METACLAW: JUST TALK – AN AGENT THAT META-LEARNS AND EVOLVES IN THE WILD

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Model (LLM) agents deployed in dynamic environments often remain static, becoming stale as task distributions drift. Existing solutions either store raw trajectories without knowledge distillation, maintain skill libraries disconnected from weight optimization, or require service downtime for retraining. We present **MetaClaw**, a continual meta-learning framework that jointly evolves a base LLM policy and a skill library of reusable behavioral instructions through two complementary mechanisms. First, *skill-driven fast adaptation* synthesizes new skills from failure trajectories via an LLM evolver, providing immediate, zero-downtime improvements. Second, *opportunistic policy optimization* performs gradient-based updates via cloud-based RL during user-inactive windows, managed by a scheduler (OMLS) that monitors activity signals like Google Calendar. To ensure data integrity, a versioning mechanism strictly separates support data (for skill evolution) from query data (for RL updates), preventing stale reward contamination. Built on a proxy-based architecture, MetaClaw scales to production LLMs without local GPUs. Experiments on MetaClaw-Bench and the AutoResearchClaw pipeline demonstrate that skill-driven adaptation alone improves accuracy by up to 32% relative. The full framework elevates Kimi-K2.5 from 21.4% to 40.6% accuracy, nearly matching the GPT-5.2 baseline (41.1%), while achieving an 8.25× gain in end-to-end task completion and an 18.3% boost in robustness.

1 INTRODUCTION

Large language model (LLM) agents have demonstrated remarkable capabilities across complex tasks (Yao et al., 2022; Shinn et al., 2023), yet agents deployed in the wild remain largely static, trained once and served unchanged regardless of how the user's needs evolve (Zhang et al., 2025b; Naihin et al., 2023; Song et al.). Consider OpenClaw (OpenClaw, 2026), an open-source CLI agent platform connecting to 20+ messaging channels, where a single user's workload may shift from multi-step file system operations one week to multi-agent messaging workflows the next. As the task distribution drifts, a frozen model becomes increasingly misaligned with actual usage patterns, repeatedly failing on task types underrepresented during pretraining.

Existing approaches to agent adaptation fall into three broad categories, each with notable limitations. Memory-based methods (Shinn et al., 2023; Zhao et al., 2024; Fang et al., 2025; Tang et al., 2025; Ouyang et al., 2025; Chhikara et al., 2025; Liu et al., 2026a) store raw conversation trajectories for future retrieval, but such trajectories are verbose and redundant, preventing the agent from extracting transferable behavioral patterns. Skill-based methods (Xia et al., 2026; Zhang et al., 2025a; 2026b; Wu et al., 2025; Zhang et al., 2026a) compress experience into reusable behavioral instructions, yet treat the resulting skill library as a static database never coordinated with weight optimization. RL-based methods (Schulman et al., 2017; Ahmadian et al., 2024; Shao et al., 2024; Feng et al., 2025; Zheng et al., 2025) update model weights, but operate in small-scale or offline settings and ignore a critical data validity problem: once skills have evolved, trajectories collected under the old skill context carry stale rewards that contaminate gradient updates if reused without filtration. A common thread across all three categories is that each addresses only one aspect of adaptation in isolation, leaving the complementary dimensions unexploited.

We observe that two distinct adaptation timescales are naturally complementary: behavioral heuristics (e.g., "always verify a file path before reading," "confirm before destructive commands") can be

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.

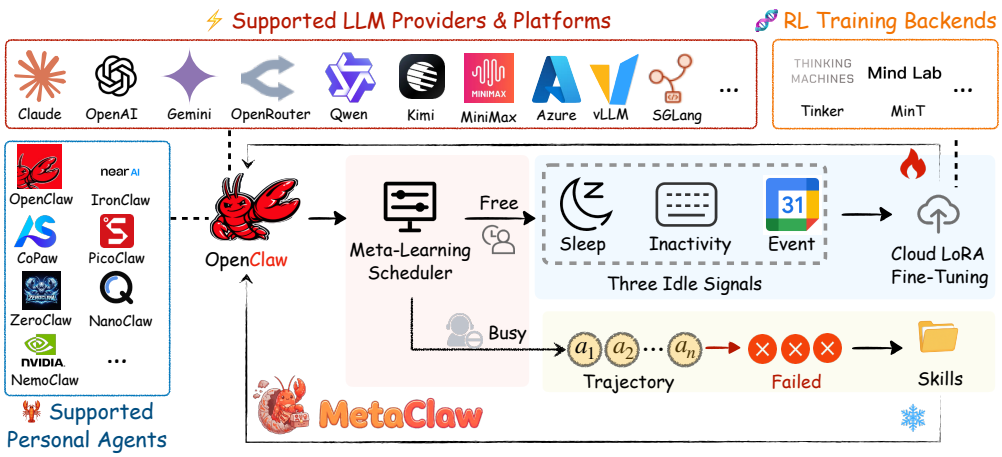

Figure 1: Overview of MetaClaw. The framework improves the meta-model $\mathcal{M} = (\theta, \mathcal{S})$ via two complementary loops operating at different timescales. Skill-driven fast adaptation (left) analyzes failed trajectories and instantly expands the skill library $\mathcal{S}$ without parameter updates, taking effect immediately for subsequent tasks. Opportunistic policy optimization (right) accumulates post-adaptation trajectories and, once sufficient data is available, leverages idle signals (sleep, inactivity, calendar) detected by the Opportunistic Meta-Learning Scheduler to trigger RL-based weight updates on θ via Cloud LoRA fine-tuning.

distilled and injected in seconds, while underlying policy improvements require gradient-based optimization over hours. These mechanisms are mutually reinforcing—a superior policy generates more informative failures for skill synthesis, while richer skills produce higher-reward trajectories for RL. No existing system unifies these two forms of adaptation into a coherent framework that exploits this virtuous cycle.

