OPENCLAW-SKILL: COLLECTIVE SKILL TREE SEARCH FOR AGENTIC LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Equipping Large Language Model (LLM) agents with effective skills is crucial for solving complex tasks in real-world systems like OpenClaw. In this work, we aim to develop a framework that automatically constructs such reusable skills to enhance LLMs in tool use, multi-step reasoning, and dynamic environment interaction. To this end, we propose Collective Skill Tree Search (CSTS), a novel tree-search-based skill construction framework that constructs structured, diverse and generalizable *tree of skills*. The core idea of CSTS is to leverage collective intelligence to jointly search, identify and compose effective skills via two iterative phases: Collective Skill Node Generation (CSN-Gen) and Collective Skill Node Assessment (CSN-Assess). CSN-Gen exploits collective knowledge from multiple models to explore diverse candidate skills for each subtask, enabling comprehensive skill exploration. CSN-Assess employs multiple models as judges to evaluate and select skill nodes with two scoring mechanisms: (1) collective quality scoring that aggregates independent evaluations to produce a robust estimate of skill effectiveness, and (2) collective transferability scoring that explicitly verifies whether a skill generalizes well across different models. With CSTS, we construct a set of comprehensive *skill trees* along with skill-augmented training data, enabling models to effectively learn and utilize skills. Besides, we introduce Collective Skill Reinforcement Learning, which actively selects multiple relevant skills from the tree to broaden solution-space exploration, avoid being trapped by a single skill and its resulting homogeneous or suboptimal solutions. As a result, our trained model, OpenClaw-Skill, exhibits outstanding agentic capabilities in long-horizon planning, tool use and generalization over challenging benchmarks.

1 INTRODUCTION

Large Language Model (LLM) agents have recently demonstrated strong potential for solving complex real-world tasks through natural language instructions OpenAI (2026a); Anthropic (2025); Comanici et al. (2025); Chiang et al. (2023), particularly in interactive environments such as OpenClaw OpenClaw (2026), where LLM agents Koh et al. (2025); Yao et al. (2023b); Schick et al. (2023) are required to coordinate files, tools, web pages, execution feedback, and intermediate artifacts over multiple steps. To further improve the reliability and generalization of such OpenClaw-like systems, recent studies have introduced the concept of "skills" Xia et al. (2026); Chen et al. (2026); Li et al. (2026), which encapsulate compact and reusable procedural strategies for accomplishing recurring subtasks, such as tool use, verification, and error recovery.

However, current skills Anthropic (2025) are largely handcrafted and require substantial manual design and maintenance, making large-scale skill construction costly, time-consuming, and labor-intensive, thereby significantly limiting scalability.

To address this issue, recent studies Xia et al. (2026); Alzubi et al. (2026); Zhang et al. (2026a) have explored automatic skill construction and maintenance. For example, by distilling LLM agent execution experiences and raw trajectories into reusable procedural knowledge, SkillRL Xia et al. (2026) learns a skill bank from past interactions to facilitate policy improvement while Trace2Skill Ni et al. (2026) consolidates trajectory-local lessons and common patterns into agent skills. In addition, CoEvoSkills Zhang et al. (2026a) enables skill refinement by integrating skill generation with surrogate verification, allowing skills to be iteratively evaluated and revised according to execution feedback.

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.

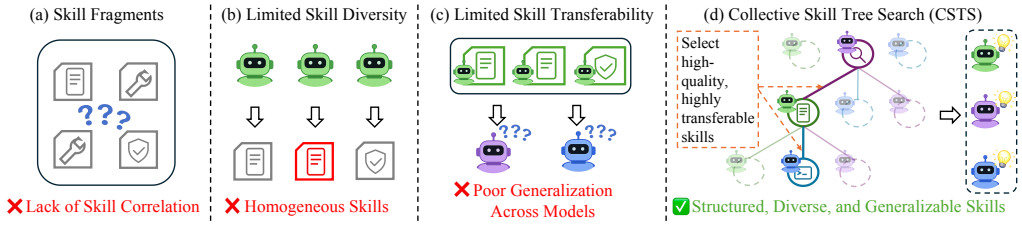

Figure 1: **Motivation.** Current skill construction paradigms generally face several limitations: *(a) Skill Fragmentation*, capturing merely local procedures for isolated subtasks; *(b) Limited Diversity*, suffering from the inherent biases of a single model; and *(c) Poor Transferability*, exhibiting clear performance drops across different LLM backbones. To tackle these challenges, we propose CSTS, a novel tree-search-based skill construction framework that constructs structured, diverse, and generalizable *tree of skills*, empowering LLMs in solving sophisticated tasks in real-world systems.

Despite these advances, most current paradigms for skill construction and maintenance still face several limitations, particularly for long-horizon tasks in real-world interactive environments such as OpenClaw, as illustrated in Figure 1: (1) **Skill Fragmentation**: Current methods often produce fragmented and unstructured skills that merely capture local procedures for individual subtasks, but lack explicit mechanisms to orchestrate skill sequences, handle long-term dependencies, and verify execution across multiple steps; (2) **Limited Skill Diversity**: existing approaches typically construct the skill from a narrow set of trajectories generated by a single model. As a result, the constructed skills are inherently biased toward the problem-solving preferences of that model, limiting their coverage across diverse task types and reasoning strategies. (3) **Limited Skill Transferability**: Skills acquired through current methods often exhibit limited generalization ability. In particular, their performance tends to degrade when transferred to different backbone LLMs.

To tackle these challenges, we propose Collective Skill Tree Search (CSTS), a new tree-search-based skill construction framework that constructs structured, diverse and generalizable skills for LLM agents, enhancing their capability in solving sophisticated real-world tasks. Specifically, CSTS works by leveraging collective intelligence to jointly search, identify and compose effective skills iteratively towards cohesive and reliable "tree of skills". CSTS operates through two iterative phases: *Collective Skill Node Generation* (CSN-Gen) and *Collective Skill Node Assessment* (CSN-Assess). In skill node generation phase, CSN-Gen exploits collective knowledge from multiple models to explore diverse candidate skills for the current subtask, enabling rich and comprehensive skill discovery. In skill node assessment phase, CSN-Assess employs multiple models as judges to evaluate and select skill nodes with strong effectiveness and transferability through two scoring mechanisms. The first is *collective skill quality scoring*, where multiple judges independently assess the skill quality and aggregate their evaluations into a robust score. The second is *collective skill transferability scoring*, which works by measuring whether a skill synthesized by one model can effectively benefit other models, thereby explicitly encouraging the selection of highly generalizable skills.

In this way, our proposed CSTS enables structured, diverse, and transferable skill construction: (1) "Tree of skills" mitigates skill fragmentation. By modeling skills as nodes and subtask-solving procedures as paths, CSTS transforms isolated local skills into structured, dependency-aware skill hierarchies, enabling LLM agents to effectively compose and orchestrate skill sequences across complex multi-stage tasks. (2) The design of collective skill node generation enhances skill diversity. By aggregating candidate skills from multiple heterogeneous models, CSN-Gen explicitly alleviates single-model bias and enriches the skill space with diverse and complementary behavioral patterns, thereby reducing overfitting to specific reasoning styles. (3) The design of collective skill transferability scoring improves skill transferability. By requiring skills synthesized by one model to effectively benefit other models, this mechanism explicitly encourages capturing skills that generalize well across different backbone LLMs, leading to generalizable performance.

Based on our proposed CSTS, we construct a set of comprehensive *skill trees* together with skill-augmented training data for complex tasks under OpenClaw-style systems. By using these skill-augmented training data, we first train our model via Supervised Fine-Tuning (SFT). In addition, we propose Collective Skill Reinforcement Learning (CSRL) to further optimize the model and obtain our final model, termed OpenClaw-Skill. Specifically, for each task, our CSRL actively se-

lects multiple relevant skills from the tree of skills to substantially broaden the exploration of the solution space, thereby preventing the model from being trapped in a single skill and its resulting homogeneous or suboptimal solutions. As a result, the model is allowed to adaptively fit the most appropriate skill among multiple candidate skills, significantly improving the flexibility, robustness, and effectiveness of skill reinforcement learning.

The main contributions of this work are summarized as follows. First, we introduce Tree Search into skill construction for agents and propose Collective Skill Tree Search (CSTS), which leverages collective intelligence to jointly search, identify and compose effective skills iteratively towards cohesive and reliable "tree of skills". Second, we construct a set of comprehensive trees of skills and skill-augmented training data for complex OpenClaw-style tasks, enabling models to learn procedural knowledge from the collective experiences of multiple agents. Third, we introduce Collective Skill Reinforcement Learning (CSRL), which further optimizes the model by actively selecting multiple skills to broaden the exploration of the solution space, thereby preventing the model from being trapped in a single skill and its induced homogeneous and low-quality solutions. Fourth, we develop OpenClaw-Skill, a series of models with outstanding capabilities in long-horizon planning, tool use, error recovery and cross-task generalization, demonstrating superior performance on challenging benchmarks.

2 Related Works

2.1 Large Language Model Agents

Recent advances have transformed large language models Qwen Team (2024; 2025); Yang et al. (2024); DeepSeek-AI (2025; 2024c;b;a); Dai et al. (2024); Ouyang et al. (2022) from static text generators into interactive agents capable of interleaving reasoning, action, and environment feedback Plaat et al. (2025); Luo et al. (2025); Chen et al. (2024); Wang et al. (2023b); Zhang et al. (2025). ReAct Yao et al. (2023b) first established a simple yet influential reasoning–acting paradigm for sequential decision-making and tool use, while WebVoyager He et al. (2024) and OSWorld Xie et al. (2024) pushed LLM agents toward more realistic web and computer-use tasks. This shift has driven the emergence of agent harnesses that expose models to filesystems, browsers, and desktop applications. OpenClaw OpenClaw (2026) exemplifies a move beyond toy environments toward real-world persistent runtimes with messaging interfaces, sessions, tools, and structured workspace state. Building on this paradigm, DeerFlow ByteDance (2026) and Hermes Nous Research (2026) further extend OpenClaw with memory, sandboxed execution, and communication tools. ClawGym Bai et al. (2026) and OpenClaw-RL Wang et al. (2026b) take initial steps toward studying data generation and model training in such Claw-style environments. Nevertheless, how to automatically construct effective skills to guide LLM agents to accomplish complex tasks in real-world systems remains a significant open challenge.

2.2 Skills for Large Language Model Agents

Anthropic first introduced the concept of skills Anthropic (2025) as reusable procedural knowledge that encapsulates instructions and tool-usage patterns, enabling agents to dynamically extend their capabilities at inference time. Recently, a growing body of work has begun to explore the evaluation, generation, and utilization of LLM skills Zhang et al. (2026b); Wang et al. (2023a); Jiang et al. (2026b); Xu & Yan (2026); Yang et al. (2026); Zhang et al. (2026a); Ni et al. (2026); Alzubi et al. (2026); Xia et al. (2026); Chen et al. (2026); Jiang et al. (2026a); Lu et al. (2026). For example, SkillRL Xia et al. (2026) constructs a SkillBank and learns skill invocation policies via reinforcement learning. Trace2Skill Ni et al. (2026) consolidates trajectory-local lessons into skill repositories. Despite recent progress, current skill construction paradigms still suffer from several limitations, especially for long-horizon tasks in real-world environments such as OpenClaw, including skill fragmentation, limited skill diversity, and weak skill transferability. Different from previous approaches, our CSTS introduces a novel tree-search-based skill construction framework that builds structured, diverse, and generalizable *trees of skills*, empowering LLM agents to solve sophisticated tasks in real-world systems.

