anything environment and automated data pipeline. **Left:** The environment comprises connected devices with system event streams and multiple services with persistent states and service-specific histories. **Right:** From a persona-grounded initial state, the pipeline iteratively samples task or noise templates and uses an LLM-based simulator to adapt events and update the world state. A final simulation generates the task query, reference solution, and grader; automatic filtering then yields task instances, with optional human verification for benchmark cases.

offering limited coverage of reasoning over noisy event streams, coordinating across devices and backend systems, or acting from accumulated personal context. Claw-Anything addresses this gap by evaluating agents over a broader slice of the user's digital world, including long-horizon activity streams, interconnected systems, heterogeneous devices, and proactive opportunities.

Scalable Agent Training Environments. In software-agent research, prior work on scalable environments has mainly followed two directions: code-centric scenarios (Jimenez et al., 2024; Zhou et al., 2026), such as SWE-smith (Yang et al., 2025a) and SWE-Gym (Pan et al., 2024); and terminal-centric scenarios (Teams, 2026), such as CLI-Gym (Lin et al., 2026) and TermiGen (Zhu et al., 2026b). Together, these works suggest that scalable environments matter not only for evaluation but also for broader agent development. This paradigm remains underexplored in personal-assistant settings, where verifiable environments often depend on manual construction, limiting both realism and scalability. We address this gap by combining a realistic setting across services, time, and devices with a multi-round automated pipeline that jointly simulates personas, histories, and cross-service states. The resulting framework enables controlled variation in task difficulty and environmental complexity, providing a practical basis for scalable evaluation and development of personal-assistant agents.

Proactive and Personalized Assistance. Proactive Agent (Lu et al., 2025) evaluates whether proposed assistance is accepted or rejected, while ContextAgent (Yang et al., 2026) uses sensory and persona context to decide whether assistance is needed and then invokes tools. FingerTip 20K (Yang et al., 2025b) studies proactive intent suggestion and personalized execution using real, long-term Android interactions. These settings are complementary to ours: Claw-Anything fixes the check time and evaluates what to recommend from noisy, long-horizon, multi-service evidence. Its proactive subset is CLI-only, whereas the broader benchmark additionally includes CLI+GUI tasks. We do not evaluate intervention timing or false alarms on no-op checks; Appendix A.3 details this boundary.

3 Methodology

Claw-Anything is a benchmark for evaluating whether an agent can complete both reactive and proactive personal-assistant tasks when endowed with broad access to a user's digital world. Each task is grounded in a coherent persona and embedded in an environment spanning three contextual dimensions: long-horizon history, diverse backend services, and coordinated interactions across multiple devices with heterogeneous interfaces (e.g., GUI and CLI). Within this setting, the agent must isolate task-relevant signals from substantial background noise and execute required actions.

Algorithm 1 Automated task generation pipeline.

Input: seed persona $\mathcal{P}_0$; task-seed pool $\mathcal{S}$; noise-event pool $\mathcal{N}$; rollout horizon R; snapshot rounds $\mathcal{I}_{\text{task}}$.
Initialize: fixture state $\mathcal{F} \leftarrow \emptyset$, event log $\mathcal{L} \leftarrow \emptyset$, persona state $\mathcal{P} \leftarrow \mathcal{P}_0$, and task set $\mathcal{T} \leftarrow \emptyset$.
for $r = 1, \ldots, R$ **do**

1. $e \leftarrow \text{Sample}(\mathcal{S}, \mathcal{N}, \text{noise_ratio})$ — Sample a task or noise event
2. $\tilde{e} \leftarrow \text{AdaptToEnv}(e, \mathcal{P}, \mathcal{F}, \mathcal{L})$ — Ground it in the current environment
3. Use an LLM to generate updates $\Delta\mathcal{F}, \Delta\mathcal{L}, \Delta\mathcal{P}$ from $\tilde{e}$
4. Update the environment: $\mathcal{F} \leftarrow \mathcal{F} \cup \Delta\mathcal{F}$, $\mathcal{L} \leftarrow \mathcal{L} \cup \Delta\mathcal{L}$, $\mathcal{P} \leftarrow \mathcal{P} \cup \Delta\mathcal{P}$
5. **if** $r \in \mathcal{I}_{\text{task}}$ **then**
   - $X_r \leftarrow \text{Snapshot}(\mathcal{F}, \mathcal{L}, \mathcal{P}, r)$ — Snapshot the current environment
   - $Q_r \leftarrow \text{GenTaskQuery}(X_r)$ — Generate a task query
   - $V_r, A_{\text{ref},r} \leftarrow \text{GenVerifier}(Q_r, X_r)$ — Generate the verifier and reference answer
   - $\tau_r \leftarrow \text{AutoFilter}(X_r, Q_r, V_r, A_{\text{ref},r})$ — Filter the task instance
   - **if** $\tau_r \neq \varnothing$ **then** $\mathcal{T} \leftarrow \mathcal{T} \cup \{\tau_r\}$

Output: Task set $\mathcal{T}$. — May undergo human verification for benchmark cases.

3.1 Task Formulation

As illustrated in the left panel of Figure 3, Claw-Anything places the agent in a digital environment with broad access to the user's digital world, formulates reactive and proactive personal-assistant tasks in that environment, and evaluates completion with an executable verifier over the interaction trace and resulting task state.

Context-rich digital environment. We instantiate each task in a context-rich, realistic, and noisy digital environment. Formally, each environment is defined as $\mathcal{E} = (\mathcal{P}, \mathcal{D}, \mathcal{F}, \mathcal{L})$, where $\mathcal{P}$ denotes a user persona specifying the user's profile and preferences; $\mathcal{D}$ denotes a set of devices with heterogeneous interfaces, including CLI-based computers and GUI-based mobile phones; $\mathcal{F}$ denotes a fixture bank of persistent states across backend services spanning lifestyle, work, and related domains; and $\mathcal{L}$ denotes a long-horizon activity stream covering approximately three months of system-level and service-specific logs. We further populate these environments with irrelevant events, services, and state to better approximate real-world settings, requiring agents to reason over large-scale context and complete tasks in a closed loop.

Queries across time, services, and devices. Each query is written in naturalistic and sometimes underspecified language. Solving it requires identifying task-relevant signals across event streams, services, and devices, including CLI-based Linux and GUI-based Android environments. Reactive tasks specify a user goal; proactive tasks instead provide a heartbeat or broad check at a fixed time, leaving the agent to decide what to raise. All 15 proactive tasks are CLI-only and contain a discoverable need. We therefore evaluate *heartbeat-conditioned recommendation/discovery*, not when to interrupt or whether to remain silent. Each evaluation is a bounded episode in a history-rich environment, rather than a months-long deployment. For example, a discovery task may require linking a drafted, repeatedly edited, and discarded message to an unresolved matter in contacts, meetings, or notes, then recommending a follow-up without sending the message (Appendix A.3).

Outcome-oriented evaluation for multi-path tasks. Our evaluation builds on the rubric-based framework of Claw-Eval (Ye et al., 2026), combining rule-based checks with LLM judgments to produce both a soft score and a binary pass/fail label. Because many tasks admit multiple valid solution paths, we assign greater weight to the final outcome and correspondingly less to intermediate actions. This modification retains the strengths of rubric-based evaluation while better reflecting the open-ended nature of personal-assistant tasks.

3.2 Construction Pipeline

Manually constructing a context-rich digital world together with its associated tasks is prohibitively expensive and difficult to scale. We therefore generate both evaluation and training data with an

Category	Metric	Claw-Eval	Claw-Anything	
			Eval	Train
Size	# Instance	300	200	2000
Context Text	# Word of fixture	5.3k	108.0k	97.3k
	# Word of log	0	83.7k	65.7k
Services	# Task-involved	1.3	10.1	9.2
	# Env-support	19	35	35
Devices	Support Type	CLI	CLI + GUI	CLI + GUI



Figure 4: Benchmark statistics of Claw-Anything. **Left:** Comparison with Claw-Eval in size, context length, services per task, and supported devices. **Right:** Category distribution of evaluation instances.

automatic pipeline, illustrated in Algorithm 1 and Figure 3, that incrementally builds an evolving user environment, extracts tasks from intermediate states, and removes low-quality instances.

Stage I: Iterative digital environment synthesis. We construct an evolving digital environment through an iterative generation loop. Human-abstracted seed mechanisms, such as time conflicts, contradictory information, budget constraints, and unfinished intent, are adapted to the current persona and world state. Each event can update several services; a shared fixture registry deduplicates records, and rule-based action templates keep the resulting logs consistent with fixture facts. Genuine events also enrich the persona's history, whereas noise events add distractions without advancing it. Thus tasks are grounded in accumulated world state rather than generated independently from static prompts (Appendix A).

Stage II: Task and verifier generation. At designated rounds, the pipeline captures a temporary snapshot, adapts a new task seed, and adds supporting fixtures. From this shared state, it jointly produces a user query, scoring specification, executable verifier, and reference solution. Failed generation attempts roll back so that they do not contaminate later tasks. Each task is thereby grounded in a specific temporal slice of the evolving digital world.

Stage III: Automatic filtering. Because the pipeline depends on LLM generation, automated quality control is necessary. We therefore combine rule-based checks with LLM-based filtering to remove invalid instances before human review. Rule-based checks target surface inconsistencies, such as references to nonexistent tools or services. LLM-based filtering then evaluates higher-level validity by using the environment state and reference solution to determine whether a task is solvable and whether its verifier is logically consistent with the specification.