We present **MetaClaw**, a continual meta-learning (Finn et al., 2017) framework that jointly maintains a base LLM policy and an evolving skill library of reusable behavioral instructions. The skill library serves a dual role: as a *meta-parameter* that accumulates behavioral knowledge across the task stream, and as an *adaptation basis* from which task-specific skills are retrieved at inference time. MetaClaw improves both components through two mechanisms. *Skill-driven fast adaptation* performs gradient-free skill evolution: an LLM analyzes failure trajectories and synthesizes new behavioral instructions (Xia et al., 2026) that take effect immediately with zero service downtime. *Opportunistic policy optimization* uses RL with a process reward model (PRM) (Zhang et al., 2025c) to update model weights via cloud (TML, 2026) LoRA fine-tuning (Hu et al., 2021), optimizing post-adaptation performance. Two design principles govern their coordination. First, *when* to run policy optimization: our Opportunistic Meta-Learning Scheduler (OMLS) monitors three idle signals, i.e., configurable sleep hours, system keyboard inactivity, and Google Calendar event occupancy, and triggers weight updates only during user-inactive windows, eliminating downtime. Second, *which data* to use: we distinguish *support data* (failure trajectories consumed by skill evolution) from *query data* (trajectories collected after new skills take effect). Only query data, reflecting the agent's post-adaptation behavior, is valid for RL; support data carries rewards conditioned on the old skill context and is excluded. Our skill generation versioning mechanism enforces this separation by stamping each trajectory with its skill generation index and flushing stale samples from the training buffer whenever skills evolve.

In summary, our primary contribution is MetaClaw, a continual meta-learning framework that unifies skill-driven fast adaptation with opportunistic policy optimization, enabling deployed LLM agents to evolve continuously through a proxy-based architecture without requiring a local GPU. We evaluate on MetaClaw-Bench, a new benchmark of 934 questions over 44 simulated workdays, where each day forms a sequential, feedback-driven multi-round session of real CLI tasks. Experiments with GPT-5.2 and Kimi-K2.5 show that Skill-driven adaptation alone improves relative accuracy by 32.2%. The full framework boosts Kimi-K2.5 from 21.4% to 40.6%, improves task completion by 8.25×, and nearly closes the gap with the GPT-5.2 baseline. We further validate on AutoResearchClaw, a 23-stage autonomous research pipeline, where skill injection increased the composite robustness score by 18.3%, proving cross-domain generalization.

2 PROBLEM SETUP

We consider a deployed CLI agent that serves a user over a stream of tasks $\tau_1, \tau_2, \ldots$ drawn from a non-stationary distribution $p_t(\tau)$. Each task τ_i consists of a user instruction and environmental context (file system state, shell history, etc.), and the agent must produce a sequence of actions $a_{1:T}$ to accomplish the task. The agent's behavior at any point in time is fully determined by a *meta-model*:

$$\mathcal{M} = (\theta, \mathcal{S}), \tag{1}$$

where θ denotes the parameters of the base LLM policy and $\mathcal{S} = \{s_1, s_2, \ldots, s_K\}$ is a library of *skill instructions*, i.e., concise, reusable behavioral directives injected into the agent's system prompt at inference time. Given a task τ, the agent generates actions according to:

$$a \sim \pi_\theta(\cdot \mid \tau,\ \textsc{Retrieve}(\mathcal{S}, \tau))\,, \tag{2}$$

where $\textsc{Retrieve}(\mathcal{S}, \tau) \subseteq \mathcal{S}$ selects the most relevant skills for the current task via embedding-based retrieval.

The meta-model $\mathcal{M}$ evolves over the task stream as the agent accumulates experience. We distinguish two types of trajectory data based on their role in this evolution. *Support data* $\mathcal{D}^{\text{sup}}$ consists of trajectories whose failures drive adaptation of the skill library $\mathcal{S}$; these trajectories are consumed by the adaptation process and reflect pre-adaptation behavior. *Query data* $\mathcal{D}^{\text{qry}}$ consists of trajectories collected after adaptation has taken effect; these reflect the agent's post-adaptation behavior and are used to optimize the policy parameters θ. Maintaining a strict separation between support and query data is essential: mixing them would cause θ to be optimized against stale reward signals that no longer reflect the agent's current capabilities.

The goal of MetaClaw is to continuously improve $\mathcal{M}$ over the task stream, not merely to solve each task in isolation, but to *become progressively better at adapting* to new tasks as they arrive. This positions MetaClaw as a continual meta-learning system: the agent learns from a non-stationary task stream while simultaneously improving its own adaptation capability.

3 METACLAW

3.1 OVERVIEW

MetaClaw improves the meta-model $\mathcal{M} = (\theta, \mathcal{S})$ through two complementary mechanisms operating at different timescales (Figure 1). *Skill-driven fast adaptation* analyzes failure trajectories and synthesizes new skill instructions that are immediately injected into the agent's prompt, evolving $\mathcal{S}$ without touching model weights. *Opportunistic policy optimization* uses post-adaptation trajectories to update θ via reinforcement learning, deferred to user-inactive windows by the Opportunistic Meta-Learning Scheduler (OMLS). A skill generation versioning mechanism ensures that policy optimization always trains on query data collected under the current skill library, preventing stale reward contamination from support data. The two mechanisms are mutually reinforcing: a better θ produces more informative failures for skill synthesis, and richer skills produce higher-reward trajectories for policy optimization. This virtuous cycle enables the system to *learn to become better at adapting*. The complete procedure is summarized in Algorithm 1.

3.2 SKILL-DRIVEN FAST ADAPTATION

Given the current meta-model $(\theta, \mathcal{S}_g)$, the agent executes tasks and collects trajectories. Trajectories that reveal failure modes form the support set $\mathcal{D}_g^{\text{sup}}$. Skill-driven adaptation evolves the skill library via a gradient-free experience distillation process:

$$\mathcal{S}_{g+1} = \mathcal{S}_g \cup \mathcal{E}(\mathcal{S}_g, \mathcal{D}_g^{\text{sup}}), \tag{3}$$

where $\mathcal{E}$ is a *skill evolver*, an LLM that analyzes failure trajectories and synthesizes new behavioral instructions. The index g denotes the *skill generation*, incremented each time the library changes. This step modifies only $\mathcal{S}$, leaving θ fixed, and takes effect immediately for all subsequent tasks. Because skill injection operates through the prompt rather than model parameters, fast adaptation incurs zero service downtime.