2.3 TREE SEARCH

Tree search has emerged as an important paradigm for improving decision-making and LLM reasoning Koh et al. (2025); Yao et al. (2023a; 2024); Zhou et al. (2024); Silver et al. (2017); Ye et al. (2021). A representative example is AlphaGo Silver et al. (2017), which combines neural network priors with tree search to enable strategic planning and decision-making, achieving remarkable performance in complex board and video game environments Silver et al. (2017); Ye et al. (2021). Tree Search for Language Model Agents showed that explicit search over action trajectories can substantially improve success rates in realistic web environments Koh et al. (2025). Inspired by these advances, we introduce "tree search" into automatic skill construction and propose CSTS, a framework that decomposes complex tasks into subtasks and leverages collective intelligence to jointly search, identify and compose effective skills iteratively towards cohesive and reliable "tree of skills".

3 METHOD

We first introduce **Collective Skill Tree Search** (CSTS), which constructs a structured, diverse, and transferable space of compositional skills through collective intelligence. CSTS decomposes a complex task into subtasks, generates and assesses multiple candidate skill nodes for each subtask, and progressively constructs a skill path conditioned on preceding skill choices and their execution history. We then present **Collective Skill Reinforcement Learning** (CSRL), which compares trajectories induced by different candidate skills for the same subtask and prefix context. CSRL optimizes the skill-conditioned execution policy and calibrates the initial skill-node scores using subtask-level feedback. The calibrated scores further guide CSTS to revisit retained skill nodes and expand new paths under revised prefix contexts.

3.1 COLLECTIVE SKILL TREE SEARCH

The core idea of CSTS is to use collective intelligence to iteratively search, identify, and compose effective skills into a cohesive and reliable "tree of skills." Specifically, CSTS consists of two iterative phases: (1) **Collective Skill Node Generation** (CSN-Gen) and (2) **Collective Skill Node Assessment** (CSN-Assess). CSN-Gen decomposes a complex task into ordered subtasks and generates candidate skills conditioned on the selected prefix and its execution history. CSN-Assess evaluates their quality and transferability under the same context. Candidates extend their corresponding prefixes to form a layered tree, where each layer corresponds to a subtask and each complete root-to-leaf path defines a skill composition. Finally, (3) **CSTS-SFT** uses the selected skill composition and its verified trajectory for skill-augmented SFT.

Collective Skill Node Generation (CSN-Gen). Given a complex task T, CSN-Gen first decomposes it into an ordered sequence of M subtasks, $T \rightarrow (t_1, \ldots, t_M)$, which defines the depth of the skill tree, with the m-th layer corresponding to subtask t_m. Under a given prefix, $s_{m,n}$ denotes the n-th candidate and $s^\star_m$ denotes the skill selected after assessment to extend the initial path. Each skill describes its applicable context, required inputs, recommended actions, expected outputs, and recovery strategies.

To make the dependencies between successive skills explicit, at layer m let $S^\star_{<m} = (s^\star_1, \ldots, s^\star_{m-1})$ denote the selected skill prefix during initial construction, and let $\tau^\star_{<m} = (\tau^\star_1, \ldots, \tau^\star_{m-1})$ denote their execution trajectories along the selected path. The conditioning context $c_{<m} = (T, S^\star_{<m}, \tau^\star_{<m})$ combines the task, preceding skill choices, and their execution history. At the first layer, no preceding skills or execution trajectories are included.

For each subtask, a set of heterogeneous agent models $\mathcal{M} = \{M_1, \ldots, M_N\}$, with $N > 1$, independently interacts with the environment under the same subtask inputs and prefix context to generate execution trajectories:

$$\tau_{m,n} \sim \pi_{\theta_n}(\cdot \mid t_m, c_{<m}), \qquad n = 1, \ldots, N. \tag{1}$$

For each prefix, candidate rollouts start independently from the same recorded environment state. The same initialization protocol applies to CSN-Assess and CSRL. Each trajectory contains $L_{m,n}$ interaction steps, with $(\psi^\ell_{m,n}, a^\ell_{m,n}, o^\ell_{m,n})$ recording the reasoning state, agent action, and environment feedback at step ℓ, respectively, and $r_{m,n}$ denoting the final execution outcome. Across models,

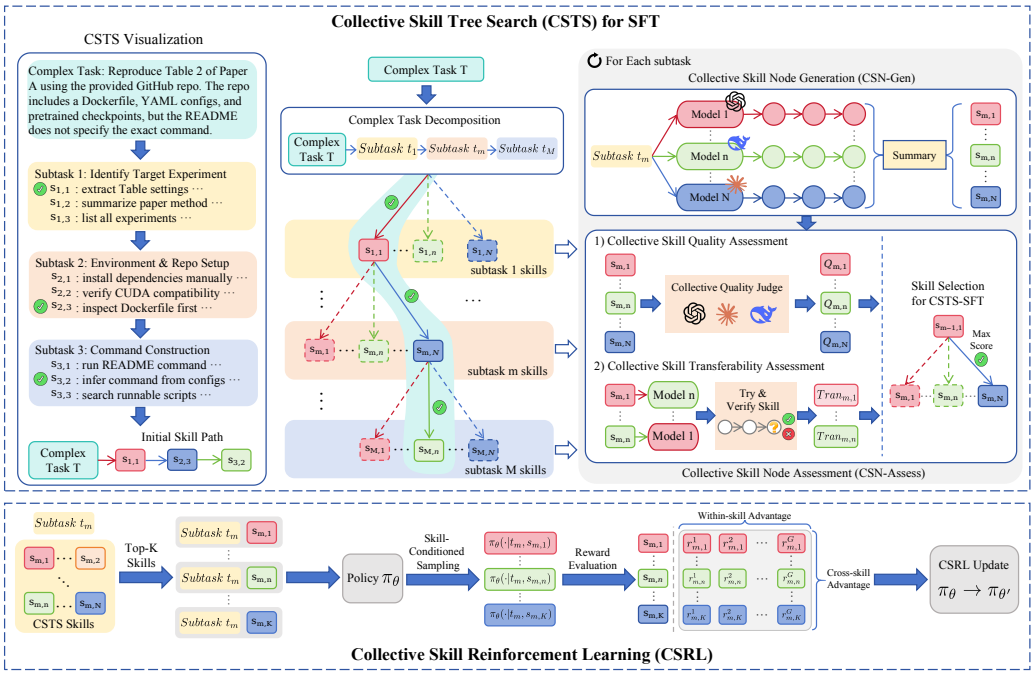

Figure 2: **Overview of OpenClaw-Skill.** CSTS decomposes a task, generates and assesses candidate skills under the selected prefix context, and selects a verified path for SFT. CSRL compares rollouts from the Top-K skills under the same subtask and prefix context, using within- and cross-skill advantages to update the policy. Cross-skill feedback recalibrates node scores and guides subsequent path revision and tree expansion.

these trajectories expose complementary solution routes, failure modes, and verification opportunities, mitigating the procedural bias of any single model. A shared skill synthesizer Φ_{skill} then distills each trajectory into a candidate skill node under the same prefix context:

$$s_{m,n} = \Phi_{\text{skill}}(t_m, c_{<m}, \tau_{m,n}). \tag{2}$$

The resulting candidates form the initial pool $\mathcal{S}_m(c_{<m}) = \{s_{m,1}, \ldots, s_{m,N}\}$, which can be expanded with historical or additional candidates when $K > N$ (Appendix E.3). These candidates form alternative child nodes of the same prefix. When CSRL later revises a prefix, the generation procedure is repeated under the updated context so that subsequent nodes remain compatible with preceding choices and execution outcomes.

Collective Skill Node Assessment (CSN-Assess). After generating candidate skills, CSTS evaluates each node in terms of collective skill quality and transferability. For quality assessment, J judge models independently use a shared rubric to evaluate its clarity, executability, completeness, and relevance to the target subtask, including whether its input requirements and procedures are consistent with the outputs and constraints recorded in $c_{<m}$. Given the normalized score $q^j_{m,n} \in [0,1]$ assigned by judge j, the collective quality score is $Q_{m,n} = \frac{1}{J}\sum_{j=1}^{J} q^j_{m,n}$.

For transferability assessment, skill $s_{m,n}$ distilled from model M_n is shared with the remaining $N-1$ models. Each model M_k, where $k \neq n$, executes the same subtask conditioned on this skill and the shared prefix context, producing $\tilde{\tau}^k_{m,n} \sim \pi_{\theta_k}(\cdot \mid t_m, c_{<m}, s_{m,n})$. Let $r^k_{m,n} \in [0,1]$ denote the normalized verification score of this rollout. The collective transferability score is $\text{Tran}_{m,n} = \frac{1}{N-1}\sum_{k\neq n} r^k_{m,n}$, where the sum runs over the remaining models. Combining both yields

$$\text{Score}_0(s_{m,n} \mid c_{<m}) = Q_{m,n} + \text{Tran}_{m,n}\,. \tag{3}$$

Both terms lie in $[0,1]$ (Appendix D.2); their dependence on the shared prefix context is omitted from the subscripts for readability. This score favors skills that provide reusable procedural guidance across models and are suitable for extending the current skill prefix. It serves as the prior for

subsequent CSRL calibration. CSTS constructs the initial path through greedy, prefix-conditioned tree expansion. At each layer, it retains the Top-K candidate skills, with $1 \leq K \leq |\mathcal{S}_m(c_{<m})|$, and selects the highest-scoring one to extend the initial path:

$$\mathcal{S}_m^K(c_{<m}) = \mathrm{TopK}_{s \in \mathcal{S}_m(c_{<m})} \mathrm{Score}_0(s \mid c_{<m}), \quad s_m^\star = \arg \max_{s \in \mathcal{S}_m^K(c_{<m})} \mathrm{Score}_0(s \mid c_{<m}). \tag{4}$$

A stronger teacher model executes $s_m^\star$ to obtain $\tau_m^\star$ and updates the prefix context to $c_{<m+1} = (T, S_{\leq m}^\star, \tau_{\leq m}^\star)$. This conditions subsequent candidate generation and assessment on preceding skill choices and their observed outcomes, progressively forming the initial path $S_T^\star = (s_1^\star, \ldots, s_M^\star)$. Failed subtask verification terminates construction and discards the incomplete instance. Each prefix context and its environment state are stored alongside the full candidate set and prior scores for subsequent reranking. The retained Top-K skills support CSRL exploration, whose feedback can revise node selection and trigger expansion of alternative branches.