Stage IV: Human verification with execution support. Finally, we perform human verification supplemented by execution-based validation. A strong agent is given the reference solution and asked to execute the task in the environment with the verifier. Successful execution indicates that the task admits at least one valid solution consistent with the intended logic, enabling human reviewers to focus on assessing the consistency among the query, environment, and verifier. Instances that fail execution are escalated for manual review to determine whether they should be revised or discarded.

3.3 CLAW-ANYTHING

Benchmark Statistics. As shown in Figure 4, the full pipeline, including fourth-stage human verification, yields 200 tasks across 9 categories: 150 CLI-only tasks (135 reactive and 15 proactive) and 50 CLI+GUI tasks. Proactivity is a standalone category. Each instance is a complete interactive environment, averaging 191.7k words of fixtures and logs and 10.1 involved services, rather than a static question. The task count is within the range of the executable benchmarks in Table 1; our emphasis is on per-instance contextual scope, not dataset size alone.

Trajectory Collection with Claw-Anything. For training trajectory collection, we execute the first three stages of the automated pipeline to generate 2,000 task environments. To prevent contamination of the evaluation set, these environments are drawn from a persona pool fully disjoint from the evaluation personas. We then collect 1,500 successful trajectories from these environments for the subsequent post-training of Qwen3.5-27B.

Table 2: Main results on the Claw-Anything benchmark. We evaluate both state-of-the-art open- and closed-source models under a unified OpenHarness framework for fair comparison. The best result in each column is shown in bold.

Model	# Params	Score	Pass@1	Pass@3	Pass^3	# Tokens (I / O)
Open-Source						
Qwen3.5-27B (Qwen, 2026a)	27B	0.50	9.8	19.0	2.0	83.8M / 0.9M
MiniMax-M2.7 (MiniMax-AI, 2026)	229B	0.52	13.5	28.5	3.5	79.0M / 1.1M
Qwen3.6-27B (Qwen, 2026b)	27B	0.58	22.5	42.0	6.0	99.4M / 2.0M
Kimi-K2.6 (Moonshot AI, 2026)	1.1T	0.57	22.8	44.0	6.5	178.1M / 2.3M
GLM-5.1 (ZAI, 2026)	754B	0.59	31.7	47.0	**17.0**	125.0M / 2.2M
Claw-Anything-Qwen3.5-27B (ours)	27B	**0.61**	**33.5**	**52.0**	15.5	117.8M / 1.1M
Gain over Qwen3.5-27B	-	*+0.11*	*+23.7*	*+33.0*	*+13.5*	-
Closed-Source						
Claude Sonnet 4.5 (Anthropic, 2025)	-	0.59	28.0	45.0	12.0	149.0M / 1.5M
Claude Opus 4.7 (Anthropic, 2026)	-	0.62	31.8	48.0	13.5	123.5M / 1.5M
GPT-5.5 (OpenAI, 2026)	-	**0.65**	**34.5**	**53.5**	**20.0**	77.7M / 0.9M

4 EXPERIMENT

4.1 MAIN RESULTS OF CLAW-ANYTHING

Frontier baselines. We benchmark a broad set of frontier LLMs, covering open-source families such as Qwen series (Qwen, 2026a;b), MiniMax 2.7 (MiniMax-AI, 2026), GLM 5.1 (ZAI, 2026), and Kimi 2.6 (Moonshot AI, 2026), as well as closed-source models including Claude Opus 4.7 (Anthropic, 2026) and GPT-5.5 (OpenAI, 2026). All models are evaluated under OpenHarness (HKUDS Teams, 2026b), a widely adopted ultra-lightweight agent scaffold for personal agents implemented in pure Python. The automatic graders combine executable checks with Claude Sonnet 4.5 (Anthropic, 2025) as the LLM judge. Following Claw-Eval, we report Pass@1, Pass@3, and Pass^3 as the primary metrics, where Pass^3 requires success in all three independent runs. We further use continuous execution score and token consumption as complementary indicators of solution quality. Table 2 summarizes the results. Even the strongest closed-source model reaches only 20.0% on Pass^3, which suggests that bringing the agent's perceptual scope closer to that of the user materially increases benchmark difficulty, because success now depends on both accurate understanding of the user's digital environment and correct action grounded in that context.

Improvement from collected training trajectories. As a secondary study of pipeline utility, we construct 2,000 training environments and use GLM-5.1 to collect trajectories, retaining 1,500 successful runs for full-parameter fine-tuning of Qwen3.5-27B over 10 epochs. Evaluation environments are never used for training, and the training and evaluation persona pools are disjoint. Pass@1 increases from 9.8% to 33.5%, a gain of 23.7 percentage points, exceeding the other open-weight baselines in Table 2. Figure 6 shows improvement with trajectory count. The fine-tuned model also yields modest improvements on Claw-Eval, another personal-assistant benchmark, with Pass@1 rising from 69.7% to 70.7%; Appendix E.1 reports the full protocol and all three metrics.

Benchmark validity. The complete protocols and results for the benchmark-validity and human-evaluation studies are provided in Appendix B.

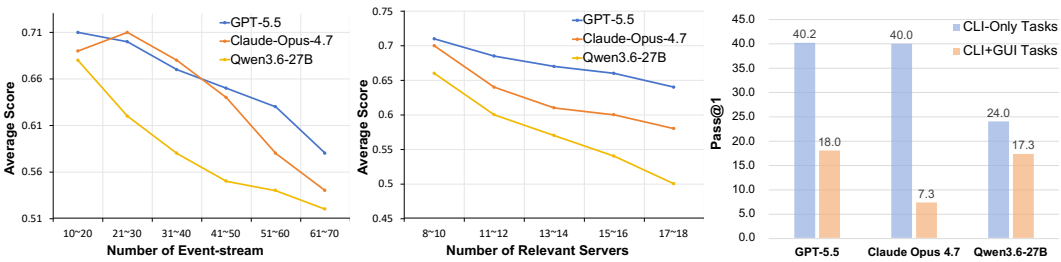
4.2 ABLATION STUDY

We ablate contextual scale (Section 4.2.1), the data pipeline (Section 4.2.2), and evaluation settings (Section 4.2.3); additional details appear in the appendix.

4.2.1 SCALING CONTEXT

This section ablates whether expanding the agent's operational scope unlocks previously infeasible tasks, and whether larger context constitutes a fundamental bottleneck for current agents.

Long-horizon event streams. We remove event streams from history-dependent tasks while keeping their other components fixed. As shown in Table 3, success rates drop substantially, supporting the



(a) Long-horizon event streams. (b) Multiple backend services. (c) Cross-device (CLI + GUI).

Figure 5: Ablation of contextual scale, showing the effects of event-stream volume and the number of services on average score, as well as the effect of GUI access on Pass@1.

Table 3: Effects of access to event streams, cross-service environments, and cross-device interaction. The proactive/reactive comparison uses 15/135 CLI-only tasks with Qwen3.6-27B; all results are Pass@1.

Factor	w/	w/o
Event Stream	21.0	0.0
Cross-services	24.0	0.0
Cross-devices	16.0	2.0
Task Type	**Reactive**	**Proactive**
Pass@1	25.9	6.7

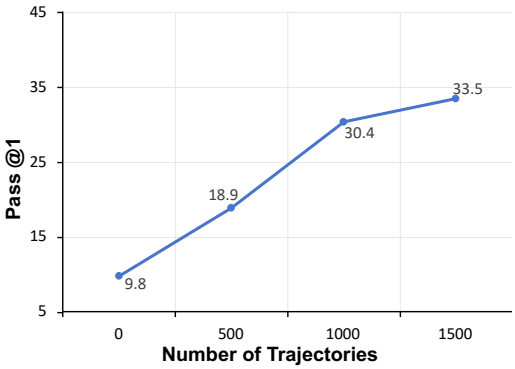


Figure 6: Trajectory scaling.

need for historical evidence beyond static fixtures. Separately, Figure 5 groups existing tasks by history length: mean task scores are lower in longer-history groups. This association motivates better long-context reasoning, but does not isolate history length from other task differences.

Cross-backend services. Removing all backend tools reduces success to zero (Table 3), confirming that the tasks require backend access. This is an access-necessity test, not a controlled single- versus cross-service comparison. Figure 5 separately shows an observational decline in mean score as the number of services increases.

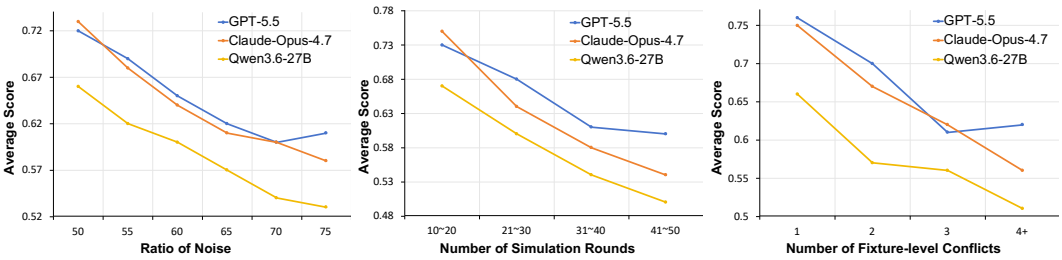
CLI–GUI collaboration. Removing GUI access makes tasks requiring CLI–GUI collaboration nearly unsolvable, whereas joint access restores nonzero performance (Table 3). Even with both interfaces, these tasks remain substantially harder than CLI-only tasks (Figure 5). Thus, joint access expands the solvable task set, but cross-interface coordination remains challenging.

4.2.2 Data pipeline

Noise injection ratio. Figure 7 shows lower mean scores among tasks with higher recorded noise ratios, consistent with harder signal recovery. This grouping is observational, not a controlled noise intervention.