Algorithm 1 MetaClaw: Continual Meta-Learning for Deployed LLM Agents

Require: Meta-model $\mathcal{M} = (\theta_0, \mathcal{S}_0)$, skill evolver $\mathcal{E}$, task stream $\{\tau_i\}$, PRM R, OMLS idle detector
Ensure: Continuously improved meta-model $\mathcal{M}$

Initialize skill generation $g \leftarrow 0$, RL buffer $\mathcal{B} \leftarrow \varnothing$
for each task τ_i in stream **do**
  ▷ *Serve task with current meta-model*
  $\mathcal{S}_{\tau_i} \leftarrow \textsc{Retrieve}(\mathcal{S}_g, \tau_i)$ *// retrieve relevant skills*
  $\xi_i \leftarrow \textsc{Execute}(\pi_\theta(\cdot \mid \tau_i, \mathcal{S}_{\tau_i}))$ *// collect trajectory*
  $r_i \leftarrow R(\xi_i)$; stamp (τ_i, ξ_i, r_i) with generation g
  if ξ_i reveals failure **then**
    Add (τ_i, ξ_i) to support set $\mathcal{D}_g^{\text{sup}}$
  else
    Add (τ_i, ξ_i, r_i, g) to RL buffer $\mathcal{B}$
  end if
  ▷ *Skill-driven fast adaptation (when failures accumulate)*
  if $|\mathcal{D}_g^{\text{sup}}| \geq$ threshold **then**
    $\Delta\mathcal{S} \leftarrow \mathcal{E}(\mathcal{S}_g, \mathcal{D}_g^{\text{sup}})$ *// synthesize new skills from failures*
    $\mathcal{S}_{g+1} \leftarrow \mathcal{S}_g \cup \Delta\mathcal{S}$ *// evolve skill library*
    Flush all samples with version $\leq g$ from $\mathcal{B}$ *// support-query separation*
    $g \leftarrow g + 1$
  end if
  ▷ *Opportunistic policy optimization (when user is idle)*
  if OMLS detects idle window **and** $|\mathcal{B}| \geq$ batch size **then**
    $\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}_{(\tau,\xi,r,g')\sim\mathcal{B}}[R(\pi_\theta(\cdot \mid \tau, \mathcal{S}_{g'}))]$ *// RL update*
    Hot-swap model weights *// deploy updated θ*
  end if
end for

This mechanism is gradient-free by design, not by approximation. The skill library $\mathcal{S}$ lives in a discrete natural-language space where gradient descent is ill-defined; LLM-based failure analysis is the natural adaptation mechanism for this space.

The skill library $\mathcal{S}$ plays a dual role in the learning structure. As a *meta-parameter*, $\mathcal{S}$ accumulates behavioral knowledge across the entire task stream, with each skill generation $\mathcal{S}_{g+1} \supseteq \mathcal{S}_g$ representing the system's growing operational knowledge. As an *adaptation basis*, $\textsc{Retrieve}(\mathcal{S}, \tau)$ extracts a task-specific subset at inference time, providing instant specialization without any parameter update. This dual character arises because natural-language instructions are inherently cross-task transferable: a skill distilled from one failure (e.g., "verify file path before reading") generalizes to all tasks involving file operations. Unlike systems where task-specific adaptations are ephemeral and discarded after each task, each adaptation episode in MetaClaw contributes lasting knowledge to the meta-model, making knowledge accumulation a feature rather than a side effect.

3.3 Opportunistic Policy Optimization

After each skill-driven adaptation step, the agent continues serving tasks under the latest skill library. Because policy optimization is deferred to idle windows, the skill library may have advanced through several generations by the time training begins. Let g^* denote the current skill generation when a training window opens. The RL buffer $\mathcal{B}$ accumulates query trajectories across all post-adaptation generations, and policy optimization updates θ over this buffer:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \mathbb{E}_{(\tau,\xi,g')\sim\mathcal{B}}\left[R(\pi_\theta(\cdot \mid \tau, \mathcal{S}_{g'}))\right], \tag{4}$$

where $g' \leq g^*$ is the skill generation under which each trajectory was collected, and R is a process reward model (PRM) score. The versioning mechanism (Section 3.4) guarantees that $\mathcal{B}$ contains only query data, i.e., every sample reflects post-adaptation behavior under its respective skill generation. Crucially, policy optimization does not optimize θ for raw task performance, but for how well the agent performs *after skill adaptation*. A better θ yields a meta-model from which skill-driven adaptation produces stronger post-adaptation behavior, resulting in an improved meta-model $\mathcal{M}' = (\theta_{t+1}, \mathcal{S}_{g^*})$.

In practice, policy optimization is realized via cloud LoRA fine-tuning using GRPO, deferred to idle windows by the Opportunistic Meta-Learning Scheduler (Section 3.5). Importantly, training is initiated only after the query buffer $\mathcal{B}$ has accumulated a sufficient number of trajectories; launching

RL with too few samples leads to high-variance gradient estimates and unstable policy updates. This means policy optimization naturally lags behind skill-driven adaptation by days or longer, reinforcing the asymmetry between the two timescales: skills evolve continuously, while the policy improves in discrete, data-gated steps.

3.4 Skill Generation Versioning

The support-query separation defined in Section 2 must be enforced in MetaClaw's online setting, where tasks arrive sequentially and skill evolution is triggered asynchronously. Without a dedicated mechanism, support data can leak into the policy optimization buffer.

The problem is concrete: a trajectory (τ_i, ξ_i) that triggers skill evolution from $\mathcal{S}_g$ to $\mathcal{S}_{g+1}$ carries a reward r_i reflecting performance under $\mathcal{S}_g$, *before* the new skill existed. If this trajectory enters the RL buffer, policy optimization receives a gradient that penalizes θ for a failure that skill-driven adaptation has already corrected, optimizing for pre-adaptation rather than post-adaptation performance and violating the meta-learning objective in Eq. 4.

We enforce separation via a *skill generation version* g_i stamped on each collected sample:

- **Support set** $\mathcal{D}_g^{\text{sup}}$: trajectories collected under $\mathcal{S}_g$ whose failures trigger skill evolution $\mathcal{S}_g \to \mathcal{S}_{g+1}$. These are consumed by the skill evolver and *discarded from the RL buffer*.
- **Query set** $\mathcal{D}_{g+1}^{\text{qry}}$: trajectories collected after $\mathcal{S}_{g+1}$ takes effect. Only these, reflecting the agent's post-adaptation behavior, are eligible for policy optimization gradient updates.

When the skill generation counter advances from g to $g+1$, the trainer flushes all samples with version $\leq g$ from its buffer. This ensures policy optimization always updates θ with respect to the agent's adapted behavior, preserving the integrity of the meta-learning structure.

3.5 Opportunistic Meta-Learning Scheduler

Policy optimization requires a model weight hot-swap upon completion, which briefly interrupts inference. In a deployed interactive system, this creates a tension: policy optimization must run periodically to improve θ, but it must not degrade the user's experience.

We introduce the *Opportunistic Meta-Learning Scheduler* (OMLS), a background daemon that defers policy optimization to periods when the user is not actively interacting with the agent. OMLS monitors three complementary idle signals:

(1) Sleep window. The user configures a sleep schedule (e.g., 23:00–07:00). During this window, the system is guaranteed to be idle, providing the largest contiguous training block.

(2) System inactivity. OMLS polls the operating system's input device idle timer (e.g., `ioreg HIDIdleTime` on macOS). If no keyboard or mouse activity is detected for δ minutes (default: 30), a training window opens. Upon renewed input, the trainer pauses gracefully via mid-batch checkpointing.