CSTS-SFT. The teacher executions collected during path construction provide demonstrations of how to use the selected skills. Concatenating them yields the full-task trajectory $\tau_T^\star = (\tau_1^\star, \ldots, \tau_M^\star)$ associated with the selected path $S_T^\star$. Only complete trajectories that pass task-level verification are retained as skill-augmented SFT instances $(T, S_T^\star, \tau_T^\star) \in \mathcal{D}_{\mathrm{SFT}}$. The SFT objective is

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\mathbb{E}_{(T, S_T^\star, \tau_T^\star) \sim \mathcal{D}_{\mathrm{SFT}}} \log \pi_\theta(\tau_T^\star \mid T, S_T^\star). \tag{5}$$

CSTS-SFT trains the target policy to ground structured skill guidance in executable action sequences. The loss applies only to teacher-generated tokens, with environment feedback serving as context. Each retained task contributes one verified initial path and its trajectory to $\mathcal{D}_{\mathrm{SFT}}$. These full-task demonstrations are split into training examples for SFT. The alternative skills in each Top-K set $\mathcal{S}_m^K(c_{<m})$ are used by CSRL below. Later path revisions supply new contexts for RL, while $\mathcal{D}_{\mathrm{SFT}}$ remains fixed.

3.2 Collective Skill Reinforcement Learning

CSTS-SFT teaches the target policy to execute one verified path per task, but the relative utility of retained alternatives under that policy remains unknown. Moreover, the initial CSN-Assess scores reflect judgments and executions by other models and may not capture the needs of the evolving target policy. CSRL therefore couples policy optimization over multiple candidate skills with score calibration based on the target policy's execution feedback. By comparing candidates for the same subtask and prefix context, it provides feedback for both policy learning and subsequent skill-tree expansion. For each subtask t_m and its prefix context $c_{<m}$ recorded along the current skill path, the old policy samples G rollouts for each Top-K candidate skill $s_{m,n} \in \mathcal{S}_m^K(c_{<m})$:

$$\tau_{m,n}^g \sim \pi_{\theta_{\mathrm{old}}}(\cdot \mid t_m, c_{<m}, s_{m,n}), \qquad g = 1, \ldots, G. \tag{6}$$

These rollouts form the collective group $\mathcal{B}_m = \{\tau_{m,n}^g \mid s_{m,n} \in \mathcal{S}_m^K(c_{<m}),\ g = 1, \ldots, G\}$. All KG rollouts share the same subtask inputs, prefix context, and initial environment state; rollouts from different prefix contexts are treated as separate groups. Prefix contexts and candidate sets are held fixed during rollout collection and the associated policy updates; path revisions occur between CSRL rounds. A subtask verifier assigns $r_{m,n}^g = R(t_m, c_{<m}, s_{m,n}, \tau_{m,n}^g)$, measuring subtask completion, tool-use correctness, output validity, intermediate verification, and error recovery when applicable.

Within the collective group, each retained skill defines a subgroup of G independent trajectories. For each skill $s_{m,n}$, the mean reward is $\overline{r}_{m,n} = \frac{1}{G} \sum_{g=1}^{G} r_{m,n}^g$. Let μ_m and σ_m denote the mean and standard deviation of rewards over the entire collective group. We then decompose the advantage of rollout $\tau_{m,n}^g$ into two complementary terms:

$$A_{m,n,g}^{\mathrm{within}} = \frac{r_{m,n}^g - \overline{r}_{m,n}}{\sigma_m + \delta}, \qquad A_{m,n}^{\mathrm{cross}} = \frac{\overline{r}_{m,n} - \mu_m}{\sigma_m + \delta}, \tag{7}$$

where $\delta > 0$ is a numerical stabilizer. The within-skill advantage compares a rollout with the mean of its skill-conditioned subgroup, measuring execution quality under a fixed skill. The cross-skill advantage compares that subgroup's mean with the overall mean, measuring the relative utility of the skill under the target policy and shared prefix context. The same $A_{m,n}^{\mathrm{cross}}$ is assigned to all G rollouts conditioned on $s_{m,n}$. The total advantage is $A_{m,n}^g = A_{m,n,g}^{\mathrm{within}} + A_{m,n}^{\mathrm{cross}} = (r_{m,n}^g - \mu_m)/(\sigma_m +$

δ). Thus, the policy update uses a single group-relative advantage over all KG rollouts, while the decomposition also exposes the cross-skill signal used to calibrate node scores.

For rollout $\tau^g_{m,n}$, let $L^g_{m,n}$ be its length and $h^g_{m,n,<\ell}$ the interaction history within the current subtask before action ℓ. The action-level importance ratio is

$$\rho^g_{m,n,\ell}(\theta) = \frac{\pi_\theta(a^g_{m,n,\ell} \mid t_m, c_{<m}, s_{m,n}, h^g_{m,n,<\ell})}{\pi_{\theta_{\text{old}}}(a^g_{m,n,\ell} \mid t_m, c_{<m}, s_{m,n}, h^g_{m,n,<\ell})}. \tag{8}$$

The clipped surrogate is

$$u^g_{m,n,\ell}(\theta) = \min\Big(\rho^g_{m,n,\ell}(\theta)A^g_{m,n}, \text{clip}\Big(\rho^g_{m,n,\ell}(\theta), 1-\epsilon, 1+\epsilon\Big) A^g_{m,n}\Big), \tag{9}$$

where ϵ is the clipping parameter. The CSRL objective is

$$\mathcal{L}_{\text{CSRL}}(\theta) = -\mathbb{E}_{t_m, c_{<m}, \mathcal{B}_m}\left[\frac{1}{|\mathcal{B}_m|}\sum_{\tau^g_{m,n}\in\mathcal{B}_m}\frac{1}{L^g_{m,n}}\sum_{\ell=1}^{L^g_{m,n}} u^g_{m,n,\ell}(\theta)\right]. \tag{10}$$

Only actions in the current-subtask rollouts enter this objective; recorded prefix trajectories provide conditioning information.

CSN-Assess supplies a prior based on collective judgments and cross-model execution, whereas $A^{\text{cross}}_{m,n}$ measures relative skill utility under the evolving target policy. To make node selection responsive to policy learning, we maintain a moving-average utility for each skill under its recorded prefix context: $\widehat{U}(s_{m,n} \mid c_{<m}) \leftarrow \eta_u \widehat{U}(s_{m,n} \mid c_{<m}) + (1-\eta_u)A^{\text{cross}}_{m,n}$, where η_u is the EMA decay coefficient. Combining this policy-dependent utility with the initial score gives $\text{Score}(s_{m,n} \mid c_{<m}) = \text{Score}_0(s_{m,n} \mid c_{<m}) + \widehat{U}(s_{m,n} \mid c_{<m})$. For each recorded prefix context, we keep Score_0 fixed and update $\widehat{U}$ only for skills evaluated under that context. Initializing $\widehat{U}$ to zero preserves the original CSN-Assess ranking in the first CSRL round. After each round, CSTS uses the calibrated scores to reselect $\mathcal{S}^K_m(c_{<m})$ from $\mathcal{S}_m(c_{<m})$. Skills that have not received CSRL feedback retain zero empirical utility.

Feedback-guided tree expansion. Let $S^{(r)}_T$ denote the skill path used in CSRL round r, initialized by $S^{(0)}_T = S^\star_T$. After score calibration, CSTS switches to the highest-scoring retained skill at the earliest layer where it exceeds the current node's score under the same prefix context; if no such node exists, the path remains unchanged. When a switch occurs, the preceding prefix is preserved, and the teacher executes the replacement skill to update its execution history. CSN-Gen and CSN-Assess then reconstruct the subsequent layers under the revised context. Scores and utilities remain associated with their prefix contexts; newly generated nodes receive fresh prior scores and zero initial empirical utility. The revised path is adopted as $S^{(r+1)}_T$ if its trajectory passes subtask- and task-level verification; otherwise, $S^{(r+1)}_T = S^{(r)}_T$. The accepted path then conditions the next CSRL round, closing the loop between policy learning and skill-tree expansion.

4 Experiments

4.1 Implementation Details

For the OpenClaw-style agentic tasks, we use Qwen3-4B and Qwen3-8B (Qwen Team, 2025), and Qwen3.5-4B and Qwen3.5-9B (Qwen Team, 2026a) as backbones. Following prior work (Xia et al., 2026; Wang et al., 2026a), we use Qwen2.5-7B-Instruct (Qwen Team, 2024) for embodied tasks. In CSN-Gen and CSN-Assess, we set the number of models $N = J = 3$: DeepSeek-V4-Pro, GPT-5.4, and Kimi-K2.5, with Claude Opus 4.5 as the skill synthesizer. We construct CSTS-SFT-2K by (1) reverse-synthesizing approximately 200 tasks from public agent trajectories and (2) building skill trees with CSN-Gen and CSN-Assess to generate verified, skill-augmented demonstrations (Appendix D.1). We train each model for 2 epochs with a learning rate of 5×10^{-6} in CSTS-SFT. For CSRL, we set the number of retained skills $K = 2$ and sample $G = 6$ rollouts per skill following standard RL settings (Xia et al., 2026), with a learning rate of 1×10^{-6}. We set the skill utility smoothing coefficient to $\eta_u = 0.99$. All experiments use 8 NVIDIA H100 GPUs.

Table 1: Main results on QWENCLAWBENCH (Qwen Team, 2026c) and PINCHBENCH (Kilo Code, 2026). Nemotron-3-120B is short for Nemotron-3-Super-120B-A12B.

QWENCLAWBENCH		PINCHBENCH				
			23 tasks		123 tasks	
Method	Avg.	Method	Best	Avg.	Best	Avg.
Closed-source models		*Closed-source models*				
Claude Opus 4.6 (Anthropic, 2026a)	59.5	Claude Opus 4.6 (Anthropic, 2026a)	93.3	81.6	88.9	67.5
Qwen3.6-Plus (Qwen Team, 2026b)	57.4	Qwen3.6-Plus (Qwen Team, 2026b)	63.9	63.3	82.8	62.6
GPT-5.4 (OpenAI, 2026b)	56.7	GPT-5.4 (OpenAI, 2026b)	90.5	79.4	–	–
Claude Sonnet 4.6 (Anthropic, 2026b)	60.5	Gemini 3.1 Pro Preview (Google, 2026a)	86.7	77.5	81.7	80.2
MiMo-V2-Pro (Xiaomi MiMo Team, 2026)	56.5	MiMo-V2-Pro (Xiaomi MiMo Team, 2026)	87.4	80.4	–	–
Open-source / open-weight models		*Open-source / open-weight models*				
MiniMax-M2.7 (MiniMax, 2026)	50.5	MiniMax-M2.7 (MiniMax, 2026)	89.8	82.8	–	–
GLM-5.1 (Z.AI, 2026)	58.7	GLM-5.1 (Z.AI, 2026)	82.8	63.2	–	–
Kimi-K2.5 (Moonshot AI, 2026)	51.9	GPT-OSS-120B (Agarwal et al., 2025)	67.1	52.0	47.4	44.5
DeepSeek-V3.2 (DeepSeek-AI, 2025)	50.7	Nemotron-3-120B (Blakeman et al., 2025)	–	–	50.6	38.6
Gemma4-31B (Google, 2026b)	41.7	Llama 4 Maverick (Meta AI, 2025)	46.1	34.8	–	–
Qwen3-4B (Qwen Team, 2025)	7.0	Qwen3-4B (Qwen Team, 2025)	45.2	31.8	22.4	13.6
OpenClaw-Skill-Qwen3-4B	12.8	OpenClaw-Skill-Qwen3-4B	64.5	47.9	31.1	20.8
Qwen3-8B (Qwen Team, 2025)	11.5	Qwen3-8B (Qwen Team, 2025)	49.8	35.4	27.9	18.3
OpenClaw-Skill-Qwen3-8B	15.8	OpenClaw-Skill-Qwen3-8B	64.9	49.2	33.0	22.5
Qwen3.5-4B (Qwen Team, 2026a)	31.5	Qwen3.5-4B (Qwen Team, 2026a)	71.0	55.7	60.9	45.9
OpenClaw-Skill 4B	41.2	OpenClaw-Skill 4B	71.5	56.4	61.4	47.6
Qwen3.5-9B (Qwen Team, 2026a)	34.5	Qwen3.5-9B (Qwen Team, 2026a)	67.5	53.8	61.1	47.1
GRPO (Shao et al., 2024)	36.2	GRPO (Shao et al., 2024)	68.4	54.6	64.3	47.8
Skill-GRPO	38.7	Skill-GRPO	69.8	55.4	65.6	48.5
Skill-SD (Wang et al., 2026a)	40.3	Skill-SD (Wang et al., 2026a)	69.5	56.1	65.5	50.1
SkillRL (Xia et al., 2026)	41.5	SkillRL (Xia et al., 2026)	70.2	55.9	66.1	49.6
OpenClaw-Skill 9B	44.9$^{10.4}$	OpenClaw-Skill 9B	72.8$^{5.3\uparrow}$	58.9$^{5.1\uparrow}$	68.2$^{7.1\uparrow}$	53.6$^{6.5\uparrow}$