Persona richness. Mean scores are lower for tasks generated with more simulation rounds (Figure 7). Because rounds add both persona history and environmental state, this curve does not isolate persona richness.

Fixture-level conflicts. Mean scores are lower in groups with more cross-service fixture conflicts (Figure 7). This characterizes a difficult benchmark subset rather than a causal effect of conflict count.



(a) Ratio of noise rounds. (b) Simulation Rounds. (c) Fixture-level conflicts.

Figure 7: Ablation of the automatic data-generation pipeline, showing the effects of the noise-round ratio, the number of simulation rounds, and the number of fixture-level conflicts.

Table 4: Skill-loading ablation. We compare full and lazy loading across models. Under lazy loading, the agent must select tools autonomously, making the setting much more challenging. All results are reported in Pass@1.

Model	Full	Lazy
Minimax 2.7	22.7	10.0
Qwen3.6-27B	24.7	23.7
GLM-5	29.3	14.0
Claude Sonnet 4.6	43.0	26.7

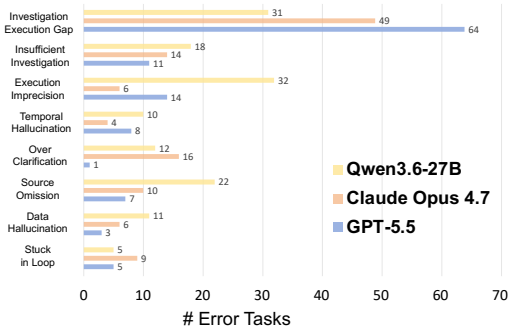


Figure 8: Failure-mode distribution.

4.2.3 Evaluation Setting

Proactivity. On CLI-only tasks, Qwen3.6-27B obtains 25.9% Pass@1 on 135 reactive tasks versus 6.7% on 15 proactive tasks (Table 3). This unpaired comparison evaluates recommendation after a fixed heartbeat, not intervention timing or false-alarm control.

Skill loading strategy. Full loading supplies all tool specifications, whereas lazy loading supplies brief descriptions and requires autonomous skill selection. Lazy loading substantially reduces Pass@1 for most models (Table 4); Qwen3.6-27B is comparatively robust, suggesting better recovery from incomplete skill context.

4.3 Failure Mode Analysis

Figure 8 identifies the investigation–execution gap as the dominant failure mode: agents find relevant context but fail to act successfully. Qwen3.6-27B more often exhibits execution imprecision and source omission, whereas Claude Opus 4.7 more often over-clarifies or loops; hallucination-related errors are comparatively rare.

5 Conclusion

We introduced Claw-Anything, a benchmark for evaluating personal-assistant agents under a substantially broader operational scope. By combining long-horizon event streams, diverse backend services, multi-device interaction, and proactive tasks, it captures core challenges that are largely absent from existing evaluations. Our results reveal a pronounced gap between current frontier models and the requirements of real-world assistance: lower performance is associated with greater contextual breadth, and proactive settings remain especially difficult. The accompanying automated data-generation pipeline further supports scalable environment construction and provides a practical foundation for future research on personal-assistant agents.

AI USE STATEMENT

Generative AI systems were used in both the research pipeline and manuscript preparation. GLM-5.1 was used to generate environments, tasks, grader code, and training trajectories. Claude Sonnet 4.5 was used as the LLM judge in the evaluation pipeline, together with deterministic checks. Codex was used solely for language polishing and grammar correction. We reviewed all AI-generated codes and AI-assisted text and verified their accuracy and consistency with the underlying research.

REPRODUCIBILITY STATEMENT

We release the complete benchmark and supporting code in an anonymous repository, including the environment- and task-generation pipeline, seed and noise templates, executable graders, evaluation scripts, Docker and Android setup instructions, raw model outputs and scores, and human-audit annotations. Appendix A documents the construction pipeline and validation procedure; Appendix C.1 specifies the agent scaffold, decoding settings, execution budgets, scoring protocol, and data splits; Appendix D specifies the ablation settings; Appendix E reports trajectory collection and fine-tuning hyperparameters; and Appendix E.1 gives the independent Claw-Eval protocol. Appendix F describes the released artifact structure and links each reported result to its supporting materials. Exact reruns involving proprietary models require access to the corresponding model APIs and may remain subject to provider-side nondeterminism.

REFERENCES

Pranjal Aggarwal, Graham Neubig, and Sean Welleck. Gym-anything: Turn any software into an agent environment. *arXiv preprint arXiv:2604.06126*, 2026.

Anthropic. Introducing claude sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5, September 2025.

Anthropic. Introducing claude opus 4.7. https://www.anthropic.com/news/claude-opus-4-7, 2026. Anthropic news release, accessed: 2026-04-28.

Shuangrui Ding, Xuanlang Dai, Long Xing, Shengyuan Ding, Ziyu Liu, Jingyi Yang, Penghui Yang, Zhixiong Zhang, Xilin Wei, Yubo Ma, Haodong Duan, Jing Shao, Jiaqi Wang, Dahua Lin, Kai Chen, and Yuhang Zang. Wildclawbench. https://github.com/InternLM/WildClawBench, 2026. GitHub repository.

Evolvent AI. ClawMark: A living-world benchmark for multi-day, multimodal coworker agents. https://github.com/evolvent-ai/ClawMark, 2026. GitHub repository.

Google Workspace. One cli for all of google workspace — built for humans and ai agents. https://github.com/googleworkspace/cli, 2026. GitHub repository, accessed 2026-04-24.

HKUDS Team. CLI-Anything: Making all software agent-native. https://github.com/HKUDS/CLI-Anything, 2026. GitHub repository, accessed 2026-04-24.

HKUDS Teams. Nanobot: The ultra-lightweight personal ai agent. https://github.com/HKUDS/nanobot, 2026a. GitHub repository.

HKUDS Teams. Openharness: Open agent harness with a built-in personal agent–ohmo! https://github.com/HKUDS/OpenHarness, 2026b. GitHub repository.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

kilo.ai. PinchBench: Real-world benchmarks for ai coding agents. https://github.com/pinchbench/skill, 2026. GitHub repository.

Larksuite. lark-cli: The official lark/feishu cli for humans and ai agents. https://github.com/larksuite/cli, 2026. GitHub repository, accessed 2026-04-24.

Yusong Lin, Haiyang Wang, Shuzhe Wu, Lue Fan, Feiyang Pan, Sanyuan Zhao, and Dandan Tu. Cli-gym: Scalable cli task generation via agentic environment inversion. *arXiv preprint arXiv:2602.10999*, 2026.

Yaxi Lu, Shenzhi Yang, Cheng Qian, Guirong Chen, Qinyu Luo, Yesai Wu, Huadong Wang, Xin Cong, Zhong Zhang, Yankai Lin, Weiwen Liu, Yasheng Wang, Zhiyuan Liu, Fangming Liu, and Maosong Sun. Proactive agent: Shifting LLM agents from reactive responses to active assistance. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=sRIU6k2TcU.

MiniMax-AI. Minimax-m2.7. https://huggingface.co/MiniMaxAI/MiniMax-M2.7, 2026. Hugging Face model repository, version 2.7, accessed: 2026-04-28.

Moonshot AI. Kimi k2.6: Advancing open-source coding. https://www.kimi.com/blog/kimi-k2-6, 2026.

NanoClaw Teams. A lightweight alternative to openclaw that runs in containers for security. https://github.com/qwibitai/nanoclaw, 2026. GitHub repository.

NousResearch. Hermes agent: The agent that grows with you. https://github.com/nousresearch/hermes-agent, 2026.

OpenAI. Introducing gpt-5.5. https://openai.com/zh-Hans-CN/index/introducing-gpt-5-5/, 2026. OpenAI blog post, accessed: 2026-04-28.

OpenClaw. Openclaw: Open-source personal ai assistant. https://github.com/openclaw/openclaw, 2026. GitHub repository.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139*, 2024.

Qwen. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, February 2026a.

Qwen. Qwen3.6-27B: Flagship-level coding in a 27B dense model. https://qwen.ai/blog?id=qwen3.6-27b, April 2026b.

Alibaba Group Qwen Team. QwenClawBench: Real-user-distribution benchmark for openclaw agents. https://github.com/SKYLENAGE-AI/QwenClawBench, April 2026. GitHub repository.

Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2446–2454, 2020.

Yuanbo Tang, Huaze Tang, Tingyu Cao, Lam Nguyen, Anping Zhang, Xinwen Cao, Chunkang Liu, Wenbo Ding, and Yang Li. Proagentbench: Evaluating llm agents for proactive assistance with real-world data. *arXiv preprint arXiv:2602.04482*, 2026.

Terminal-Bench Teams. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *The Fourteenth International Conference on Learning Representations*, 2026.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, et al. Mobile-Agent-v3.5: Multi-platform fundamental GUI agents. *arXiv preprint arXiv:2602.16855*, 2026.

Bufang Yang, Lilin Xu, Liekang Zeng, Kaiwei Liu, Siyang Jiang, Wenrui Lu, Hongkai Chen, Xiaofan Jiang, Guoliang Xing, and Zhenyu Yan. ContextAgent: Context-aware proactive LLM agents with open-world sensory perceptions. *Advances in Neural Information Processing Systems*, 38: 167509–167543, 2026.

John Yang, Kilian Lieret, Carlos E Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track (NeurIPS)*, 2025a.

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. *arXiv preprint arXiv:2507.21071*, 2025b.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, et al. Claw-eval: Toward trustworthy evaluation of autonomous agents. *arXiv preprint arXiv:2604.06132*, 2026.

ZAI. Glm-5.1: Towards long-horizon tasks. `https://z.ai/blog/glm-5.1`, 2026.