(3) Calendar-aware scheduling. OMLS queries the user's Google Calendar API. When the current time falls within a scheduled meeting, the user is presumed unavailable, opening an opportunistic training window. This is the most anticipatory of the three signals: it leverages the user's own schedule to predict idle periods proactively.

A training window opens when *any* signal indicates user absence and closes when *any* signal indicates the user has returned. The RL trainer supports pause/resume across fragmented idle windows, accumulating gradient steps opportunistically without requiring a single long contiguous block.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

4.1.1 BENCHMARK AND EVALUATION PLATFORM

MetaClaw-Bench. We construct MetaClaw-Bench, a continual agentic benchmark comprising two complementary evaluation parts (934 questions total across 44 simulated workdays) for evaluating an agent's ability to adapt across a sequential stream of real-world CLI tasks. Existing agent benchmarks present tasks as independent episodes, providing no mechanism to assess whether an agent improves from accumulated experience. MetaClaw-Bench addresses this gap by structuring evaluation as multi-workday simulations in which the agent operates under consistent workspace and policy rulesets that evolve through user feedback.

1) Part I structures evaluation as a 30-workday simulation (346 questions, days 01–30, 10–15 per day). The workspace shares partial common files daily, with an independent folder for exclusive daily working files. Each question incorporates the previous round's evaluation outcome as corrective feedback context. Task difficulty increases monotonically with day index, with days 25–30 requiring sophisticated multi-step reasoning.

2) Part II extends the evaluation to a 14-workday simulation (588 questions, 42 per day: 434 multi-choice and 154 file-check). Part II's file-check tasks are rule-based transformations where compliance with behavioral heuristics is the primary bottleneck, making them more amenable to skill distillation.

Questions fall into two types: *file-check* tasks (structured edits or transformations producing output files validated by automated checkers) and *multi-choice* tasks (conceptual procedural questions on domain-specific rules). We report two primary metrics across both parts: overall accuracy (mean per-question score) and file-check completion rate (fraction of file-check outputs passing all automated checker assertions simultaneously).

Downstream evaluation: AutoResearchClaw. To test whether MetaClaw's adaptation mechanisms generalize beyond CLI-task benchmarks, we additionally evaluate on AutoResearchClaw (Liu et al., 2026b), a fully autonomous 23-stage research pipeline that transforms a single research idea into a conference-ready paper, covering literature search, hypothesis generation, experiment design, code synthesis, sandbox execution, result analysis, paper drafting, and multi-agent peer review. Unlike MetaClaw-Bench's structured file-check and multi-choice tasks, AutoResearchClaw presents an open-ended, long-horizon agentic workload where failures manifest as stage retries, excessive refinement cycles, and incomplete pipeline runs. We report four pipeline-level metrics: *stage retry rate*, *refine cycle count*, *pipeline stage completion* (out of 19 scorable stages), and a *composite robustness score* (weighted average of stage completion rate at 40%, retry reduction at 30%, and refine cycle efficiency at 30%).

Baselines and Implementation Details. We evaluate two frontier LLMs as backbone policies: GPT-5.2 (OpenAI, 2025) and Kimi-K2.5 (Team et al., 2026). We compare three conditions: 1) Baseline: the base model served without any adaptation mechanism. 2) MetaClaw (Skills): the base model augmented with skill-driven fast adaptation; after each failed trajectory, the skill evolver synthesizes behavioral instructions immediately injected into the system prompt, with top-k retrieval via cosine similarity over sentence embeddings. 3) MetaClaw (Full): the full pipeline combining skill-driven fast adaptation with opportunistic policy optimization via RL (5-day training run), evaluated for Kimi-K2.5 only, as it requires a cloud LoRA training endpoint configured for the target backbone. For the AutoResearchClaw evaluation, we deploy MetaClaw's skill-driven fast adaptation within AutoResearchClaw's pipeline executor. After each pipeline run, failures and warnings from all 23 stages are captured as structured lessons and converted into reusable skill files via MetaClaw's lesson-to-skill evolver. On subsequent runs, accumulated skills are injected into the system prompt of all 18 LLM-driven stages. We run controlled A/B experiments with the same research topic, backbone LLM, and pipeline configuration, differing only in whether MetaClaw's skill injection is active.

Table 1: Main results on MetaClaw-Bench (Parts I and II). Acc.: mean per-question accuracy. Compl.: file-check completion rate. Best result per model per part is **bolded**.

		Part I (30 days, 346 Q)		Part II (14 days, 588 Q)	
Model	**Condition**	**Acc. (%)**	**Compl. (%)**	**Acc. (%)**	**Compl. (%)**
GPT-5.2	Baseline	41.1	14.7	44.9	58.4
	MetaClaw (Skills)	**44.0**	**17.1**	**49.1**	**67.5**
Kimi-K2.5	Baseline	21.4	2.0	21.1	18.2
	MetaClaw (Skills)	28.3	2.0	26.9	33.8
	MetaClaw (Full)	**40.6**	**16.5**	**39.6**	**51.9**

4.2 Main Results

Table 1 reports performance on both parts of MetaClaw-Bench for all five model–condition pairs. MetaClaw consistently improves over the respective baselines across both models, both adaptation modes, and both benchmark parts.

MetaClaw improves both models and the full pipeline yields the largest gains. Results are consistent across both benchmark parts. For GPT-5.2, MetaClaw (Skills) raises overall accuracy from 41.1% to 44.0% on Part I and from 44.9% to 49.1% on Part II, with file-check completion rising from 14.7% to 17.1% on Part I and from 58.4% to 67.5% on Part II. For Kimi-K2.5, MetaClaw (Skills) improves accuracy from 21.4% to 28.3% on Part I and from 21.1% to 26.9% on Part II. MetaClaw (Full) yields substantially larger gains: on Part I, accuracy reaches 40.6% and task completion rises 8.25×; on Part II, accuracy reaches 39.6% and file-check completion jumps from 18.2% to 51.9%.

Stronger models benefit less and weaker models benefit more. GPT-5.2 starts from a higher baseline (41.1% vs. 21.4% on Part I), leaving less headroom for skill-driven gains. Kimi-K2.5, by contrast, lacks implicit procedural knowledge that the skill library provides explicitly, so skill injection yields larger returns. Notably, MetaClaw (Full) with Kimi-K2.5 (40.6%) nearly closes the gap with GPT-5.2's baseline (41.1%), demonstrating that the combination of skill injection and gradient-based policy optimization can largely compensate for model capability differences. This pattern suggests MetaClaw is particularly valuable for deploying capable but not state-of-the-art models at production scale.