4.2 MAIN RESULTS

We conduct extensive experiments to evaluate the effectiveness and cross-domain generalization of OPENCLAW-SKILL. We evaluate OPENCLAW-SKILL on QWENCLAWBENCH (Qwen Team, 2026c), PINCHBENCH (Kilo Code, 2026), and ALFWORLD (Shridhar et al., 2021), covering OpenClaw-style and embodied agentic tasks. We conduct five independent evaluation runs on the OpenClaw benchmarks and three on ALFWORLD, and report mean scores; evaluation and reporting details are provided in Appendix C.1. We evaluate across Qwen models of different scales and further compare Qwen3.5-9B with vanilla GRPO and skill-based methods, including Skill-GRPO, which conditions policy rollouts on skills, Skill-SD (Wang et al., 2026a), and SkillRL (Xia et al., 2026).

Results on OpenClaw-Style Tasks. We first evaluate OPENCLAW-SKILL on QWENCLAWBENCH and PINCHBENCH, as shown in Table 1. Compared with the corresponding Qwen backbone models, OPENCLAW-SKILL consistently achieves substantial gains across different model scales. In particular, OPENCLAW-SKILL-9B achieves an overall score of 44.9 on QWENCLAWBENCH, significantly outperforming vanilla GRPO and skill-based methods, including Skill-GRPO, Skill-SD, and SkillRL. Similar improvements are observed on both the 23-task and expanded 123-task settings of PINCHBENCH, demonstrating stronger execution robustness across diverse agent tasks. These results indicate that the gains come not merely from incorporating skills as additional context, but from constructing structured skill paths with CSTS and optimizing multiple skill-conditioned rollout groups with CSRL, which better handle complex tasks involving multi-step tool use, file operations and web interaction.

Results on Embodied Household Tasks. We further evaluate OPENCLAW-SKILL on the embodied household benchmark ALFWORLD. As shown in Table 2, OPENCLAW-SKILL-7B achieves a macro-average success rate of 92.5%, outperforming GRPO, Skill-GRPO, Skill-SD, and SkillRL by

Table 2: Success rates on ALFWORLD (Shridhar et al., 2021).

Method	ALFWORLD						
	Pick	Look	Clean	Heat	Cool	Pick2	Avg.
Qwen2.5-7B-Instruct	33.4	21.6	19.3	6.9	2.8	3.2	14.5
GRPO	84.2	67.1	79.6	76.7	62.3	65.4	72.6
Skill-GRPO	90.5	84.2	87.7	82.0	68.9	71.2	80.8
Skill-SD	93.9	93.8	90.9	100.0	69.2	68.4	85.1
SkillRL	97.9	71.4	90.0	90.0	95.5	87.5	89.9
OPENCLAW-SKILL-7B	98.7	94.3	93.1	82.0	94.7	92.1	**92.5**

Table 3: Ablation study of OPENCLAW-SKILL on QWENCLAWBENCH (Qwen Team, 2026c).

Setting	CSTS-SFT		CSRL	Avg.
	CSN-Gen	CSN-Assess		
Vanilla skill gen.	–	–	–	35.3
Vanilla multi-model skill gen.	–	–	–	35.6
CSN-Gen	✓	–	–	39.8
CSN-Gen + CSN-Assess	✓	✓	–	42.8
CSTS-SFT + GRPO	✓	✓	–	43.3
OPENCLAW-SKILL	✓	✓	✓	**44.9**

19.9, 11.7, 7.4, and 2.6 percentage points, respectively. These gains suggest that structured skill guidance is also useful for embodied tasks, where valid actions depend on the current environment and object states. CSTS decomposes long-horizon goals into ordered subgoals and organizes candidate skills into a tree conditioned on preceding execution, making the dependencies between subgoals explicit. The resulting skill paths provide concise guidance on applicable conditions, actions, and expected outcomes, helping the policy respect these dependencies and choose appropriate next steps as the environment state changes.

4.3 ABLATION STUDY

We conduct ablation studies on QWENCLAWBENCH using Qwen3.5-9B as the backbone to examine the contributions of skill generation, assessment, and reinforcement learning (Table 3). We first compare CSTS-SFT, comprising CSN-Gen and CSN-Assess, with vanilla skill generation. SFT with vanilla single-model skill generation and SFT with vanilla multi-model skill generation score 35.3 and 35.6, respectively, whereas CSTS-SFT reaches 42.8, gains of 7.5 and 7.2 points. This improvement is tied to structured tree construction: CSN-Gen decomposes a task into ordered subtasks and synthesizes prefix-conditioned candidate nodes from the execution trajectories of heterogeneous models, capturing complementary actions and recovery strategies across the candidate set. CSN-Assess then combines collective quality judgments with cross-model transferability tests for a more reliable ranking of nodes and selects a coherent skill path through the tree. With matched initialization and rollout budgets, CSRL improves over CSTS-SFT + GRPO from 43.3 to 44.9 (Appendix E.1). CSRL compares rollouts from the Top-K skills under the same subtask and prefix context; cross-skill feedback recalibrates node scores, and revised paths condition later policy updates. This creates bidirectional optimization between the skill tree and policy. The component ablations show that CSN-Gen, CSN-Assess, and CSRL yield cumulative gains when progressively combined. Further baseline discussion, the transferability ablation, and hyperparameter studies appear in Appendices E.1, E.2, and E.3, respectively.

5 CONCLUSION

In this paper, we present OPENCLAW-SKILL, a framework with automatic skill construction and skill-augmented training to enhance LLM agents on complex tasks in real-world systems such as OpenClaw. At the core of our framework is Collective Skill Tree Search (CSTS), a novel tree-search-based skill construction framework that builds structured, diverse, and generalizable skills for LLM agents, improving their ability to solve sophisticated real-world tasks. CSTS operates through two iterative phases: *Collective Skill Node Generation* and *Collective Skill Node Assessment*. Together, these phases leverage collective intelligence to iteratively search, identify, and compose effective skills into cohesive and reliable *trees of skills*. Furthermore, we introduce *Collective Skill Reinforcement Learning*, which further optimizes the policy over skill-conditioned rollout groups and encourages the model to favor more effective procedural strategies. Extensive experiments on QWENCLAWBENCH, PINCHBENCH, and ALFWORLD, together with ablation studies, demonstrate that OPENCLAW-SKILL consistently achieves strong performance on diverse long-horizon tasks involving tool use, file operations, web interaction, and execution feedback.

REFERENCES

S. Agarwal et al. gpt-oss-120b & gpt-oss-20b model card, 2025.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems, 2026.

Anthropic. Equipping agents for the real world with agent skills. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills, October 2025. Anthropic Engineering Blog.

Anthropic. Claude-4-sonnet, 2025.

Anthropic. Introducing claude opus 4.6. https://www.anthropic.com/news/claude-opus-4-6, 2026a. Accessed: 2026-05-07.

Anthropic. Introducing claude sonnet 4.6. https://www.anthropic.com/news/claude-sonnet-4-6, 2026b. Accessed: 2026-07-28.

Fei Bai, Huatong Song, Shuang Sun, Daixuan Cheng, Yike Yang, Chuan Hao, Renyuan Li, Feng Chang, Yuan Wei, Ran Tao, Bryan Dai, Jian Yang, and Wayne Xin Zhao. Clawgym: A scalable framework for building effective claw agents, 2026.

A. Blakeman et al. NVIDIA Nemotron 3: Efficient and open intelligence, 2025.

ByteDance. Deerflow: Deep exploration and efficient research flow, 2026. URL https://github.com/bytedance/deer-flow. Official repository, accessed 2026-05-07.

Shiqi Chen, Jingze Gai, Ruochen Zhou, Jinghan Zhang, Tongyao Zhu, Junlong Li, Kangrui Wang, Zihan Wang, Zhengyu Chen, Klara Kaleb, Ning Miao, Siyang Gao, Cong Lu, Manling Li, Junxian He, and Yee Whye Teh. Skillcraft: Can llm agents learn to use tools skillfully?, 2026.

Shuaihang Chen, Yuanxing Liu, Wei Han, Weinan Zhang, and Ting Liu. A survey on llm-based multi-agent system: Recent advances and new frontiers in application, 2024.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6, 2023.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025.

Damai Dai et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models, 2024.

DeepSeek-AI. Deepseek llm: Scaling open-source language models with longtermism, 2024a.

DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024b.

DeepSeek-AI. Deepseek-v3 technical report, 2024c.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

DeepSeek-AI. DeepSeek-V3.2 release. https://api-docs.deepseek.com/news/news251201, 2025. Accessed: 2026-05-07.

Google. Gemini 3.1 Pro: A smarter model for your most complex tasks. https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/, 2026a. Accessed: 2026-05-07.

Google. Gemma 4: Byte for byte, the most capable open models. https://blog.google/innovation-and-ai/technology/developers-tools/gemma-4/, 2026b. Accessed: 2026-05-07.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6864–6890, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.371. URL https://aclanthology.org/2024.acl-long.371/.

Guanyu Jiang, Zhaochen Su, Xiaoye Qu, and Yi R. Fung. Xskill: Continual learning from experience and skills in multimodal agents, 2026a.

Y. Jiang et al. Sok: Agentic skills – beyond tool use in llm agents, 2026b.

Kilo Code. PinchBench: OpenClaw LLM Model Benchmarking. https://pinchbench.com/, 2026. Accessed: 2026-05-07.

Jing Yu Koh, Stephen Marcus McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=QF0N3x2XVm.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Binxu Li, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, Junwei He, Yizhuo He, Liqiang Jing, Luyang Kong, Xin Lan, Jiachen Li, Songlin Li, Yijiang Li, Yueqian Lin, Xinyi Liu, Xuanqing Liu, Haoran Lyu, Ze Ma, Bowei Wang, Runhui Wang, Tianyu Wang, Wengao Ye, Yue Zhang, Hanwen Xing, Yiqi Xue, Steven Dillmann, and Han-chung Lee. Skillsbench: Benchmarking how well agent skills work across diverse tasks, 2026.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Skill0: In-context agentic reinforcement learning for skill internalization, 2026.

Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, et al. Large language model agent: A survey on methodology, applications and challenges, 2025.

Meta AI. The Llama 4 herd: The beginning of a new era of natively multimodal ai innovation. https://ai.meta.com/blog/llama-4-multimodal-intelligence/, 2025. Accessed: 2026-05-07.

MiniMax. MiniMax M2.7: Early echoes of self-evolution. https://www.minimax.io/news/minimax-m27-en, 2026. Accessed: 2026-05-07.

Moonshot AI. Kimi K2.5: Visual agentic intelligence. https://www.kimi.com/blog/kimi-k2-5, 2026. Accessed: 2026-05-07.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills, 2026.

Nous Research. Hermes agent, 2026. URL https://github.com/NousResearch/hermes-agent. Official repository, accessed 2026-05-07.

OpenAI. Openai gpt-5 system card, 2026a.

OpenAI. Introducing GPT-5.4. https://openai.com/index/introducing-gpt-5-4/, 2026b. Accessed: 2026-05-07.

OpenClaw. Openclaw, 2026. URL https://docs.openclaw.ai/. Official documentation and repository, accessed 2026-05-07.

Long Ouyang, Jeffrey Wu, Xu Jiang, et al. Training language models to follow instructions with human feedback, 2022.

Aske Plaat, Max van Duijn, Niki van Stein, Mike Preuss, Peter van der Putten, and Kees Joost Batenburg. Agentic large language models, a survey, 2025.

Qwen Team. Qwen2.5 technical report, 2024.

Qwen Team. Qwen3 technical report, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, 2026a. Accessed: 2026-05-07.

Qwen Team. Qwen3.6-Plus: Towards real world agents. https://qwen.ai/blog?id=qwen3.6, 2026b. Accessed: 2026-05-07.

Qwen Team. QwenClawBench: Real-user-distribution benchmark for openclaw agents, April 2026c. URL github.com/SKYLENAGE-AI/QwenClawBench.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551, 2023. URL https://openreview.net/forum?id=Yacmpz84TH.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2010.03768.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Guanzhi Wang, Yuqi Xie, Wenhu Jiang, et al. Voyager: An open-ended embodied agent with large language models, 2023a.

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, and Honggang Qi. Skill-SD: Skill-conditioned self-distillation for multi-turn LLM agents, 2026a.

Lei Wang, Chen Ma, Xue Feng, et al. A survey on large language model based autonomous agents, 2023b.

Yinjie Wang, Xuyang Chen, Xiaolong Jin, Mengdi Wang, and Ling Yang. Openclaw-rl: Train any agent simply by talking, 2026b.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning, 2026.

Xiaomi MiMo Team. MiMo-V2-Pro. https://mimo.xiaomi.com/mimo-v2-pro, 2026. Accessed: 2026-05-07.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, pp. 52040–52094, 2024. doi: 10.52202/079017-1650. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5d413e48f84dc61244b6be550f1cd8f5-Abstract-Datasets_and_Benchmarks_Track.html. Datasets and Benchmarks Track.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward, 2026.

An Yang et al. Qwen2 technical report, 2024.

Yutao Yang, Junsong Li, Qianjun Pan, et al. Autoskill: Experience-driven lifelong learning via skill self-evolution, 2026.

Huanjin Yao, Jiaxing Huang, Wenhao Wu, Jingyi Zhang, Yibo Wang, Shunyu Liu, Yingjie Wang, Yuxin Song, Haocheng Feng, Li Shen, and Dacheng Tao. Mulberry: Empowering MLLM with o1-like reasoning and reflection via collective monte carlo tree search, 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10–16, 2023*, volume 36. Neural Information Processing Systems Foundation, 2023a. URL https://papers.nips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.

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*, 2023b. URL https://openreview.net/forum?id=WE_vluYUL-X.

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *Advances in neural information processing systems*, 34:25476–25488, 2021.

Z.AI. GLM-5.1: Towards long-horizon tasks. https://z.ai/blog/glm-5.1, 2026. Accessed: 2026-05-07.

Guibin Zhang, Hejia Geng, Xiaohang Yu, et al. The landscape of agentic reinforcement learning for llms: A survey, 2025.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. Coevoskills: Self-evolving agent skills via co-evolutionary verification, 2026a.

Ziao Zhang, Kou Shi, Shiting Huang, et al. Skillflow: Benchmarking lifelong skill discovery and evolution for autonomous agents, 2026b.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62138–62160. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

APPENDIX

A AI USE STATEMENT

Generative AI tools assisted with drafting and polishing parts of this manuscript and auxiliary code edits. LLMs were also used to generate and assess skills and to construct the synthetic training dataset CSTS-SFT-2K (Appendix D.1).

B BENCHMARK DETAILS

QwenClawBench. QWENCLAWBENCH (Qwen Team, 2026c) is a real-world agent benchmark with 100 tasks spanning eight categories. All scores are reported as percentages averaged over tasks in each category. The category abbreviations are defined as follows: WAO denotes Workflow and Agent Orchestration (21 tasks), SOA denotes System Operations and Administration (20 tasks), KMM denotes Knowledge and Memory Management (15 tasks), FQT denotes Finance and Quantitative Trading (10 tasks), DAM denotes Data Analysis and Modeling (10 tasks), SVM denotes Security and Vulnerability Management (9 tasks), CS denotes Communication and Scheduling (8 tasks), and RIR denotes Research and Information Retrieval (7 tasks) Qwen Team (2026c).

PinchBench. PINCHBENCH (Kilo Code, 2026) is a real-world benchmark for evaluating LLMs operating as OpenClaw coding agents. It comprises practical tasks involving tool use, file operations, web research, coding, writing, information synthesis, email handling, memory utilization, and skill-based execution. Each task specifies a user prompt, expected behavior, grading criteria, automated checks, and, where appropriate, an LLM-as-a-judge rubric. PINCHBENCH combines automated, LLM-based, and hybrid evaluation to assess both objectively verifiable outcomes and qualitative task completion. The early public benchmark contains 23 tasks spanning calendar creation, stock research, blog writing, document summarization, API workflows, skill installation, inbox triage, market research, spreadsheet summarization, PDF comprehension, and memory persistence. We additionally evaluate on a fixed expanded benchmark comprising 123 tasks. We report both the best and average success rates on the 23-task and 123-task benchmarks.

ALFWorld. ALFWORLD (Shridhar et al., 2021) is a benchmark that aligns text-based environments with embodied household tasks. An agent must understand natural-language goals, explore the environment, manipulate objects, and execute long-horizon action sequences to complete each task. We evaluate six categories: picking and placing an object (Pick), examining an object under a light (Look), cleaning and placing an object (Clean), heating and placing an object (Heat), cooling and placing an object (Cool), and picking and placing two objects (Pick2). We report the success rate for each category and their macro average.

C EVALUATION PROTOCOL AND DOMAIN RESULTS

C.1 EVALUATION PROTOCOL

Repeated evaluation. For evaluations conducted in this work, we perform five independent runs on the OpenClaw benchmarks, QWENCLAWBENCH and PINCHBENCH, and three on ALFWORLD. Each run evaluates the same task set for the corresponding benchmark. We allocate more runs to the OpenClaw benchmarks because of their greater run-to-run variability: differences in early actions can propagate through subsequent tool interactions and affect final outcomes. Averaging over more runs reduces the influence of this variability on the reported scores.

QwenClawBench reporting. Following the official reporting convention (Qwen Team, 2026c), we report only Avg. in Table 1. For each of our five runs, we average the task scores over all 100 tasks; Avg. is the arithmetic mean of these five benchmark scores. Table 4 provides the results for all eight task categories. For our evaluations, each category score is also averaged across the five runs, and Overall corresponds to Avg. in the main results table.

Table 4: Domain-level results on QWENCLAWBENCH (Qwen Team, 2026c).

Method	QWENCLAWBENCH								
	WAO	SOA	KMM	FQT	DAM	SVM	CS	RIR	Overall
Closed-source models									
Claude Opus 4.6 (Anthropic, 2026a)	57.8	57.3	55.6	48.5	54.3	76.8	67.3	71.0	59.5
Qwen3.6-Plus (Qwen Team, 2026b)	56.2	52.2	55.1	46.6	56.1	76.9	62.6	67.5	57.4
GPT-5.4 (OpenAI, 2026b)	55.2	54.3	57.9	38.4	52.6	70.6	71.1	63.7	56.7
Claude Sonnet 4.6 (Anthropic, 2026b)	61.7	56.2	59.3	46.3	58.6	75.7	63.6	71.3	60.5
MiMo-V2-Pro (Xiaomi MiMo Team, 2026)	57.1	49.1	57.3	46.4	53.4	74.0	60.6	64.9	56.5
Open-source / open-weight models									
MiniMax-M2.7 (MiniMax, 2026)	46.6	46.5	53.7	40.6	54.2	62.7	51.7	58.7	50.5
GLM-5.1 (Z.AI, 2026)	63.1	53.3	58.8	39.4	49.4	75.7	66.9	70.8	58.7
Kimi-K2.5 (Moonshot AI, 2026)	51.6	51.5	50.7	39.3	46.3	62.2	60.2	59.8	51.9
DeepSeek-V3.2 (DeepSeek-AI, 2025)	45.5	46.8	49.7	35.6	57.1	63.8	63.7	60.7	50.7
Gemma4-31B (Google, 2026b)	–	–	–	–	–	–	–	–	41.7
Qwen3-4B (Qwen Team, 2025)	6.6	10.8	7.1	0.8	7.2	2.5	6.3	13.2	7.0
OpenClaw-Skill-Qwen3-4B	13.5	10.2	21.9	2.9	10.1	11.3	23.3	7.2	12.8$^{5.8\uparrow}$
Qwen3-8B (Qwen Team, 2025)	17.3	7.9	13.0	2.0	14.2	13.1	9.3	10.7	11.5
OpenClaw-Skill-Qwen3-8B	12.7	15.9	18.2	5.8	9.4	22.0	34.3	13.3	15.8$^{4.3\uparrow}$
Qwen3.5-4B (Qwen Team, 2026a)	24.3	28.8	34.2	15.4	34.3	48.3	55.4	24.4	31.5
OpenClaw-Skill 4B	35.8	46.1	46.7	7.8	33.2	51.5	61.0	54.1	41.2$^{9.7\uparrow}$
Qwen3.5-9B (Qwen Team, 2026a)	26.6	39.9	58.8	15.4	19.9	33.2	30.2	44.8	34.5
GRPO (Shao et al., 2024)	28.3	42.1	50.4	11.7	23.1	48.2	40.8	45.5	36.2
Skill-GRPO	33.1	40.5	49.8	12.5	24.4	57.8	50.3	46.6	38.7
Skill-SD (Wang et al., 2026a)	32.6	41.4	48.8	13.4	25.7	58.7	69.2	44.7	40.3
SkillRL (Xia et al., 2026)	34.4	40.8	50.6	14.8	27.1	60.5	70.5	46.5	41.5
OpenClaw-Skill 9B	32.3	43.3	60.2	14.2	28.4	70.9	78.4	50.1	44.9$^{10.4\uparrow}$

PinchBench reporting. Following the official leaderboard's reporting convention (Kilo Code, 2026), we report both Best and Avg. for the 23-task and 123-task settings in Table 1. For our evaluations, Avg. is the arithmetic mean of the five benchmark scores, while Best is the highest benchmark score among the five runs. Both statistics are computed separately for each task set.