Yuxuan Zhang, Yubo Wang, Yipeng Zhu, Penghui Du, Junwen Miao, Xuan Lu, Wendong Xu, Yunzhuo Hao, Songcheng Cai, Xiaochen Wang, Huaisong Zhang, Xian Wu, Yi Lu, Minyi Lei, Kai Zou, Huifeng Yin, Ping Nie, Liang Chen, Dongfu Jiang, Wenhu Chen, and Kelsey R. Allen. Clawbench: Can ai agents complete everyday online tasks? *arXiv preprint arXiv:2604.08523*, 2026.

Qixing Zhou, JiaCheng Zhang, Haiyang Wang, Rui Hao, Jiahe Wang, Minghao Han, Yuxue Yang, Shuzhe Wu, Feiyang Pan, Lue Fan, Dandan Tu, and Zhaoxiang Zhang. Featurebench: Benchmarking agentic coding for complex feature development. In *The Fourteenth International Conference on Learning Representations*, 2026.

Jared Zhu, Minhao Hu, and Junde Wu. Swe context bench: A benchmark for context learning in coding. *arXiv preprint arXiv:2602.08316*, 2026a.

Kaijie Zhu, Yuzhou Nie, Yijiang Li, Yiming Huang, Jialian Wu, Jiang Liu, Ximeng Sun, Zhenfei Yin, Lun Wang, Zicheng Liu, et al. Termigen: High-fidelity environment and robust trajectory synthesis for terminal agents. *arXiv preprint arXiv:2602.07274*, 2026b.

A Details of Task Generation Pipeline

Model roles and consistency controls. GLM-5.1 generates the environments, tasks, grader code, and training rollouts; Qwen3.5-27B is the model fine-tuned on retained trajectories. Evaluation combines deterministic checks with Claude Sonnet 4.5 (Anthropic, 2025) as the LLM judge, from a different model family than the generator. This separation reduces direct generator–judge overlap, but does not by itself establish unbiased evaluation, since tasks and grader specifications share a generator and the judge may have model-family preferences. We therefore separately assess external task quality and human–grader alignment in Appendix B.

Table 5: Construction choices for an evolving executable world. The contrast is with independently generated static task instances, not a claim that every prior generator lacks these components.

Design axis	Static-instance contrast	Claw-Anything implementation
Temporal state	Context constructed for one task	Persona, fixtures, and logs accumulate across event rounds
Cross-service consistency	Records generated in isolation	Multi-service event updates, shared record registry, fixture-grounded logs
Task isolation	No shared evolving state to preserve	Temporary snapshots with rollback on failed generation
Executability	Textual plausibility alone is insufficient	Schema/ID/tool checks, grader compilation, reference execution, human review
Interaction surfaces	Textual state alone	Executable CLI services and Android GUI state injection

A.1 Persona Creation and Enrichment

Personas guide the construction of personalized and diverse Claw-Anything environments. In Stage I of the construction pipeline, we first generate a coarse persona and then iteratively enrich it while sampling task and event templates that provide the meta-level structure for user-event simulation.

The initial persona specifies basic user attributes, such as role and traits, as illustrated in Figure 9. It is intentionally concise and can be generated with a modern LLM.

We enrich each initial persona with concrete preferences, activities, and evolving context by simulating a sequence of user events. Given a pool of *meta* tasks and events that encode conflicting or distracting factors, we repeatedly sample an item and prompt an LLM with the current persona and sampled item to instantiate a persona-specific event, including its antecedents, consequences, and accompanying noise. We inject the resulting data into service databases and logs and update the persona description and activity history. After several rounds, the persona evolves beyond its initialization and exhibits more specific preferences and behavioral patterns. Figure 10 illustrates this enrichment process.

The enrichment pool contains two types of meta items: *seed tasks*, which encode conflict mechanisms, and *noise events*, which encode distractions. We describe their instantiation in turn.

Seed Tasks

Seed tasks encode conflict mechanisms commonly encountered in everyday settings, such as scheduling conflicts, contradictory information, and financial constraints. Figure 11 shows an example based on conflicting information. Resolving such tasks requires agents to identify and reconcile the relevant constraints.

Given a seed sampled from the seed library, the instantiation goes through the following five stages in order. One such pass is run in every enrichment round, and its outputs are added to both the environment and the persona.

Stage 1: Persona-Specific Adaptation. We provide an LLM with the seed's abstract description—including its services, conflict pattern, and resource constraints—and the current persona. The LLM produces an *adapted task* by rewriting abstract entities, such as names, organizations, time anchors, and target applications, to fit the persona. The adapted task also records the involved services, which determine which applications subsequent stages update. Figure 12 shows the adaptation prompt.

Stage 2: Fixture Generation. For each service involved in the adapted task, the LLM is called again to generate several new fixture records as the task-related data of that service. Before being written to disk, the generated records are checked against a shared fixture registry that removes duplicates by record id, so that no record is repeated within or across rounds. The generation prompt is shown in Figure 13.

Stage 3: Activity Log Generation. Once the fixtures have been written to disk, a rule-based activity log engine generates a sequence of user actions for each newly added record. The rules map service type and a few key fields of the record to predefined action templates: for example, a Gmail message is first classified as "send" or "receive" by comparing its sender field with the persona's own email address, and is then expanded into an action chain such as (open, read, reply) or (compose, send). This stage is fully rule-based, so that the action order in the activity log strictly matches the facts already in the fixtures.

Stage 4: Work Journal Synthesis. Based on the fixtures and the activity log, the LLM is called once more to narrate the round's events as a first-person work journal entry in Markdown, which is added to the persona's log directory. The work journal is not used for evaluation. It serves only as context in later rounds, so that newly generated experiences stay consistent with what the persona has already gone through.

Stage 5: Persona Update. Finally, the round's experience is summarized into a short entry and appended to the persona's history. If it corresponds to an unresolved matter, such as an unanswered critical email or unconfirmed meeting, we also add an entry to the active-issue list. All other persona fields remain unchanged. This design preserves the persona's core identity while allowing its history and active context to evolve over rounds.

NOISE EVENTS

Noise events add distractions that are irrelevant to the target task. We curate a library of routine activity patterns, such as scanning the inbox, drafting and discarding an email, jotting down and deleting a note, browsing a calendar or RSS feed, or checking an inventory dashboard. These patterns are instantiated as dated sessions across the persona's working days. Some are purely *ephemeral* and appear only in activity logs, whereas others are *trace-leaving* and also deposit residual fixture records, such as deleted notes, discarded drafts, or canceled to-dos, in the corresponding services. This increases environmental complexity and realism without introducing task-relevant signals.

Unlike seed tasks, noise events are independent of the target task and do not advance the persona's history. Their instantiation therefore skips the adaptation, work-journal, and persona-update stages, and further merges fixture generation with activity-log expansion into a single LLM call. The remaining procedure reduces to three stages.

Stage 1: Seed Sampling. The workdays covered by this round are first listed from the persona's time window. For each workday, a small set of noise patterns is drawn from the library with sampling biased toward service diversity, so that one day's routines spread across different apps rather than piling onto one.

Stage 2: Session Synthesis. A single LLM call, conditioned on the persona, the sampled patterns and their assigned days, produces all routine sessions of the batch. Each session yields a dated action chain that becomes activity-log entries, together with an optional residual fixture record when its seed is trace-leaving.

Stage 3: Environment Injection. The action chains are appended to the per-service activity logs and the residual fixture records are added to the per-service fixtures. The persona itself is left untouched, so that its identity and history reflect only the genuine events introduced by seed tasks.

A.2 TASK QUERY AND VERIFIER GENERATION

At Stage II of the construction pipeline, we generate evaluation tasks and their verifiers from the environment and persona produced in Stage I. The pipeline reuses seed sampling, persona adaptation, and fixture generation from Stage I, then adds scoring-specification and grader generation. Each task is built from a temporary environment snapshot, so a failure at any step rolls the environment back and does not contaminate subsequently generated tasks.

Stage 1: Seed Sampling and Adaptation. A seed is drawn at random from the seed library. Seeds come in two *interaction modes*: an S-type seed has a single fixed surface content, while an M-type seed lists several candidate surface contents sharing the same underlying conflict pattern. For the latter, one candidate is picked here as the surface direction of this task, so the same seed can yield variants with different surface topics across draws — the main source of task diversity at Stage II. The selected seed is then adapted to the current persona by an LLM. This adaptation reuses the same machinery as in Stage I, but the prompt is now switched to evaluation mode, so the resulting task carries a clear, evaluable goal rather than continuing the persona's historical narrative.

Stage 2: Fixture Generation. For each service involved in the adapted task, we reuse the Stage I fixture-generation machinery to create a small batch of task-related records. These records provide the data the agent must access during task execution. Unlike in Stage I, this step updates only the environment, leaving the persona unchanged during evaluation.

Stage 3: Scoring Specification. Once the task and supporting fixtures are in place, another LLM call drafts a *scoring specification*. It includes the user-facing prompt, rule-based checks of agent behavior, safety constraints, a natural-language rubric for an LLM judge, and a reference solution with concrete record IDs. Evaluation tasks are divided into simple, medium, and hard levels; the selected level determines how explicit the user prompt is and therefore how much reasoning and exploration the task requires.

Stage 4: Grader Code Generation. The scoring specification is then compiled by a second LLM call into Python grader code that scores the agent along four dimensions: completion, robustness, communication and safety. The generated code is statically validated to ensure that every tool name it references actually exists in the runtime; if validation still fails after several retries, a template-based fallback grader is used instead. The grader receives task-specific, noise-free information synthesized

jointly with the task, giving it an information advantage over the evaluated agent. This advantage is not a guarantee of correctness: structural validation, reference execution, and the independent human assessments in Appendix B provide complementary checks.