The full pipeline unlocks end-to-end task completion and skills alone do not. On Part I, MetaClaw (Skills) leaves task completion rates unchanged for both models, confirming that skill injection sharpens partial execution quality without reliably enabling zero-defect outputs under heavy execution demands. MetaClaw (Full) closes this gap: Kimi-K2.5's completion rate jumps from 2.0% to 16.5% (8.25×). On Part II, where file-check tasks are rule-based, skills already drive a substantial completion gain (18.2%→33.8%), and the full pipeline pushes this further to 51.9%, confirming that weight-level optimization provides an additive benefit on top of skill injection regardless of task type.

MetaClaw generalizes to open-ended multi-stage pipelines. Table 2 reports MetaClaw's impact on AutoResearchClaw, an evaluation setting structurally different from MetaClaw-Bench. Using skills-only adaptation (no RL), MetaClaw reduces the stage retry rate by 24.8% and cuts refine cycles by 40.0%. Pipeline completion improves from 18/19 to 19/19 stages, and the composite robustness score rises from 0.714 to 0.845, an 18.3% improvement. These gains are achieved *without any gradient-based policy updates*, demonstrating that MetaClaw's lightweight, zero-downtime skill injection transfers effectively to complex, long-horizon agentic workflows beyond structured CLI tasks.

4.3 Analysis

Per-day accuracy trends. Figure 2 illustrates 3-day rolling accuracies across all conditions. A consistent decline from $> 50\%$ (days 1–10) to $< 30\%$ (days 25–30) confirms the increasing difficulty of MetaClaw-Bench. MetaClaw's advantage peaks in the mid-range (days 11–22), where its failure distillation mechanism effectively masters multi-step procedural tasks, reaching a maximum accuracy

Table 2: MetaClaw (Skills-Only) on AutoResearchClaw, a 23-stage autonomous research pipeline. Skill injection alone yields consistent improvements across all robustness metrics without requiring RL weight updates.

Metric	Baseline	+ MetaClaw (Skills)	Relative Change
Stage retry rate (↓)	10.5%	7.9%	↓ 24.8%
Refine cycle count (↓)	2.0	1.2	↓ 40.0%
Pipeline stage completion (↑)	18 / 19	19 / 19	↑ 5.3%
Composite robustness score (↑)	0.714	0.845	↑ 18.3%

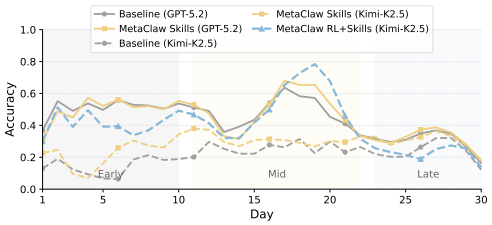

Figure 2: Per-day accuracy over 30 simulated workdays (3-day rolling average). Solid lines: GPT-5.2; dashed lines: Kimi-K2.5. MetaClaw (Full) dominates in the mid phase (day 11–22) before difficulty outpaces accumulated knowledge in late days.

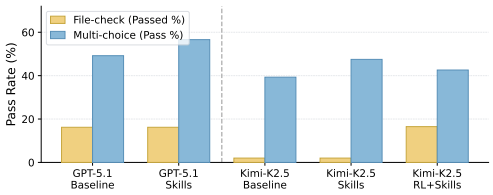

Figure 3: Per-task-type pass rates. File-check (yellow) is unchanged by skills alone but jumps 8.25× under MetaClaw (Full). Multi-choice (blue) improves with skills but slightly decreases under MetaClaw (Full), reflecting a policy shift toward file-execution.

gain of ∼0.8 around day 20. In contrast, performance converges in early days due to task simplicity and in late days due to extreme complexity exceeding current skill/weight capabilities.

Task-type breakdown. Figure 3 decomposes performance by task type, revealing that the two components address different bottlenecks. Skills-only adaptation lifts multi-choice pass rates for both models while leaving file-check completion flat, as procedural knowledge helps reasoning but not execution. MetaClaw (Full) reverses this: Kimi-K2.5's file-check completion rate jumps to match GPT-5.2's baseline, while multi-choice accuracy slightly decreases as the policy shifts toward file-execution behavior during training.

RL training dynamics. Part II analyzes policy evolution. The Kimi-K2.5 completion curve shows a sharp inflection at day 8, with pass rates climbing from ∼9% (days 1–4) to 100% by day 12. This trajectory mirrors a meta-learning update: early days accumulate support trajectories for skill synthesis, while the day 8 inflection marks the point where sufficient gradient signal triggers a LoRA-driven strategy shift. The observed two-phase pattern, i.e., initial skill-driven gains followed by RL-driven acceleration, directly validates MetaClaw's complementary timescale hypothesis.

Skill library analysis. Across the 30-day session, MetaClaw's skill evolver synthesizes skills clustered around three recurring failure categories: (1) *temporal format compliance*, normalizing natural-language time expressions to ISO 8601 format with timezone offsets; (2) *backup-before-modify protocol*, creating `.bak` files before any destructive file operation; and (3) *naming convention adherence*, following date-prefixed file naming patterns. These cross-cutting behavioral heuristics generalize across tasks, explaining why a single failure can yield skills that improve performance on subsequent, structurally different questions.

Cross-domain skill transfer to AutoResearchClaw. The AutoResearchClaw results (Table 2) demonstrate MetaClaw's skill generalization. Without domain-specific tuning, the CLI-trained skill evolver synthesized actionable skills for research automation, reducing refinement cycles by 40%. By distilling knowledge from early pipeline failures (e.g., citation and code validation), the system prevents repeated errors in subsequent runs. This cross-domain transferability and zero-downtime prompt-level injection confirm MetaClaw as a general-purpose continual learning layer for diverse agentic systems.

5 Related Work

Skill-Based and Memory-Augmented Agents. Recent work enhances LLM agents via external memory or skill libraries without weight modification (Shinn et al., 2023; Zhao et al., 2024; Fang et al., 2025; Tang et al., 2025; Ouyang et al., 2025; Chhikara et al., 2025; Liu et al., 2026a; Wang et al., 2024b). Reflexion (Shinn et al., 2023) and Mem0 (Chhikara et al., 2025) utilize episodic buffers or hierarchical retrieval to improve long-horizon performance. Similarly, Voyager (Wang et al.) and ExpeL (Zhao et al., 2024) incrementally build executable code or natural-language skill libraries. However, these methods typically treat skills/memory as static artifacts (Xia et al., 2026), failing to coordinate them with weight-level optimization or account for behavioral shifts. MetaClaw bridges this gap by coupling skill evolution with policy optimization via explicit support-query separation.