ALFWorld reporting. For our ALFWORLD evaluations, we average the success rate for each task category across the three independent runs. The Avg. score for our method in Table 2 is the macro average of the resulting success rates across the six task categories.

C.2 DOMAIN RESULTS ON QWENCLAWBENCH

Table 4 reports the domain-level results on QWENCLAWBENCH. Where domain-level scores are available, we aggregate task scores separately within each of the benchmark's eight domains. Thus, each domain value is the percentage score averaged over the tasks belonging to that domain, while Overall is computed over all 100 tasks. Baselines for which only an overall score is available are marked with dashes in the domain columns.

Across all four evaluated Qwen backbones, OPENCLAW-SKILL improves the overall score: from 7.0 to 12.8 for Qwen3-4B, from 11.5 to 15.8 for Qwen3-8B, from 31.5 to 41.2 for Qwen3.5-4B, and from 34.5 to 44.9 for Qwen3.5-9B. The OPENCLAW-SKILL 9B model also outperforms GRPO, Skill-GRPO, Skill-SD, and SkillRL, whose overall scores are 36.2, 38.7, 40.3, and 41.5, respectively. These results demonstrate the effectiveness of structured skill construction in enhancing agent performance on tasks requiring long-horizon tool use, procedural execution, intermediate-state verification, and recovery from execution errors.

C.3 GENERALIZATION TO GEMMA AND MINICPM

To assess generalization beyond the Qwen model family, we evaluate OPENCLAW-SKILL against the original models and SkillRL using two additional backbones: Gemma4-E4B-it[1] and MiniCPM4-8B[2]. Both additional checkpoints have approximately 8B total parameters. We evaluate the same 100 QWENCLAWBENCH tasks following the protocol in Appendix C.1. As shown in Table 5, OPENCLAW-SKILL raises the overall score from 29.6 to 37.8 on Gemma4-E4B-it and from 8.8 to 13.5 on MiniCPM4-8B. It also exceeds SkillRL by 4.4 and 3.3 points, respectively, showing that its gains extend beyond Qwen backbones.

Table 5: Generalization to Gemma and MiniCPM on QWENCLAWBENCH. Overall denotes the score across all benchmark tasks.

Method	Overall
Gemma4-E4B-it	29.6
SkillRL (Gemma4-E4B-it)	33.4
OPENCLAW-SKILL (Gemma4-E4B-it)	37.8
MiniCPM4-8B	8.8
SkillRL (MiniCPM4-8B)	10.2
OPENCLAW-SKILL (MiniCPM4-8B)	13.5

D ADDITIONAL EXPERIMENTAL DETAILS

D.1 TRAINING DATA CONSTRUCTION

We use public agent trajectories from ClawGym-Trajectory[3], Nex Agent-SFT[4], and SWE-agent-trajectories[5] as the starting point for constructing CSTS-SFT-2K. These sources cover OpenClaw tool interactions, general agent and coding workflows, and software issue resolution, respectively. From their recorded actions, observations, and outcomes, we reverse-synthesize approximately 200 training tasks, each specifying a task goal, an execution workflow, and verification criteria. We then decompose each reconstructed task into ordered subtasks and apply CSTS to build a skill tree. At each prefix, CSN-Gen obtains execution trajectories from heterogeneous models and distills them into candidate skill nodes conditioned on preceding skill choices and execution history. CSN-Assess evaluates the candidates through collective quality judgment and cross-model transferability estimation, selecting the highest-scoring node to extend the path while retaining alternatives for CSRL. GPT-5.4 executes the selected skill under the same prefix context, and the verified subtask executions are concatenated into a full-task demonstration. We retain demonstrations that pass task-level verification and split them into approximately 2,000 skill-augmented training examples, forming CSTS-SFT-2K. Here, 2K denotes the number of training examples after trajectory splitting.

[1] https://huggingface.co/google/gemma-4-E4B-it
[2] https://huggingface.co/openbmb/MiniCPM4-8B
[3] https://huggingface.co/datasets/RUC-AIBOX/ClawGym-Trajectory
[4] https://huggingface.co/datasets/nex-agi/agent-sft
[5] https://huggingface.co/datasets/nebius/SWE-agent-trajectories

D.2 HYPERPARAMETERS

Table 6: Hyperparameters for OPENCLAW-SKILL.

Hyperparameter	Value
CSTS	
CSN-Gen models N	3
CSN-Assess models J	3
Heterogeneous models	DeepSeek-V4-Pro, GPT-5.4, Kimi-K2.5
Skill synthesizer	Claude Opus 4.5
CSTS-SFT	
Trajectory teacher model	GPT-5.4
Learning rate	5×10^{-6}
Epochs	2
CSRL	
RL batch size	32
Retained skills per subtask K	2
Rollouts per skill G	6
Max prompt length	32,768
Max response length	8,192
Learning rate	1×10^{-6}
Skill utility smoothing coefficient η_u	0.99

Skill scoring and normalization. Each judge receives the candidate skill, target subtask, and prefix context together with a shared skill-scoring rubric. The rubric assesses clarity, executability, completeness, and relevance to the subtask, including whether the skill's input requirements and procedures are consistent with the preceding execution history. Each raw quality score is divided by the rubric's maximum score to obtain $q^j_{m,n} \in [0,1]$, and $Q_{m,n}$ is the average of these normalized scores across judges. For transferability, each cross-model execution is evaluated against the subtask verification rubric; its raw verification score is divided by the corresponding maximum to obtain $r^k_{m,n} \in [0,1]$. Averaging these scores over the other $N-1$ models yields $\mathrm{Tran}_{m,n} \in [0,1]$. Thus, $Q_{m,n}$ and $\mathrm{Tran}_{m,n}$ share a common scale and contribute equally to the initial node score $\mathrm{Score}_0 = Q_{m,n} + \mathrm{Tran}_{m,n} \in [0,2]$, which serves as the fixed prior for subsequent CSRL feedback calibration.

E DISCUSSION

E.1 COMPARISON WITH SKILL DISTILLATION

Table 3 compares OPENCLAW-SKILL with skill distillation baselines. SFT with skills generated by a single model yields only limited gains, from 34.5 to 35.3, while replacing them with independently generated skills from multiple models provides only marginal additional improvement, reaching 35.6. This suggests that directly providing unstructured and potentially conflicting skills to the teacher model is insufficient for generating high-quality skill-conditioned trajectories for long-horizon tasks. Table 7 compares CSTS-SFT + GRPO and CSRL from the same CSTS-SFT checkpoint. Single-skill GRPO uses $K = 1, G = 12$, whereas CSRL uses $K = 2, G = 6$; both use 12 rollouts per group. Under $K = 1$, the cross-skill advantage is zero; with the utility initialized to zero, node scores and skill paths remain fixed. This control compares single-skill GRPO with the combined multi-skill sampling and feedback mechanism of CSRL under the same supervised initialization and rollout budget. CSTS-SFT + GRPO achieves 43.3, a gain of 0.5 points over CSTS-SFT. CSRL reaches 44.9, exceeding this single-skill control by 1.6 points and CSTS-SFT by 2.1 points. The component ablation in Table 3 further examines the contributions of CSN-Gen, CSN-Assess, and CSRL.

Table 7: Skill-conditioned reinforcement learning on QWENCLAWBENCH with Qwen3.5-9B.

Setting	K	G	Avg.
CSTS-SFT (no RL)	–	–	42.8
CSTS-SFT + GRPO	1	12	43.3
CSTS-SFT + CSRL	2	6	44.9

E.2 EFFECT OF TRANSFERABILITY SCORING

We isolate the contribution of transferability scoring on QWENCLAWBENCH with Qwen3.5-9B. The two variants use $\mathrm{Score}_0 = Q$ and $\mathrm{Score}_0 = Q + \mathrm{Tran}$, respectively, with node and context indices omitted. Both retain collective quality assessment and the same CSTS tree construction; the training task pool, generation and judge models, trajectory teacher, and other settings are fixed. Following Appendix C.1, Table 8 shows that adding transferability raises the average score from 43.1 to 44.9, a gain of 1.8 percentage points.

Table 8: Transferability scoring ablation on QWENCLAWBENCH with Qwen3.5-9B. Scores are averaged over five evaluation runs.

Scoring variant	Initial node score	Avg.
Without transferability	Q	43.1
With transferability	$Q + \mathrm{Tran}$	44.9

E.3 HYPERPARAMETER STUDY

We first examine the effect of the number of retained skills K in Table 9. For the K sweep, we fix the total rollout budget per group at $KG = 12$, using $G = 6, 4, 3$ for $K = 2, 3, 4$, respectively. The default training configuration is $K = 2, G = 6$ (Table 6). Increasing K from 2 to 4 improves the average QWENCLAWBENCH score from 44.9 to 45.6, showing that additional retained skills provide more diverse cross-skill comparisons and learning signals. The gains diminish as K increases. We thus adopt $K = 2$ as the default to balance skill diversity with the number of rollout samples available for estimating each skill's utility. We then study the EMA coefficient η_u. The score increases from 44.2 at $\eta_u = 0.90$ to 44.9 at $\eta_u = 0.99$, but slightly decreases to 44.7 at $\eta_u = 0.999$. A smaller coefficient makes skill utility estimates more sensitive to recent observations, whereas an overly large coefficient adapts too slowly to policy updates. We therefore set $\eta_u = 0.99$ to balance stability and responsiveness.

Table 9: Hyperparameter study on QWENCLAWBENCH (Qwen Team, 2026c). Default settings are in bold.

Method	Hyperparameter	Value	Avg.
OPENCLAW-SKILL	K	**2**	**44.9**
		3	45.4
		4	45.6
	η_u	0.90	44.2
		0.99	**44.9**
		0.999	44.7

Candidate Pool for $K = 4$. For the $K = 4$ ablation with $N = J = 3$, we reuse historical skills at each layer to expand the candidate pool for the current subtask to more than four skills. When no reusable historical skills are available, we generate additional candidate skills until the pool contains more than four skills. We then retain the top $K = 4$ candidates for that subtask.