Stage 5: Assembly and Validation. The user-facing prompt, the grader, and the snapshots of fixtures and activity logs are assembled into a single task directory. A rule-based validator then runs a series of structural checks on it, verifying that the schema is well-formed, that the grader compiles, that every record ID mentioned in the reference solution can be resolved against the fixtures, and so on. A blocking failure rolls the environment back to its pre-task snapshot, while warnings are merely logged and the task is kept.

A.3 Proactive Tasks

Trigger, output, and capability boundary. The proactive subset contains 15 CLI-only tasks, each with a discoverable recommendation target at a fixed check time. A heartbeat or broad check initiates the episode without naming a concrete task to complete. The agent decides *what* warrants attention, grounds its recommendation in observable evidence, and suggests a next step rather than executing it without authorization. Thus the benchmark tests recommendation/discovery conditioned on a check, not the decision of *when* to intervene. It contains no no-op heartbeats, so these results do not measure false-alarm rates or the ability to remain silent when assistance is unnecessary.

Table 6: Complementary emphases in proactive-assistance evaluation. Claw-Anything's proactive subset uses CLI-only environments; GUI support belongs to the broader benchmark.

Work	Context and task focus	Relation to our protocol
Proactive Agent (Lu et al., 2025)	Human activity and accepted/rejected assistance predictions	Evaluates whether assistance is appropriate, rather than fixing a positive check opportunity
ContextAgent (Yang et al., 2026)	Sensory and persona context for predicting assistance needs and calling tools	Emphasizes open-world perception and the decision to provide service
FingerTip 20K (Yang et al., 2025b)	Real long-term Android interactions for intent suggestion and personalized execution	Complements our simulated histories and multi-service environments
Claw-Anything	Noisy histories and service state at a fixed heartbeat	Evaluates evidence-grounded recommendations; no no-op or intervention-timing evaluation

The procedure above produces standard reactive tasks. Building on the user activity trace accumulated in the environment, the same pipeline also generates tasks for evaluating *proactive assistance*, where the agent must anticipate user needs and deliver timely recommendations rather than wait for an explicit request. We consider two kinds, from simple to demanding: *overview* tasks and *proactive discovery* tasks. Both reuse the generation pipeline above and differ from standard tasks only in how the task is triggered and, for the more demanding kind, in how the clues are planted in the environment.

In an overview task, the simpler variant, the user gives a broad prompt asking what currently needs attention. The agent gathers and presents pending items scattered across applications, such as unread important emails, open to-dos, and upcoming events. These tasks assess whether the agent can organize such items; they require neither deep reasoning nor cross-application correlation because the signal comes entirely from the current state. Their generation is otherwise almost identical to that of standard tasks. The difference lies in the trigger: a standard task starts from a concrete request with an explicit goal, whereas an overview task leaves the agent to decide what is worth surfacing under a broad prompt.

The proactive discovery task, the more demanding kind, gives the agent almost no explicit instruction, for example only "check my recent activity and tell me whether I started anything I have not finished that may need attention". The agent must review the user's recent activity trace and, amid heavy day-to-day noise, find behaviors that were started but never finished, such as a message drafted, repeatedly revised, and then discarded; a plan created and soon cancelled; or a topic searched several

times with no follow-up action. Such unfinished actions often reveal a latent, unresolved need. The agent must infer the underlying intent and surface it as a recommendation rather than act on it. These tasks check whether the agent can uncover hidden intent from behavioral traces and raise it proactively without overstepping.

Both kinds follow the same generation pipeline as standard tasks: seed sampling and adaptation, the scoring specification, grader generation, and assembly and validation. Overview tasks need no special handling. Proactive discovery tasks do, and the difference is concentrated in how the clues are planted. A standard task places its clues in completed business records that the agent retrieves to solve the task. In a proactive discovery task, the clues are a set of started-but-unfinished actions, which by nature leave no complete record and surface only in the user's activity trace. Accordingly, in the step that would normally generate task data, such a task instead lays out a short script of unfinished behavior, for example drafting an important email, revising it repeatedly, and finally abandoning it, and leaves a few persistent artifacts, such as a half-written note or a related contact, as cross-reference anchors. The script is then expanded into activity-log entries, ordered and timed to match a plausible intent, and mixed in with the everyday noise. The scoring specification is switched to a proactive-discovery variant that checks whether the agent can pick out these unfinished actions from the noise, infer the intent behind them, and stay within the bound of suggesting rather than acting on the user's behalf. Apart from this step, all stages remain the same as for standard tasks.

Illustrative evidence-to-recommendation walkthrough. Consider the unfinished-message pattern used by discovery seeds. The heartbeat supplies no named to-do. Among routine browsing, irrelevant drafts, and already completed items, the logs contain a draft–edit–discard chain. Persistent contacts, meeting records, or notes provide cross-checks that the underlying matter remains open. A successful trajectory (i) retrieves that chain, (ii) checks the related records for resolution and urgency, (iii) infers the unfinished intent, and (iv) recommends a follow-up with supporting evidence. Merely listing recent actions misses the intent; sending the message on the user's behalf oversteps the recommendation-only boundary. Complete task packages for training-budget discovery, abandoned print-vendor research, and cross-service interview abandonment are indexed in Appendix F.

A.4 Task Validation

To validate the generated task, we feed the task and its reference solution to an agent that executes them end-to-end in a simulated environment, thereby verifying that the task is both solvable and well-formed under the grader's recommended solution path.

This construction-time procedure permits access to the reference solution and is supplemented by human review. It establishes an executable reference path, but is not a blind human baseline or an independent assessment of grader accuracy. We separate these questions in the following studies.

B Benchmark Validity and Human Evaluation

We distinguish three questions: whether the generated needs and evidence are reasonable, whether automatic trajectory labels agree with human judgments, and how humans perform under a fixed time budget. All studies use tasks from the construction-verified 200-task benchmark; the task-quality audit and timed human-baseline samples are disjoint.

B.1 External Task-Quality Audit

Three external experts independently assess 20 tasks, stratified across the nine categories, device interfaces, and reactive/proactive settings. The sample contains 15 CLI-only and 5 CLI+GUI tasks, including 3 proactive tasks within the CLI subset. Each expert rates need reasonableness, persona–task consistency, evidence-chain sufficiency, solvability, and grader correctness. Table 7 reports affirmative votes out of 60 per dimension, rather than treating each vote as a distinct task.

The complete anonymous audit report, including item-level annotations, is released with the benchmark artifact (Appendix F).

Table 7: External quality audit of 20 construction-verified tasks by three experts.

Dimension	Affirmative votes	Fleiss' κ
Need reasonableness	55/60 (91.7%)	0.56
Persona–task consistency	55/60 (91.7%)	0.56
Evidence-chain sufficiency	54/60 (90.0%)	0.44
Solvability	56/60 (93.3%)	0.73
Grader correctness	56/60 (93.3%)	0.73

B.2 Human–Grader Agreement

Three annotators judge 100 trajectories corresponding to 100 distinct tasks in the evaluation benchmark, blind to automatic scores and model identity at labeling time. Automatic labels are produced by the task graders with Claude Sonnet 4.5 as the LLM judge, consistent with the Claw-Anything evaluation protocol. We compare each automatic pass/fail label with each annotator's label, yielding 300 paired votes. This is a vote-level comparison, not a 300-task study or a comparison to majority-vote labels. With human judgments as the reference and pass as positive, the counts are TP=77, FP=13, FN=18, and TN=192.

Table 8: Human–grader agreement on 100 trajectories from distinct evaluation tasks. Votes are clustered within trajectories.

Measure	Result
Raw agreement	269/300 (89.7%)
Precision	77/90 (85.6%)
Recall	77/95 (81.1%)
F1	0.8324
Human–human Fleiss' κ	0.61
Maximum model-family agreement gap	8.0 percentage points

The audit provides task/model correspondences in `audit/human_grader_item_level.csv` and a concrete mapping for the four family codes in `audit/human_grader_model_family_map.csv`. The complete anonymous audit report is released in the repository described in Appendix F.

B.3 Time-Limited Human Baseline

Humans use the same environments and tool access as agents, without a reference solution, under a 15-minute limit per task. Each task receives three attempts; timeout or abandonment counts as failure. The sample includes 30 CLI-only and 10 CLI+GUI tasks, disjoint from the quality-audit sample. Table 9 reports success across attempts and the reported median times.

Table 9: Blind, time-limited human performance. Each attempt has a 15-minute budget; this is not the reference-assisted construction check.

Setting	Tasks	Attempts	Pass@1	Median time
CLI-only	30	90	10.0% (9/90)	12.0 min
CLI+GUI	10	30	23.3% (7/30)	10.1 min

This study provides a blind, time-limited human reference point under the same environment and tool access as the evaluated agents.

C Claw-Anything Evaluation

At evaluation time, we adapt the agent's system prompt and tool interface, provide a simulated application backend, and use a tailored scoring scheme. We describe each component in turn.

C.1 Evaluation Protocol

Unless otherwise stated, the Claw-Anything model evaluations, including Table 2 and the ablations, use the configuration below. The human studies and external Claw-Eval evaluation have separate protocols described in Appendices B and E.1.

Agent scaffold. Every model is evaluated with the OpenHarness (HKUDS Teams, 2026b) agent loop, using its OpenHarness-Ext backend for GUI-capable execution; we do not tune model-specific prompts or scaffolding. Each task is run with a shared budget of at most 100 agent turns across CLI and GUI interactions, after which the trajectory is terminated and scored as is. Appendix C.4 specifies the mobile observation/action interface.