Reinforcement Learning for LLM Agents. RLHF (Ouyang et al., 2022) and its variants establish the use of reward signals to fine-tune LLM behavior, and subsequent work applies RL to tool-using and agentic settings (Nakano et al., 2021; Yao et al., 2022). More recently, Recent methods like GRPO (Shao et al., 2024) and DAPO (Yu et al., 2025) offer stable policy gradient training but typically optimize fixed policies against static rewards (Feng et al., 2025; Zheng et al., 2025), lacking inter-rollout context updates. Furthermore, they do not address training triggers or data validity post-behavioral shifts in live deployments. MetaClaw tackles these via opportunistic scheduling and skill versioning.

Continual and Meta-Learning. Meta-learning (Finn et al., 2017; Nichol et al., 2018) optimizes for fast adaptation but typically assumes offline, episode-based settings with simple architectures. Meta-RL extensions (Duan et al., 2016; Rakelly et al., 2019; Rothfuss et al., 2019) achieve rapid adaptation in robotics but rely on fixed task distributions. Conversely, continual learning (Kirkpatrick et al., 2017; Wang et al., 2024a) addresses sequential adaptation without forgetting but often lacks inference-time fast adaptation. Online meta-learning (Finn et al., 2019) relaxes offline assumptions but remains limited to simple representation learning. MetaClaw bridges these gaps by applying meta-learning to non-stationary LLM agent tasks, combining gradient-free fast adaptation (skill synthesis) with gradient-based slow adaptation (RL policy optimization).

6 Conclusion

We presented MetaClaw, a continual meta-learning framework for autonomous agent improvement through real-world usage. By combining fast, inference-time skill injection with slow, gradient-based policy optimization during idle windows, MetaClaw enables dual-timescale adaptation without local GPUs. Experiments on MetaClaw-Bench and AutoResearchClaw demonstrate significant gains in task completion and robustness, proving that skills can generalize to open-ended pipelines even without gradient updates. While idle-window detection currently relies on user configuration, MetaClaw establishes a principled foundation for agents that evolve naturally through deployment.

AI Use Statement

In this work, generative AI was used in two distinct capacities: as a writing aid for the manuscript, and as the object of study in our experiments. 1) We used Gemini-3.6-Flash and GPT-5.2 to polish the language of this paper. These tools were applied only to text that the authors had already drafted; no section, argument, or claim was generated from scratch by an AI tool. 2) LLMs are the subject of this work rather than an authoring aid for it. GPT-5.2 and Kimi-K2.5 serve as the backbone policies under evaluation, and an LLM also acts as the skill evolver that synthesizes behavioral instructions from failure trajectories. These uses are integral to the method and are documented in Sections 3 and 4, with all corresponding prompt templates given in Appendix B. 3) Uses that did not occur. We did not use generative AI tools for research ideation or hypothesis formation, experimental design, literature search and related-work selection. Retrieval- or agent-based AI tools were not used to locate or summarize the cited literature. We have reviewed all AI-assisted work and take full responsibility for the final content of this paper, including all text, claims, and artifacts.

Ethics Statement

The deployment of MetaClaw as a continual learning framework for autonomous agents introduces important ethical considerations regarding data privacy and system safety. Because the Opportunistic Meta-Learning Scheduler (OMLS) monitors personal activity signals, e.g., such as system keyboard inactivity and Google Calendar event occupancy, to trigger background optimization, real-world deployment necessitates explicit user consent and strict data governance. Furthermore, transmitting user trajectories for cloud LoRA fine-tuning poses potential privacy risks, requiring robust anonymization and secure data pipelines. In terms of system safety, allowing an LLM agent autonomous access to a user's CLI and file system inherently risks unintended or destructive operations. To mitigate these risks, MetaClaw's skill-driven adaptation actively synthesizes and enforces behavioral safety heuristics, such as the "backup-before-modify" protocol, to safeguard user environments. Ultimately, we advocate for transparent, user-controlled agentic systems where continuous evolution aligns strictly with user safety and privacy expectations.

Reproducibility Statement

To ensure the full reproducibility, the experimental setup, infrastructure, and prompt templates are thoroughly documented within the paper and its appendices. The evaluation utilizes MetaClaw-Bench we propose, which is divided into a 30-workday simulation (346 questions) for Part I and a 14-workday simulation (588 questions) for Part II. The experiments evaluate GPT-5.2 and Kimi-K2.5 as the backbone LLM policies. Opportunistic policy optimization is implemented via cloud LoRA fine-tuning using the GRPO algorithm. The Opportunistic Meta-Learning Scheduler (OMLS) tracks idle windows by querying the Google Calendar API and polling OS-level input device idle timers, such as ioreg HIDIdleTime on macOS. Appendix B explicitly details all core prompt templates required to replicate the framework's behavior. This includes the exact Agent System Prompts and Identity Context for MetaClaw-Bench Parts I and II, task question templates, the Skill Evolver prompt, the specific formatting used for skill injection, and the instructions for system prompt compression.

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce-style optimization for learning from human feedback in llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12248–12267, 2024.

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

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

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

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*, 2025.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. Online meta-learning. In *International conference on machine learning*, pp. 1920–1930. PMLR, 2019.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Oriol Vinyals, Shakir Mohamed, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Simplemem: Efficient lifelong memory for llm agents. *arXiv preprint arXiv:2601.02553*, 2026a.

Jiaqi Liu, Peng Xia, Siwei Han, Shi Qiu, Letian Zhang, Guiming Chen, Haoqin Tu, Xinyu Yang, , Jiawei Zhou, Hongtu Zhu, Yun Li, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Autoresearchclaw: Fully autonomous research from idea to paper, 2026b. URL https://github.com/aiming-lab/AutoResearchClaw.

Silen Naihin, David Atkinson, Marc Green, Merwane Hamadi, Craig Swift, Douglas Schonholtz, Adam Tauman Kalai, and David Bau. Testing language model agents safely in the wild. *arXiv preprint arXiv:2311.10538*, 2023.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

OpenAI. Introducing gpt-5.2, 2025. https://openai.com/index/introducing-gpt-5-2.

OpenClaw. Openclaw — personal ai assistant, 2026. https://github.com/openclaw/openclaw.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

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

Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International Conference on Machine Learning*, 2019.

Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. Promp: Proximal meta-policy search. In *International Conference on Learning Representations*, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Dawn Song, Chenguang Wang, Nicholas Crispino, Ruoxi Jia, Kyle Montgomery, Yujin Potter, Vincent Siu, and Zhun Wang. Agents in the wild: Safety, security, and beyond. In *ICLR 2026 Workshop Proposals*.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, et al. Agent kb: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*, 2025.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Y Charles, HS Che, Cheng Chen, Guanduo Chen, et al. Kimi k2. 5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

Thinking Machines Lab TML. Tinker is a training api for researchers, 2026. https://thinkingmachines.ai/tinker/.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE transactions on pattern analysis and machine intelligence*, 46(8):5362–5383, 2024a.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024b.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, et al. Evolver: Self-evolving llm agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*, 2025.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. Memevolve: Meta-evolution of agent memory systems. *arXiv preprint arXiv:2512.18746*, 2025a.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. Agentracer: Who is inducing failure in the llm agentic systems? *arXiv preprint arXiv:2509.03312*, 2025b.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026a.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Weinan Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Yutao Qi, et al. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026b.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10495–10516, 2025c.

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

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

APPENDIX

A Case studies **14**

B Prompts and Templates **14**

B.1 Agent System Prompt (MetaClaw-Bench Part I) 14

B.2 Agent Identity Context (MetaClaw-Bench Part II) 15

B.3 Task Question Templates 16

B.4 Skill Evolver Prompt Template 17

B.5 Skill Injection Format 18

B.6 System Prompt Compression Prompt 19

B.7 Part II Implicit Preference Rules 19

Table 3: Representative case studies. Case 1 shows skill-driven fast adaptation (MetaClaw Skills, GPT-5.2); Case 2 shows the full pipeline (MetaClaw (Full), Kimi-K2.5). Both recover from score 0 to score 1.0; the mechanisms differ fundamentally.

	Case 1 MetaClaw (Skills)	**Case 2 MetaClaw (Full)**
Model	GPT-5.2	Kimi-K2.5
Day / Round	Day 19 / Round 4	Day 18 / Round 6
Task type	File-check	File-check
Task instruction		
	Update `sprint8_board.json`: set T-405/T-406 to `"done"`, add `completed_at` fields.	Append a deployment record to `deploy_log.json` with fields `timestamp` (ISO 8601+TZ), `env`, `status`, and `changes`.
Baseline response (score: 0)		
	Reads file; directly overwrites it. Checker detects missing `sprint8_board.json.bak` $\rightarrow$ 0.	Uses field name `date` instead of `timestamp`; omits `changes`. Checker rejects schema $\rightarrow$ 0.
MetaClaw response (score: 1.0)		
	Skill distilled from Day 2: *"Always create `.bak` before modifying (P4)."* Agent writes `sprint8_board.json.bak`, applies targeted patch. Checker passes $\rightarrow$ 1.0.	Skills inject *"use ISO 8601 with timezone offset"*; skills-only Kimi still omits `changes` array $\rightarrow$ 0. After RL: all four fields present, schema valid, backup created $\rightarrow$ 1.0.
Day accuracy (all rounds)		
	Baseline: 43.9% MetaClaw (Skills): 62.1% Δ **+18.2 pp**	Baseline: 8.3% Skills-only: 25.0% MetaClaw (Full): **80.6%**
Key mechanism		
	Skills: one distilled rule generalizes across file types and subsequent days with zero weight update.	**RL:** skills supply declarative format context; weight updates internalize the execution reliability that skill injection alone cannot enforce.

A Case studies

Table 3 contrasts two representative cases that illustrate the distinct contributions of the two MetaClaw mechanisms. In Case 1, a single distilled skill resolves a compliance error with zero weight update. In Case 2, skill injection provides necessary format context but is insufficient alone; weight-level RL updates are required to reliably execute a structurally complex file operation.

B Prompts and Templates

This appendix documents the core prompt templates used in MetaClaw-Bench evaluations and the MetaClaw framework components.

B.1 Agent System Prompt (MetaClaw-Bench Part I)

Part I evaluates agents on OpenClaw CLI tasks via a programmatic rollout loop. The agent receives the following fixed system prompt, which may be replaced by a compressed variant (see Section B.6) after the first session:

SYSTEM_PROMPT — Part I Agent

```
You are an expert CLI agent controlling an OpenClaw installation.
Your goal is to complete the given task by issuing CLI commands
via the run_command tool.
```

```
Guidelines:
- Issue ONE command at a time and carefully read the output
  before proceeding.
- Use 'openclaw status' or similar read commands to inspect
  state before making changes.
- Handle errors: if a command fails, diagnose from the output
  and retry differently.
- When the task is fully complete, call run_command with
  command="done".
- Do NOT ask clarifying questions | act based on the task
  description alone.
```

The single tool exposed to the agent is `run_command`:

run_command Tool Schema

```
{
  "name": "run_command",
  "description": "Execute a CLI command and observe the output.
    When the task is fully complete, call run_command with
    command=\"done\".",
  "parameters": {
    "type": "object",
    "properties": {
      "command": {
        "type": "string",
        "description": "The exact CLI command to run, e.g.
          'openclaw status',
          'openclaw agents add --name bot1 --model gpt-4o',
          'done'"
      }
    },
    "required": ["command"]
  }
}
```

B.2 Agent Identity Context (MetaClaw-Bench Part II)

Part II (CALMB-14) injects the following workspace context files into the agent's session at initialization, defining the agent's role, user profile, and behavioral principles.

IDENTITY.md — Agent Role Definition

```
# Identity
You are MetaClaw Agent, an AI assistant integrated into
the internal toolchain of Orion Tech.

## Role
You serve as Alex Zhang's primary AI assistant for day-to-day
engineering and project management work.

## Context
- Company:   Orion Tech | a B2B SaaS company
- Product:   Project Orion, a project management platform
- Team:      Backend engineering team
- Principal: Alex Zhang, Backend Tech Lead

## Principles
1. Accuracy comes first | double-check facts, values, and
   formats before writing files
2. Be consistent | use the same conventions across all files
3. Be complete | fill in all required fields; no placeholders
```

```
4. Be professional | produce output ready for team use
```

SOUL.md — Core Behavioral Principles

```
## Reliability
Every file you create or modify should be correct and complete.
A half-finished or incorrect file creates more work than it saves.

## Consistency
When you establish a pattern in one file, maintain it across
all related files.

## Attention to Detail
Small errors in data files can cascade into larger problems.
Pay close attention to field names, data types, value formats,
and structural requirements.

## Ownership
Own every assigned task completely. Do not produce output
requiring the user to fix obvious issues.

## Professionalism
All output should meet the standard of work that could be
shared directly with teammates or stakeholders.
```

B.3 Task Question Templates

Each day in MetaClaw-Bench presents two question types. Below are representative examples.