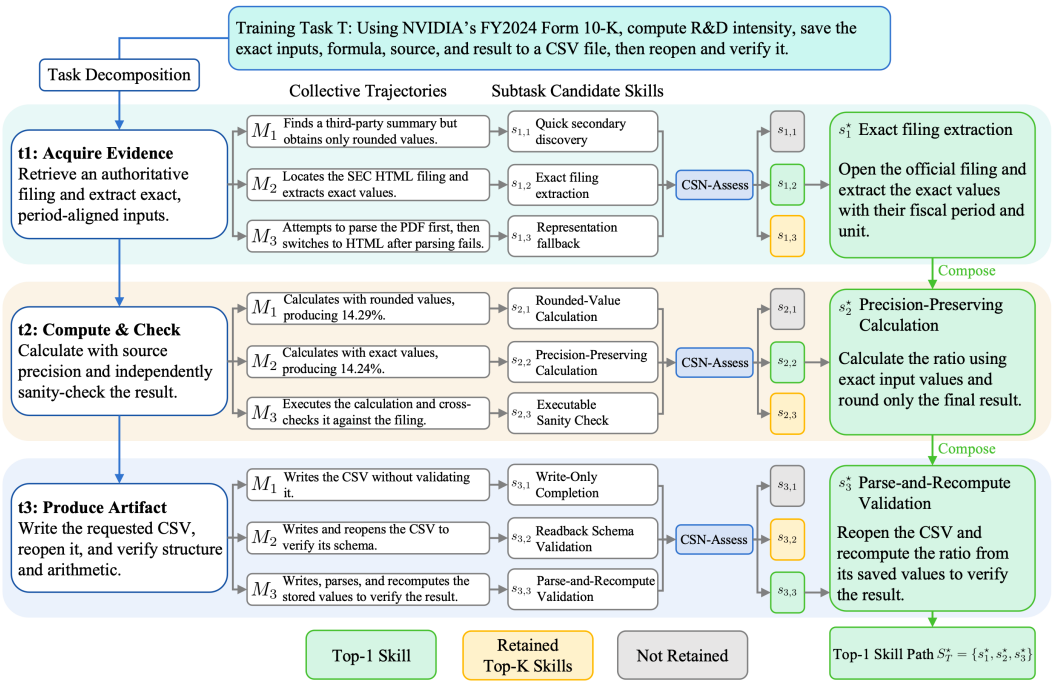

Figure 3: Illustration of the skills produced by CSN-Gen and CSN-Assess. CSN-Gen decomposes a training task into ordered subtasks and abstracts heterogeneous model trajectories into candidate skill nodes through the shared skill synthesizer. CSN-Assess combines collective quality assessment with cross-model transferability to select a Top-1 skill at each layer while retaining useful Top-K alternatives. The selected nodes are composed into the final skill path for the task.

F Illustration of Skill Construction by CSN-Gen and CSN-Assess

Figure 3 illustrates the skills produced by CSN-Gen and CSN-Assess on a representative training task. In this example, CSN-Gen first uses a strong teacher model, GPT-5.4, to decompose the overall objective into ordered subtasks, including acquiring authoritative evidence, computing with high precision from source-level inputs, and producing a verified CSV artifact. Multiple models then generate diverse execution trajectories for each subtask, revealing complementary strengths and failure modes: fast secondary-source discovery with rounded values, exact extraction from U.S. Securities and Exchange Commission (SEC) filings, representation fallback after PDF parsing failure, and validation by reopening the saved file and recomputing the result.

CSN-Gen further abstracts these execution trajectories into candidate skill nodes through the shared skill synthesizer, while CSN-Assess scores and filters the candidates using collective quality assessment and cross-model transferability evaluation. This assessment removes unstable or less transferable candidates, retains useful Top-K alternatives, and selects the Top-1 node at each layer. The selected nodes are composed into a complete skill path that includes evidence-source provenance, numerical-precision constraints, and file-level verification information.

These constructed skills are subsequently used by the training pipeline. The Top-1 skill path is injected into the teacher model to generate high-quality demonstration trajectories; meanwhile, the teacher model's task-decomposition process is also incorporated into these demonstrations and used as training data for CSTS-SFT. For CSRL, each subtask's Top-K skills are injected into the policy model to produce skill-conditioned rollouts, which are optimized with a GRPO-style objective. The cross-skill advantage further serves as an auxiliary signal for calibrating skill values, and all skills are cached in the skill bank together with their scores. Overall, the example shows that the collective-intelligence mechanism performs structured comparison across diverse execution strategies, rather than simple majority voting.

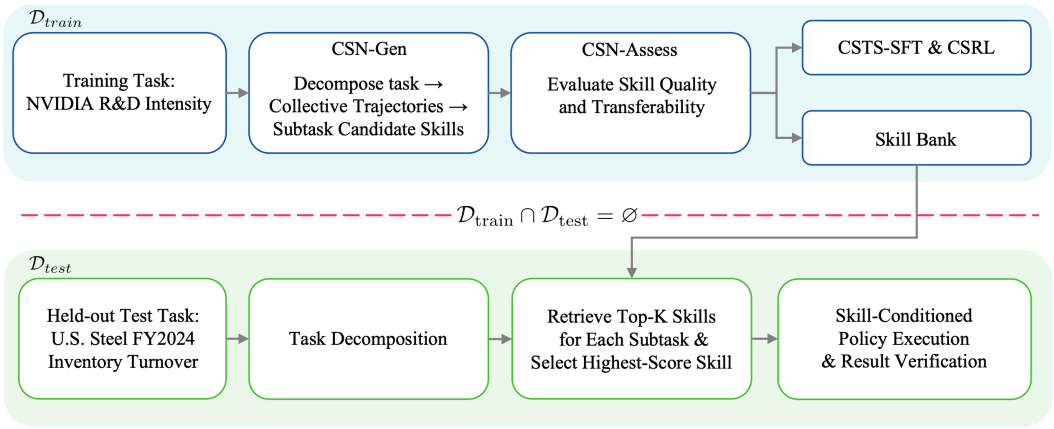

Figure 4: Test-stage skill retrieval and policy execution. The training and test task sets are disjoint: skills are constructed from training tasks and cached in the skill bank, while held-out test tasks are used only at evaluation time. At test time, the policy model decomposes the task, uses Qwen3-Embedding-8B to encode each subtask description and each skill's textual representation, retrieves the Top-K skills ($K = 2$) by cosine similarity, selects the retrieved skill with the highest cached score averaged across its evaluated training contexts, injects the selected skill into its context, and executes the task with result verification.

G Test-Stage Skill Retrieval and Policy Execution

Figure 4 illustrates the test-stage workflow and clarifies the separation between training and evaluation tasks. The training set $\mathcal{D}_{\text{train}}$ and the held-out test set $\mathcal{D}_{\text{test}}$ are disjoint. Skills are constructed only from training tasks, such as the NVIDIA R&D-intensity example, and are stored in the skill bank after CSN-Gen, CSN-Assess, CSTS-SFT, and CSRL. The held-out test task shown in the figure, U.S. Steel FY2024 inventory turnover, is not used during skill construction or training. During training, node scores remain conditioned on their prefix contexts. After training, we assign each skill-bank entry s a fixed score by averaging its final recorded calibrated scores over the set $\mathcal{C}(s)$ of training contexts in which that same skill was evaluated:

$$\text{Score}_{\text{bank}}(s) = \frac{1}{|\mathcal{C}(s)|} \sum_{c \in \mathcal{C}(s)} \text{Score}_{\text{final}}(s \mid c).$$

Each evaluated context contributes one final score to this average. The resulting scores are frozen before evaluation and used as ranking priors among retrieved candidates.

At test time, the policy model receives the held-out task and first decomposes it into ordered subtasks. For each subtask, we use Qwen3-Embedding-8B to encode both the subtask description and the textual representations of skills in the skill bank. Cosine similarity is then used as the retrieval metric to identify semantically relevant candidates, from which we retrieve the Top-K skills for each subtask, with $K = 2$ in our implementation. We then select the retrieved skill with the highest cached score $\text{Score}_{\text{bank}}(s)$, so that the injected skill is both applicable to the current subtask and reliable according to training-time assessment. The selected skill for each subtask is injected into the policy model's context, after which the policy executes the task and verifies the final result. This procedure transfers reusable procedural knowledge without leaking training tasks into the held-out evaluation set.

H Test-Time Skill-Conditioned Qualitative Result

To provide a concrete illustration of the model's tool-use and execution behavior, we present a representative trajectory of OpenClaw-Skill 9B on a financial analysis task from PinchBench. The selected task, `task_financial_ratio_calculation`, requires the agent to research financial figures, compute an inventory turnover ratio, and save the result to a specified file. This

example illustrates how the model coordinates web search, information extraction, numerical calculation, and file writing while continuing execution despite tool feedback such as failed searches or fetch errors. Before presenting the turn-level trajectory, we first show the task decomposition and the retrieved skills injected into the policy context. URLs, long command bodies, and verbose tool outputs are shortened for readability; the reasoning and execution order are preserved.

User Input

Calculate the inventory turnover ratio for U.S. Steel for fiscal year 2024. Research the necessary financial figures, perform the calculation, and save the result to `us_steel_inventory_turnover.txt`. Include the formula, the values you used, and the final ratio.

H.1 Retrieved Skill Illustration

Given the held-out task, the policy model first decomposes it into three ordered subtasks:

$$T_{\text{test}} \rightarrow (t_1^{\text{test}}, t_2^{\text{test}}, t_3^{\text{test}}),$$

where t_1^{test} acquires authoritative evidence, t_2^{test} computes and checks the inventory-turnover ratio, and t_3^{test} writes and verifies the requested artifact. Following Appendix G, each subtask retrieves the Top-K candidate skills from the frozen skill bank using Qwen3-Embedding-8B and cosine similarity, with $K = 2$. We then inject the candidate with the highest cached score $\text{Score}_{\text{bank}}(s)$, yielding the Top-1 skill path

$$\mathcal{S}_{\text{ret}}(T_{\text{test}}) = (s_{1,2}, s_{2,2}, s_{3,3}).$$

The selected skills provide reusable procedural guidance for the subsequent policy execution.

Skill Document: ExactFilingExtraction.md

Metadata:

- **Name:** `exact-filing-extraction`
- **Description:** Extract exact, period-aligned quantitative values from authoritative filings and return traceable evidence for downstream calculation.

When to Use:

- A task asks for values reported by a company, regulator, or official organization.
- The result depends on exact periods, units, statement labels, or source provenance.
- Secondary summaries can help discovery but cannot be the final evidence source.

Strategy Overview:

1. **Source Grounding:** Prefer regulator filings, issuer reports, or other authoritative sources.
2. **Representation Fallback:** Try machine-readable HTML or XBRL when PDF extraction is noisy.
3. **Period and Unit Alignment:** Distinguish flow variables from point-in-time balances and keep units explicit.
4. **Evidence Packaging:** Return item, value, date or period, unit, statement label, source, and location.

Workflow: Locate the authoritative filing; verify the entity and reporting period; extract each requested line item with its label, date, unit, and source location; retain reported precision; and pass a structured evidence record to the next skill.

Failure Handling: If parsing fails, switch representation. If search is rate-limited, use the filing index or issuer filing page. If a line item is ambiguous, compare the primary statement with the corresponding note before returning evidence.

Skill Document: PrecisionPreservingCalculation.md

Metadata:

- **Name:** `precision-preserving-calculation`
- **Description:** Compute derived metrics from exact inputs while preserving intermediate precision and checking the result independently.

When to Use:

- A task requires a ratio, percentage, average, growth rate, or other metric derived from retrieved numbers.
- Input values come from multiple lines, periods, or units and must be reconciled before calculation.
- The final answer must be rounded, but intermediate arithmetic should not be rounded early.

Strategy Overview:

1. **Formula First:** State the target formula before substituting values.
2. **Unit Normalization:** Convert inputs to compatible units and aligned periods.
3. **Precision Discipline:** Keep exact inputs and full intermediate precision; round only the final reported value.
4. **Independent Check:** Recompute through a separate calculation path and compare against tolerance.