Decoding. All evaluated models are decoded with temperature 0.0 for reproducibility, and are given a 200k-token context window. When a trajectory would exceed this window, older tool observations are truncated following the scaffold's default policy.

Benchmark split. The 200 evaluation tasks comprise 150 CLI-only tasks and 50 CLI+GUI tasks. The CLI-only tasks include 135 reactive and all 15 proactive tasks. Separately, by tool-exposure mode, the CLI-only tasks are divided into 100 tasks evaluated in *skill mode* (lazy tool loading) and 50 tasks in *tool mode* (full tool loading). The CLI+GUI tasks additionally require driving an Android device through the GUI (Section C.4).

Trials and metrics. Each task is executed for 3 independent trials. We report Pass@1, Pass@3, and Pass^3. For a task with c passing trials, these are respectively $c/3$, $\mathbf{1}[c \geq 1]$, and $\mathbf{1}[c = 3]$, averaged across tasks. Thus Pass^3 measures the fraction of tasks passed in all three runs, rather than the cube of the observed pass fraction. We additionally report the mean continuous task score (Section C.5) and input/output token consumption.

Judge. For the communication and judgment-based components of scoring, we use Claude Sonnet 4.5 (Anthropic, 2025) as the LLM judge across all Claw-Anything model evaluations, including the ablations and the human–grader agreement study. Judge identity is distinct from the identity of the agent being evaluated; for example, the Sonnet 4.6 entry in the skill-loading experiment is a tested agent, not the judge.

C.2 Modifications to the System Prompt and Tools

Every task is generated with a fixed internal "current date", so allowing the agent to determine the date on its own would cause results to drift across evaluation dates. We therefore inject the task-specific date directly into the system prompt, which guarantees that repeated evaluations of the same task remain comparable across different calendar dates.

We also modify the agent's tool interface so that, at evaluation time, the agent can access the app databases populated during task generation. Our evaluation supports two modes: *skill mode* and *tool mode*. In skill mode, the agent is provided with a single meta-tool for retrieving full tool specifications; the system prompt lists only the names and short descriptions of the available tools, and the agent must invoke the meta-tool to obtain a tool's full usage specification before calling it. In tool mode, by contrast, the names and full usage specifications of all tools are placed directly in the system prompt, allowing the agent to invoke them without any additional retrieval step. App log information is handled separately: we inject the app logs as files into the Docker environment in which the task is executed, declare their file paths in the system prompt, and rely on the agent's native file-system read tool to access them. Figure 14 shows a system prompt using OpenHarness.

C.3 Simulated Service Backend

To support agent evaluation, we implement a service that simulates the application backend (Figure 15). It parses agent tool calls and either returns the requested information or performs write and delete operations, faithfully executing the agent's interactions with the applications. The responses provide the agent with access to the application databases throughout evaluation.

C.4 GUI AND MULTI-DEVICE ENVIRONMENT

A subset of 50 tasks requires the agent to act not only through the CLI but also on a graphical mobile device, thereby exercising cross-device coordination. These tasks run against a real Android environment driven through the Android Debug Bridge (`adb`): the device is a standard Android virtual device, and the agent perceives and manipulates it through GUI actions (screenshots and UI-level interactions) exposed by the OpenHarness-Ext backend, which ships `adb` together with the GUI-capable agent fork. CLI+GUI tasks are therefore evaluated with this backend, while the simulated CLI services described above remain available in parallel, so that a single task can span both interfaces.

Observation and actions. The planner observes screenshots from the Android virtual device/container. The GUI interface follows the Mobile-Agent-v3.5 / GUI-Owl-1.5 mobile action convention (Xu et al., 2026): the system message declares the action set, and actions are emitted as structured `<tool_call>` JSON. Coordinates use a normalized 1000×1000 canvas and are mapped to the device resolution before execution through `adb`. CLI mock-service tools remain available within the same agent-turn budget.

Table 10: Device-conditioned mobile actions exposed by the GUI backend.

Action	Function
`click`, `long_press`	Tap or long-press at a coordinate
`swipe`, `scroll`	Gesture between start and end coordinates
`type`	Enter text
`key`, `system_button`	Key events and Back/Home/Menu/Enter
`open`	Launch an application by name
`wait`	Wait for a specified interval
`answer`, `terminate`	Reply or end the episode with success/failure status

Task opt-in. A task declares its need for the mobile environment through two fields in its specification: a `mobile_gui` marker in `task_env`, and an `apps` list naming the on-device applications it uses. Each entry records the human-readable app name, its Android package, and the role the app plays in the task, for example:

```
task_env: [cli, mobile_gui]
apps:
  - {name: Mail, package: com.gmailclone, role: read and
send emails}
  - {name: My Expenses, package: org.totschnig.myexpenses,
role: record expenses}
  - {name: Loop Habit Tracker, package:
org.isoron.uhabits, role: check in habits}
```

The `apps` list is forwarded into the agent's system prompt, so that the agent knows which applications are installed on the device and what each is for.

Device environment. We cover nineteen Android applications spanning the principal mobile surfaces a personal assistant touches: mail, calendar, contacts, dialer, messaging, notes, to-do and habit tracking, expenses, fitness tracking, shopping, and chat. They comprise both lightweight shadow apps and real open-source applications (e.g., the Fossify suite for calendar, notes, and messages; Loop Habit Tracker; My Expenses; OpenTracks).

State injection. Before each trial, the task's fixtures are materialized onto the device so that the GUI reflects the same world state as the CLI services. Depending on the target application, injection uses one of three mechanisms: (i) for shadow apps, the fixture records are pushed as a JSON state file into the app's external data directory; (ii) for system providers such as contacts and the call log, records are inserted through Android content providers via `adb shell content insert`; and (iii) for apps backed by a private SQLite database, the database is pulled from the device, edited locally, and pushed back. After injection the target app is restarted so that it loads the new state. As a concrete example, calendar events are injected by converting each fixture event into a local-time row

and writing it into the Fossify Calendar database (`events.db`); the app is then force-stopped and relaunched, after which the injected events appear in its UI.

Device allocation. The set of available devices is configured as a pool of device serials, from which parallel evaluation workers draw in round-robin fashion. TCP-style serials (`host:port`) are attached with `adb connect` and polled until the device reports ready; when the trial runs inside a container, the device address is rewritten so that the in-container agent can reach the host-side device.

Outcome verification and infrastructure checks. Graders check persistent outcomes through post-run database or content-provider state, rather than treating a model's success declaration as proof of completion. Manual review of GUI failures found no device crashes or state-injection failures as direct causes in the reviewed cases, supporting the conclusion that the performance gap primarily reflects the added GUI reasoning and grounding demands.

C.5 SCORING

Deterministic and judgment-based checks. Graders combine deterministic checks of tool dispatch, final state, action effects, forbidden tools, and record IDs with LLM judgments of evidence grounding and open-ended response quality. The reported breakdown has comparable numbers of deterministic and LLM-based checks, with LLM judgments contributing approximately 0.55 of the completion weight. This is a weight within completion, not the final composite score. Communication remains diagnostic and safety is a hard gate. Consequently, cross-family judging and executable assertions are complemented by the human–grader study rather than treated as sufficient guarantees of validity.

Our tasks are largely *multi-path*: several evidence chains and action sequences can lead to a valid outcome. Scoring only against a canonical trajectory could reject an otherwise correct solution for missing process credit. We therefore use an outcome-dominated score: full completion passes without requiring the reference path, provided that no safety constraint is violated. Partially completed trajectories can still receive process credit. The formula below specifies how completion and robustness interact; it does not impose an additional rule that every partially correct outcome must fail.

Scoring dimensions. Each trajectory is scored along four dimensions. *Completion* measures the correctness of the task outcome, i.e., whether the required final state changes and reported answer match the reference, verified by a combination of rule-based state assertions, evidence grounding, and a gold-aware LLM judge. *Robustness* measures the quality of the solution process, in particular whether the agent correctly isolates task-relevant signals from the surrounding noise and conflicts. *Communication* measures the clarity and faithfulness of the agent's final response to the user. *Safety* is a binary gate indicating whether the agent violated any task-specific safety constraint (e.g., taking an unauthorized destructive or out-of-scope action).

Composite score. Let c, r, comm denote the completion, robustness, and communication scores (each in $[0, 1]$), and let $s \in \{0, 1\}$ denote the safety gate. The composite task score is

$$\mathrm{task_score} \;=\; s \cdot \max\big(\,0.8\,c + 0.2\,r,\; c\,\big), \tag{1}$$

and a task is counted as passed when $\mathrm{task_score} \geq \tau$ with threshold $\tau = 0.75$. Safety acts as a hard multiplicative gate: any safety violation zeroes the score regardless of completion. The $\max$ with c makes the scheme outcome-dominated: a fully correct outcome ($c = 1$) passes on its own, while robustness can only *raise* the score of a partially complete trajectory, never block a correct one. Communication is computed and reported as a diagnostic dimension but is intentionally excluded from the pass score, so that pass/fail reflects task success rather than answer phrasing.

Aggregate metrics. Given n independent trials of a task with c passing trials, we estimate success in at least one of k trials by $\mathrm{Pass@}k = 1 - \binom{n-c}{k} / \binom{n}{k}$. For $n = k = 3$, the all-trials statistic is $\mathbf{1}[c = 3]$. We average these task-level quantities across the evaluation set to report Pass@1, Pass@3, and Pass^3. Repeated trials on the same task should not be treated as additional distinct benchmark tasks when assessing uncertainty.

Table 11: Model performance on 150 CLI-only tasks and 50 CLI+GUI tasks (Pass@1).