Multi-choice question (Part II, Day 01 / r1):

Multi-Choice Question Template

```
Regarding the source and applicability of the baseline daily
revenue of 4500 yuan/store in the financial model, which of
the following descriptions are consistent with the project
documentation? (Select all correct options)

A. The assumption document labels 4500 yuan as "Source:
   operational baseline data from market research report",
   while the original description in the market research
   report is "median of 86 tier-1 city stores"
B. The rent assumption of 650 yuan/m2 is noted as tier-2 and
   tier-3 city market rates, but the revenue assumption of
   4500 yuan references tier-1 city store data
C. Operations VP Zhang Wei mentioned an estimated daily revenue
   of approximately 2500-3200 yuan, but this is only a
   hypothetical opinion with no direct contradiction
D. The validation memo confirms that 4500 yuan "has been
   cross-validated with operational baseline data", indicating
   the assumption underwent applicability review

Please answer using \bbox{X} or \bbox{X,Y} format.
```

Ground truth: A, B **Scoring:** $\max(0,\ 1-(FP+FN)/n_\text{options})$

File-check question (Part II, Day 01 / r21):

File-Check Question Template

```
Based on the reference documents in day01/, create a decision
log tracking key decisions from the documents. Save as
day01/decision_log_r21.json. Include these fields: title,
created_at, decisions (array with id, date, decision,
rationale, decided_by, review_date).
```

Eval: `python scripts/check_iso8601.py day01/decision_log_r21.json` (exit 0 = pass)
Feedback (incorrect): *Time/date fields must use ISO 8601 with +08:00 timezone: YYYY-MM-DDTHH:MM:SS+08:00.*

Part I task instruction format (real OpenClaw session, `train.jsonl`):

Part I User Instruction Template

```
[Sat 2026-02-21 07:25 EST] I grant you read access to
/Users/jimchen/Documents/openclaw/skills. Locate gog/skill.md
within it. Use skill.md to obtain the instructions for adding
a Google Calendar. Execute the instructions in the isolated
environment to directly add ten meetings to Google Calendar,
starting February 20th, recurring every Friday from
3:30-4:30 PM.

Language requirement: Reply in English only. Keep your
response concise and task-focused.
```

B.4 Skill Evolver Prompt Template

After each session in which failed trajectories are collected, the MetaClaw skill evolver submits the following prompt to synthesize new behavioral skills:

Skill Evolver Prompt (`skill_evolver.py`)

```
You are a skill engineer for an AI assistant trained with RL.
Your job: analyze the failed conversations below and generate
NEW skills that would have prevented those failures.

---
## Failed Conversations

### Failure 1  (reward=0.0)
**Conversation context (last 600 chars):**
```
...[truncated trajectory]
```
**Assistant response (first 500 chars):**
```
[model response]
```
[...up to 6 failures shown...]

---
## Existing Skills (do NOT duplicate any of these)
["skill-name-1", "[category] skill-name-2", ...]

---
## Instructions

Generate **1 to {max_new_skills}** new skills that directly
address the failure patterns observed above. Focus on
```

```
actionable, concrete guidance for future conversations.

Each skill must follow Claude skill format:
- `name`: a lowercase hyphenated slug
- `description`: one sentence | when to trigger this skill
  and what it achieves
- `content`: 6-15 lines of actionable Markdown. Include:
  a heading, numbered steps or bullet points, a concrete
  example or code snippet, and an Anti-pattern section.
- `category`: one of [coding, research, data_analysis,
  security, communication, automation, productivity, agentic]
  or "general" or "common_mistakes"

**Output:** Return ONLY a valid JSON array.

**Example output:**
[
  {
    "name": "dyn-001",
    "description": "Always verify file existence before
      reading or writing.",
    "content": "## Verify File Existence Before Acting\n\n
      1. Check: os.path.exists(path)\n
      2. If missing, ask the user for the correct path.\n
      **Anti-pattern:** Calling open(path) without checking.",
    "category": "coding"
  }
]
```

B.5 Skill Injection Format

Retrieved skills are appended to the agent's system message by `SkillManager.format_for_conversation`:

Skill Injection Block (appended to system prompt)

```
## Active Skills

### backup-before-modify
_Always create a .bak copy before modifying any existing file._

## Backup Before Modify

1. Before editing any file, create a backup:
   cp <filename> <filename>.bak
2. Verify the backup exists before proceeding.
3. Apply all modifications to the original file.

**Anti-pattern:** Overwriting a file without a backup,
leaving no recovery path if the edit is incorrect.

### iso8601-timezone-format
_Use when writing any date/time field to a file._

## ISO 8601 Timestamp with Timezone

Always format timestamps as: YYYY-MM-DDTHH:MM:SS+08:00

- Correct:   2026-03-16T09:30:00+08:00
- Incorrect: 2026-03-16, March 16 at 3pm,
             2026-03-16T09:30:00Z
```

```
**Anti-pattern:** Omitting the timezone offset or using
natural-language date expressions.
```

B.6 System Prompt Compression Prompt

To prevent context overflow during long sessions, MetaClaw compresses OpenClaw's native system prompt using the following instruction:

System Prompt Compression Instruction (`utils.py`)

```
You are compressing an OpenClaw system prompt. Rewrite it to
be under 2000 tokens while preserving behavior. Keep all
critical policy and routing rules:
(1) tool names and their intended usage constraints,
(2) safety and non-delegable prohibitions,
(3) skills-selection rules,
(4) memory recall requirements,
(5) update/config restrictions,
(6) reply-tag/messaging rules,
(7) heartbeat handling rules.
Remove duplicated prose, repeated examples, and decorative
language. Prefer compact bullet sections with short imperative
statements. Do not invent or weaken any rule. Output only
the rewritten system prompt text.
```

B.7 Part II Implicit Preference Rules

MetaClaw-Bench Part II introduces five implicit preference rules progressively across 14 days. These rules are *not* stated in the agent's system prompt; they must be inferred from task feedback and internalized through skill evolution or RL training.

Rule	Category	Requirement	Active from
P1	Timestamp	All date/time fields: `YYYY-MM-DDTHH:MM:SS+08:00`	Day 01
P2	File naming	Output files: `YYYYMMDD_description.ext` (snake_case)	Day 04
P3	Metadata	Every output file must include `created_at`, `author`, `status`	Day 06
P4	Backup	Create `<file>.bak` before modifying any existing file	Day 08
P5	Completion log	Append `[DONE] <timestamp> | <task_id> | <summary>` to `done.log`	Day 10

Table 4: The five implicit preference rules in MetaClaw-Bench Part II, introduced progressively across 6 learning arcs. Each rule is verified by a dedicated automated checker script.