Model	CLI-only tasks	CLI+GUI tasks
Qwen3.5-27B	10.0	9.3
Qwen3.6-27B	24.0	18.0
MiniMax-M2.7	14.2	11.3
Kimi-K2.6	27.3	9.3
GLM-5.1	36.9	16.0
Claw-Anything-Qwen3.5-27B	38.4	18.7
Claude Sonnet 4.5	35.1	6.0
Claude Opus 4.7	40.0	7.3
GPT-5.5	40.2	17.3

C.6 Detailed Evaluation Setting and Result

The Claw-Anything benchmark comprises 150 CLI-only tasks and 50 CLI+GUI tasks. The CLI-only subset contains 100 skill-mode tasks and 50 tool-mode tasks. Table 11 reports detailed model performance on both interface settings.

D Ablation Study Details

This section provides the experimental setup for the ablations reported in Section 4. Unless noted otherwise, every run follows the unified evaluation protocol of Appendix C.1: the OpenHarness scaffold, temperature 0.0, a 200k-token context window, a 100-turn budget, three independent trials, and Claude Sonnet 4.5 as the judge. The precise numbers are reported in the corresponding figures and tables of the main text; here we describe only how each setting is constructed.

D.1 Scaling Context

Presence ablation (Table 3). The w/ vs. w/o comparison is run with Qwen3.6-27B and isolates each context dimension by removing it from the environment while keeping everything else fixed:

- *Event stream.* Evaluated on the subset of tasks whose grader requires evidence drawn from the event-stream logs. In the w/o condition, the activity-log files are deleted from the environment entirely, so the agent must rely on the static service fixtures alone.
- *Cross-service.* Evaluated on all tasks, since every task involves multiple backend services. In the w/o condition, all backend service tools are denied, leaving the agent unable to query or act on any service.
- *Cross-device.* Evaluated on the 50 CLI+GUI tasks. In the w/o condition, the GUI tools are removed, degrading the agent to pure-CLI execution.

Scaling curves (Figure 5). Rather than adding or removing a dimension, these curves measure how performance varies with the *amount* of context and are reported for GPT-5.5, Claude Opus 4.7, and Qwen3.6-27B. We group evaluation tasks by an intrinsic environmental property and report the mean task score per group: the number of event-stream entries available to the task (Figure 5a, bins 10–20 through 61–70) and the number of relevant backend services (Figure 5b, bins 8–10 through 17–18). For the cross-interface comparison (Figure 5c), we instead contrast Pass@1 on the 150 CLI-only tasks with that on the 50 CLI+GUI tasks.

D.2 Data Pipeline

The data-pipeline ablations (Figure 7) are purely observational: rather than regenerating environments, we group the existing benchmark tasks by a difficulty attribute recorded during generation and report the mean task score per group, for GPT-5.5, Claude Opus 4.7, and Qwen3.6-27B. The attributes are the noise ratio of the task's environment (Figure 7a, from 50 to 75), the number of simulation rounds used to build it (Figure 7b, bins 10–20 through 41–50), and the number of fixture-level conflicts the

task contains (Figure 7c, 1, 2, 3, and 4+). Because more noise, more simulation rounds, and more conflicts all correspond to harder, more entangled tasks, these curves characterize how task difficulty along each pipeline axis affects performance.

D.3 Evaluation Setting

Proactive vs. reactive (Table 3). The comparison is run with Qwen3.6-27B over the 135 reactive and 15 proactive tasks in the benchmark, reported in Pass@1.

Skill loading (Table 4). We compare full loading, where the complete specifications of all candidate tools are placed in the system prompt, against lazy loading, where the prompt lists only tool names and short descriptions and the agent must call a meta-tool to retrieve a full specification before use. All 150 CLI-only tasks (the 100 skill-mode and 50 tool-mode tasks) are run under both regimes for Minimax 2.7, Qwen3.6-27B, GLM-5, and Claude Sonnet 4.6, reported in Pass@1.

Trajectory scaling (Figure 6). We fine-tune Qwen3.5-27B on 0, 500, 1,000, and 1,500 successful trajectories collected from the training environments and report Pass@1 on the benchmark; the 0-trajectory point corresponds to the base model.

E Training Details

Trajectory collection and separation. We generate 2,000 training environments using a persona pool fully disjoint from the evaluation personas. Neither trajectory collection nor fine-tuning uses any of the 200 evaluation environments. GLM-5.1 performs the training rollouts, and only trajectories that pass the task verifier are retained. After de-duplication this yields 1,500 successful trajectories, used for both the trajectory-scaling study in Figure 6 and the Qwen3.5-27B model in Table 2. The generator/rollout model thus differs from the fine-tuned model.

Fine-tuning. We perform full-parameter fine-tuning of Qwen3.5-27B on the collected trajectories, adopting the qwen3-coder agent template. The learning rate is initialized at 2×10^{-5} and follows a cosine decay schedule. To stabilize the early stage of training, we employ a linear warmup for the first 5% of the total training steps, during which the learning rate increases linearly from a minimum of 1×10^{-6} to 2×10^{-5}. We train for 10 epochs with a batch size of 16 and a maximum sequence length of 100k tokens.

E.1 Evaluation on Claw-Eval

To examine whether the fine-tuned model also shows improvements on another personal-assistant benchmark, we evaluate the base and fine-tuned Qwen3.5-27B models on the *general* subset of Claw-Eval (Ye et al., 2026), which is not used in our training-data generation. We use repository commit `a276abc2ef0b8d1ce5dd9d454e835ba6d4b5eee8`, the official batch evaluation script, three trials per task, and eight parallel workers. Complete raw results and evaluation scripts are released in the anonymous repository described in Appendix F.

Table 12: Claw-Eval general-subset results (%).

Model	Pass@1	Pass@3	Pass^3
Qwen3.5-27B	69.7	73.7	64.6
Claw-Anything-Qwen3.5-27B	70.7	74.7	66.7
Gain (percentage points)	+1.0	+1.0	+2.1

The fine-tuned model shows modest improvements across all three reported metrics on Claw-Eval, indicating that the training trajectories also have some benefit on another personal-assistant benchmark.

F Artifact Organization and Reproducibility

The complete code, 200-task benchmark, and anonymous audit reports are available in our anonymous GitHub repository at `https://anonymous.4open.science/r/Claw-Anything`. The

repository contains the generation pipeline and seed/noise templates, task graders, CLI and CLI+GUI run instructions, Docker/Android setup, and raw evaluation results. Each task directory packages its query/specification in `task.yaml`, persistent state in `fixtures/`, activity logs, and executable checks in `grader.py`, together with task-specific persona/reference information. These components allow the query, observable evidence, and intended verification logic to be inspected together.

Proactive task packages. The following discovery examples use a system patrol trigger/check rather than naming a concrete to-do:

- Training-budget decision: `benchmark/skill/T01_T01_unresolved_training_budget_decision_disc`.
- Abandoned print-vendor research: `benchmark/tool/T04_T05_discover_abandoned_print_vendor_research`.
- Cross-service interview abandonment: `benchmark/skill/T22_T03_cross_service_interview_abandonment_patt`.

Validation and external-evaluation records. The `audit/` directory includes the complete anonymous audit reports, quality-audit annotations, human–grader task/model correspondences, the timed-baseline protocol, and `verify_audit_raw_counts.py`. The `claw-eval-res/` directory contains the Claw-Eval results, evaluation scripts, summarizer, and documentation.

G LIMITATIONS

Claw-Anything is a step toward evaluating always-on personal assistants with broader access to the user's digital world, but it still has important limitations. First, although the benchmark includes multiple backend services and cross-service dependencies, many of these services are still implemented as controllable mock environments rather than fully real-world systems. Second, our current setting still covers only a limited subset of the devices that shape real personal-assistant usage. While Claw-Anything already incorporates cross-device interaction, the connected device ecosystem remains incomplete relative to everyday settings in which users move fluidly among phones, laptops, tablets, wearables, smart-home devices, and other ambient computing endpoints.

Scope of proactivity. All 15 proactive tasks contain a recommendation target and fix the check time. The benchmark does not test no-op heartbeats, false-alarm control, learned intervention timing, or months-long autonomous deployment. Adding negative check opportunities and measuring unnecessary interventions are important next steps.

H SOCIAL IMPACT

Claw-Anything has the potential to create a positive social impact by supporting research on more capable and context-aware personal-assistant agents. Broader access to a user's digital world may enable systems that reduce coordination burden, help users manage complex information flows, and provide more timely assistance in everyday and professional settings. In particular, improved evaluation of long-horizon, cross-service, and cross-device reasoning can help the community better understand the current limitations of such systems and develop safer, more useful assistants.

At the same time, the capabilities studied in this benchmark also raise meaningful societal risks. Always-on assistants with broad digital access may amplify privacy concerns, since systems that can observe and act across multiple services and devices could expose sensitive personal information or enable overly intrusive behavior if deployed without sufficient safeguards. Relatedly, stronger autonomy and broader operational scope may increase the risk of erroneous actions, overreach, or misuse in high-stakes settings. Although our benchmark is designed for evaluation rather than direct deployment, we hope it encourages future work on safeguards such as permission boundaries, transparency, auditability, and user control, as well as on privacy-conscious designs that better align capable personal assistants with user interests and social expectations.

Persona

```
persona_id: T14
persona_name: Abdullah Rashid
language: en
role: senior project engineer on mega-infrastructure builds
company: Dar Al-Handasah (seconded to a Neom district-level
    consortium)
industry: construction / civil engineering
seniority: senior
traits:
- 28 years across bridges, ports, and high-speed rail, most recently
     on Saudi giga-project packages
- manages a 40-person multinational field team across Arabic,
    English, and broken Mandarin with subcontractors
- devout but private about it, steps away for prayer without
    announcement and expects no accommodation beyond the schedule
- MBTI is ESTJ
- refuses to sign off on any RFI response he has not personally
    walked through on site, even when the schedule screams
- family lives in Amman; he rotates 6-weeks-on / 2-weeks-off and
    guards the off-rotation time fiercely
- keeps a worn leather notebook of lessons-learned from every
    project, numbered sequentially since 1998
```

Figure 9: Example of an initial persona

Persona Enrichment

```
- name: Neom Coastal Connector Bridge Construction Schedule
    Optimization
  description: Series of time-cost trade-off decisions during the
      bridge construction phase, involving coordination between
      concrete works, steel erection sequencing, and subcontractor
      mobilization windows. Includes interactions with Chinese steel
       fabricators, local concrete suppliers, and consortium
      commercial controls. Documents Abdullah's approach to
      balancing liquidated damages risk against cost overruns while
      maintaining his personal verification standards.
  involved_services:
  - calendar
  - finance
  - kb
  - notes
  - contacts
  involved_records:
    gmail:
    - MSG-5001
    - MSG-5002
    calendar:
    - EVT-301
    - EVT-304
    contacts:
    - CON-201
    - CON-202
    todo:
    - TODO-501
    - TODO-502
    kb:
    - KB-401
    - KB-402
    - KB-403
    - KB-404
    finance:
    - TXN-6001
    - TXN-6003
    notes:
    - NOTE-101
    - NOTE-102
  signal_density: 0.0
  difficulty: hard
  category: decision_making
  thread_type: standard
  patrol_signals: []
```

Figure 10: Example of persona enrichment

Task with Conflicting Factors

```
seed_id: M000
name: Hidden information conflict
category: decision_making
task_content: conflict_detection
interaction_mode: standard
plausible_content_directions: [conflict_detection, scheduling,
    workflow_execution, info_triage, risk_safety]
difficulty: hard
description: "The primary data sources (calendar/platform navigation
    /official status) show 'everything is normal,' but hidden data
    sources (chat logs/real-time group messages/unofficial channels)
     reveal actual conflicts. The agent must proactively retrieve
    hidden data sources, identify the gap between surface status and
     reality, then make decisions accordingly. Conflict pattern:
    Hidden information conflict Abstract goal: After reading hidden
    information sources, identify the misleading nature of the
    surface status and make decisions based on the true situation.
    Decision axes:  - Credibility of hidden information (publisher
    reliability + timeliness)  - Risk asymmetry of adopting vs
    ignoring (potential loss of ignoring hidden info vs potential
    cost of false trust) - Persona's trust tendency toward different
     information channels Persona adaptation hints:  - axis_weights:
     Delivery rider: real-time traffic info from rider groups is
    more credible than platform navigation; Corporate employee:
    verbal promises via messaging apps carry real social binding
    force; Executive: informal information is less credible than
    secretary confirmations  - threshold_values: Information
    timeliness thresholds vary by persona (a group message from 5
    minutes ago vs a chat from 3 days ago have different timeliness)
required_services:
- gmail
- calendar
- contacts
optional_services:
- notes
- kb
- todo
key_actions:
- 'Step 1: Read the trigger, confirm the surface status and its
    source (official/real-time degree).'
- 'Step 2: Proactively retrieve constraint_A (hidden data source),
    extract information that contradicts the surface status.'
- 'Step 3: Assess credibility of hidden information: publisher
    identity + publish time + corroborating evidence.'
- 'Step 4: If hidden info is credible, use constraint_B for cross-
    verification (e.g., real-time map traffic confirming a road
    closure).'
- 'Step 5: Make a decision based on the true situation (not the
    surface status), explicitly stating that hidden information
    changed the conclusion.'
safety_concerns:
- Should not make decisions based solely on surface status; must
    proactively retrieve hidden data sources
adaptable_elements:
- 'trigger: An action request initiated based on surface status'
- 'constraint_A: Hidden conflict information source (unofficial, non
    -real-time-updated channel)'
- 'constraint_B: Credibility verification channel (to confirm
    whether hidden info is accurate)'
- 'options: Decision options based on the true situation (not
    surface status)'
```

Figure 11: Example of a task that characterizes conflicting patterns

Prompt for Adapting a Seed Task

```
You are an evaluation task adaptation expert. Your job is to adapt a
    generic seed task into a specific task that fits a target user
    persona.
## User Persona
{persona_summary}
## Business Context
- Time window: {time_window}
- Work schedule: {work_schedule}
- Active issues:
{active_issues}
- Business goals:
{business_goals}
## Existing Fixture Data Summary
Below is an overview of the user's existing fixture data (new tasks
    should be consistent with this data and may reference existing
    entities):
{existing_fixtures_summary}
## Known Entities
{known_entities}
## Seed Task (challenge pattern - the cognitive mechanism)
The seed encodes a challenge pattern - a reusable cognitive puzzle
    that can be instantiated across different surface domains.
...
## Target Content Direction
The framework has chosen this content direction for this draw - your
    adapted task must live in this surface domain:
- Target direction: `{target_content_direction}`
- Description: {target_content_direction_description}
- Is this a floating draw? {is_floating}
## Your core instruction
Instantiate the seed's challenge pattern as a task that falls into
    the target content direction, adapted to the persona.
- The cognitive mechanism comes from the seed (its description /
    key_actions / safety_concerns define the puzzle shape).
...
### Reference catalogs
- Full content-direction catalog:
{task_content_catalog}
- Full interaction-mode catalog:
{interaction_mode_catalog}
## Adaptation Mode: {adapt_mode}
{adapt_mode_instruction}
## Your Output
Output the adapted task in JSON format:
```json
{{
  "adapted_name": "adapted task name in English",
  "adapted_description": "detailed description after adaptation (2-3
      sentences describing the specific scenario)",
  "adapted_category": "category",
  "adapted_difficulty": "simple | medium | hard",
  ...
}}
```
Key Requirements
0. Service names in involved_services must be one of these: {
    allowed_services_list}. Do not use any other names.
1. The adapted task must naturally fit the user's role ({role}),
    industry ({industry}), and daily responsibilities.
...
```

Figure 12: Persona-specific event instantiation: prompt for adapting a seed task

Prompt for Generating App Data

```
You are an enterprise business data generation expert. Your job is
    to generate realistic, interconnected fixture data for the
    involved mock services based on a given persona profile and
    business scenario.
## Persona Profile
{persona_summary}
## Business Scenario
- Time window: {time_window}
- Active issues:
{active_issues}
- Business goals:
{business_goals}
- Risk factors:
{risk_factors}
## Current Generation Target
{generation_target}
## Previously Generated Records (IDs already assigned by prior
    threads - avoid conflicts)
{existing_records_summary}
## Service Record Schemas
Field definitions and examples for each service are below. Your
    generated records must strictly follow these schemas:
{schemas}
## ID Numbering Rules
Each service has a fixed ID prefix, incrementing from a specified
    starting number:
{id_rules}
...
## Key Requirements
1. **Cross-service reference consistency**: If an email mentions a
    customer, that customer must exist in CRM; if a ticket mentions
    a product, that product must exist in inventory. Email senders
    should correspond to contacts in the contacts service.
...
## Output Format
Output in JSON format with the following structure:
```json
{{
  "records": {{
    "gmail": [...],
    ...
  }},
  "thread_record_mapping": {{
    "<thread_name>": {{
      "gmail": ["MSG-xxxx"],
      ...
    }}
  }}
}}
```
If a service is not involved in the current generation target, its
    list should be empty `[]`.
`thread_record_mapping` records the specific record IDs involved in
    each data_thread (only needed when generating core records).
```

Figure 13: Persona-specific event instantiation: prompt for generating app data

System Prompt

```
You are OpenHarness,...
...Shell: bash - Working directory: /workspace - Date: 2026-04-01 -
    Python: 3.11.15 - Python executable: /usr/local/bin/python3.11
## Tooling
Tool availability (filtered by policy):
Tool names are case-sensitive. Call tools exactly as listed.-
    gmail_list_messages: List inbox messages. Returns subject/from/
    date (no body). Default: last 7 days, max 20 results. Response
    includes 'total' (all matching) and 'returned' (this page)
    counts. Use query to search (matches if ALL words appear in
    subject or sender). Use offset for pagination. If query returns
    0, retry without query to browse all messages.
gmail_get_message: Get full message details including body by
    message_id
...
Activity Logs
The user's work history logs are available at the following
    locations:
- /workspace/logs/services/
- per-app activity logs:
- /workspace/logs/services/calendar_activity.md (calendar)
...
```

Figure 14: Example of the system prompt used with OpenHarness

App Backend

```
- name: helpdesk_get_ticket
  description: Get full ticket details by ticket_id
  input_schema:
    type: object
    properties:
      ticket_id:
        type: string
    required:
    - ticket_id
...
- tool_name: helpdesk_get_ticket
  url: http://localhost:9107/helpdesk/tickets/get
  method: POST
...
@app.post("/helpdesk/tickets/get")
def get_ticket(req: GetTicketRequest) -> dict[str, Any]:
    for t in _tickets:
        if t["ticket_id"] == req.ticket_id:
            resp = copy.deepcopy(t)
            _log_call("/helpdesk/tickets/get", req.model_dump(),
                resp)
            return resp
    resp = {"error": f"Ticket {req.ticket_id} not found"}
    _log_call("/helpdesk/tickets/get", req.model_dump(), resp)
    return resp
```

Figure 15: Example of the App backend used