Workflow: Validate periods and units; write the formula; substitute exact source values; compute the raw result; round the final answer according to the task; and return the formula, exact inputs, intermediates, raw result, rounded result, and check status.

Failure Handling: If the check fails, inspect unit conversion, period alignment, formula choice, numerator and denominator order, signs, premature rounding, and the extracted source values before recomputing.

Skill Document: ParseAndRecomputeValidation.md

Metadata:

- **Name:** `parse-and-recompute-validation`
- **Description:** Validate a saved artifact by reopening it, parsing required fields, and recomputing derived results from the stored content.

When to Use:

- A task requires writing a persistent output file such as text, CSV, JSON, or a spreadsheet.
- The artifact stores derived values that must remain consistent with the inputs and formula.
- Completion should be judged from the saved file contents rather than from a successful write call alone.

Strategy Overview:

1. **Write and Reopen:** Save the artifact, then read it back from disk.
2. **Schema Check:** Confirm all required fields are present, nonempty, and parseable.
3. **Recompute from Saved Values:** Recalculate the derived result using only values stored in the artifact.
4. **Repair Loop:** If structure or arithmetic fails, rewrite the artifact and repeat the readback check.

Workflow: Write the requested artifact; reopen it; parse fields, units, periods, and numerical values; recompute the target metric from the saved inputs; compare the recomputed and saved results; and return a validation report.

Failure Handling: If the file is missing, write it again to the correct path. If fields are missing, add them and repeat readback. If parsing fails, normalize the numeric representation. If recomputation disagrees, correct the stored inputs, formula, intermediate value, or final result before validating again.

H.2 Test-Time Trajectory Illustration

We first provide a compact illustration of the test-time setup for the trajectory, covering task decomposition, skill retrieval, and context injection before tool execution begins.

Task Decomposition

The policy model decomposes the held-out task into ordered subtasks:

$$T_{\text{test}} \rightarrow (t_1^{\text{test}}, t_2^{\text{test}}, t_3^{\text{test}}),$$

where t_1^{test} acquires authoritative evidence, t_2^{test} computes and checks the ratio, and t_3^{test} writes and verifies the artifact.

Skill Retrieval

For each subtask, the policy retrieves Top-K candidate skills ($K = 2$) from the frozen skill bank using Qwen3-Embedding-8B and cosine similarity, then selects the highest-scoring candidate for injection:

- $t_1^{\text{test}} \rightarrow s_{1,2}$: `ExactFilingExtraction.md`
- $t_2^{\text{test}} \rightarrow s_{2,2}$: `PrecisionPreservingCalculation.md`
- $t_3^{\text{test}} \rightarrow s_{3,3}$: `ParseAndRecomputeValidation.md`

Context Injection

The retrieved skill documents are injected into the policy context as procedural guidance. The agent then begins tool execution.

The following trace illustrates the resulting policy execution with the retrieved skill path $\mathcal{S}_{\text{ret}}(T_{\text{test}})$.

Turn 1: Model Tool Calls

```
read("/home/node/workspace/SOUL.md")
read("/home/node/workspace/USER.md")
exec("date -u +%F && ls -1 /home/node/workspace/memory
2>/dev/null | tail -n 5")
```

Turn 1: Tool Responses

read(SOUL.md): success. Loaded `/home/node/workspace/SOUL.md`.
read(USER.md): success. Loaded `/home/node/workspace/USER.md`.
exec: success. Output: `2026-05-05`; no visible memory-file listing.

Turn 2: Model Message

I'm pulling the FY2024 inventory and cost-of-sales figures, then I'll compute and save the result.

Turn 2: Model Tool Calls

```
web_search(
query="U.S. Steel parent company fiscal year 2024 Cost of goods sold
annual report inventory current assets 12M 2024 U.S. Steel parent
company SEC EDGAR annual report",
count=10
)

web_search(
query="\"U.S. Steel\" inventory turnover ratio 2024 market data
financial statements",
count=10
)
```

Turn 2: Tool Responses

web_search #1: success. Provider: Brave; count: 10; took: 1070ms.

Query: U.S. Steel parent company fiscal year 2024 Cost of goods sold annual report inventory current assets 12M 2024 U.S. Steel parent company SEC EDGAR annual report

Returned candidate URLs:

[`https://www.sec.gov/.../tm255512d2_ars.pdf`,

`https://bullfincher.io/.../financial-statements`,

`https://investors.ussteel.com/.../tm255512d2_ars.pdf`,

...]

Verbose outputs are truncated for readability.

web_search #2: failed.

Query: Search for U.S. Steel's 2024 inventory turnover ratio and related financial statement data.

Error: Brave Search API 429 `RATE_LIMITED` (Request rate limit exceeded for plan).

Turn 3: Model Tool Calls

```
web_fetch(
url="https://investors.ussteel.com/.../tm255512d2_ars.pdf",
extractMode="markdown", maxChars=30000
)

web_fetch(
url="https://bullfincher.io/.../financial-statements",
extractMode="markdown", maxChars=30000
)
```

Turn 3: Tool Responses

web_fetch #1: failed.
URL: `https://investors.ussteel.com/.../tm255512d2_ars.pdf`
Error: `getaddrinfo ENOTFOUND investors.ussteel.com`.

web_fetch #2: success.
URL: `https://bullfincher.io/.../financial-statements`
Status: `200`; **contentType:** `text/html`; **extractMode:** `markdown`; **took:** `2018ms`.
Title: United States Steel Corporation Financial Statements 2020–2024.

Extracted financial figures:

Metric	Year	Value
Revenue	2024	$15.64B
Cost of revenue/sales	2024	$14.06B
Inventories	2023	$2.13B
Inventories	2024	$2.17B
Total Current Assets	2024	$5.23B

Tool outputs are truncated for readability.

Turn 4: Model Message

I found usable 2024 figures from a financial summary page. Next I'll try to pin them down against U.S. Steel's actual annual report, then I'll write the result file.

Turn 4: Model Tool Calls

```
web_search(
query="site:sec.gov United States Steel Corporation 2024 annual report
inventory cost of sales 2024 U.S. Steel 12M 2024",
count=10
)

web_search(
query="\"United States Steel Corporation\" \"cost of sales\" 2024
annual report inventory 2024",
count=10
)
```

Turn 4: Tool Responses

web_search #1: failed.
Error: Brave Search API 429 `RATE_LIMITED` (Request rate limit exceeded for plan).

web_search #2: success. Provider: Brave; count: 2; took: 1002ms.

Query: Search for United States Steel Corporation cost of sales, 2024 annual report, and inventory information.

Returned candidate URLs:

[https://www.ussteel.com/.../Third_Quarter_2025_Financial_Statements_Final.pdf,

https://www.ussteel.com/.../2025_Annual_Financial_Statements_vPosting.pdf]

Relevant snippets:

- The first result mentions LIFO inventory values at September 30, 2025 and December 31, 2024.
- The second result mentions 2024 inventory-related values and cost of sales context.

Tool outputs are truncated for readability.

Turn 5: Model Message

I found a U.S. Steel PDF source that mentions 2024 inventory/LIFO conversion figures. I'm opening that now to confirm the official 2024 numbers for inventory and cost of sales before saving.

Reader note: After obtaining usable values from a financial summary page, the model tries to verify them against U.S. Steel PDF sources before writing the final artifact.

Turn 5: Model Tool Calls

```
web_fetch(
url="https://www.ussteel.com/.../2025_Annual...Statements_vPosting.pdf",
extractMode="markdown",
maxChars=40000
)

web_fetch(
url="https://www.ussteel.com/.../Third_Quarter...Statements_Final.pdf",
extractMode="markdown",
maxChars=40000
)
```

Turn 5: Tool Responses

web_fetch #1: success.
URL: https://www.ussteel.com/.../2025_Annual...Statements_vPosting.pdf
Status: 200; **contentType:** application/pdf; **extractMode:** markdown; **extractor:** raw; **took:** 3550ms.
Returned content: raw PDF stream beginning with %PDF-1.6; most extracted content is binary or encoded PDF text.

web_fetch #2: success.
URL: https://www.ussteel.com/.../Third_Quarter...Statements_Final.pdf
Returned content: verbose PDF output, mostly raw or encoded content.

Reader note: The official PDF sources are accessible, but the extracted content is not immediately readable because the fetch tool returns raw PDF streams. Tool outputs are truncated for readability.

Turn 6: Model Tool Call

```
exec(
command="python3 - <<'PY' ... parse fetched PDF text ... PY",
workdir="/home/node/workspace"
)
```

Turn 6: Tool Response

exec: completed.
Output: no useful visible stdout.

Reader note: Since the PDF fetch returned raw encoded content, the model attempts a local parsing step to recover readable financial text. This attempt does not produce useful output.

Turn 7: Model Tool Call

```
exec(
command="python3 - <<'PY' ... import PyPDF2; search PDF pages for
inventory and cost-of-sales terms ... PY",
workdir="/home/node/workspace"
)
```

Turn 7: Tool Response

exec: failed.
Error: `ModuleNotFoundError: No module named 'PyPDF2'.`
Exit code: `1`.

Reader note: The model tries another PDF parsing strategy, but the required local PDF library is unavailable. This failure is non-blocking because the needed financial values have already been obtained from earlier sources.

Turn 8: Model Message

I've got enough evidence to compute the ratio. I'm writing the result to `us_steel_inventory_turnover.txt` and then reopening the file for validation.

Reader note: After the PDF parsing attempts, the model proceeds with the values already collected, writes the required artifact, and validates the saved file.

Turn 8: Model Tool Calls

```
write(
path="/home/node/workspace/us_steel_inventory_turnover.txt",
content="U.S. Steel Inventory Turnover Ratio - Fiscal Year 2024 ..."
)

read("/home/node/workspace/us_steel_inventory_turnover.txt")

exec(
command="python3 - <<'PY' ... recompute saved ratio ... PY",
workdir="/home/node/workspace",
timeout=30
)
```

Turn 8: Tool Responses

write: success.
Output: Successfully wrote `1045 bytes` to `/home/.../us_steel_inventory_turnover.txt`.

read: success. Reopened `/home/node/workspace/us_steel_inventory_turnover.txt`.

exec: success.
Output: parsed saved values; recomputed average inventory $= 2.150$B and inventory turnover $= 6.54$; validation passed.

Reader note: The file-writing and readback-validation steps succeed, confirming that the requested output file has been created and that the saved numerical values reproduce the reported result.

Final Output

Done — I calculated it and saved:

- `us_steel_inventory_turnover.txt`

Main result:

- Inventory turnover ratio for U.S. Steel FY2024: **6.54x**

Formula used:

- Inventory turnover = Cost of sales ÷ Average inventory
- Average inventory = (Beginning inventory + Ending inventory) ÷ 2

Numbers used:

- Cost of sales for 12 months ended Dec. 31, 2024: $14.060B
- Inventory at Dec. 31, 2023: $2.130B
- Inventory at Dec. 31, 2024: $2.170B
- Average inventory: $2.150B
- $14.060B ÷ $2.150B = 6.54

I LIMITATIONS

The skills constructed in this work are intentionally compact because they are associated with decomposed subtasks rather than entire end-to-end workflows. This design makes the skills easy to retrieve, compose, and inject during test-time execution, but it may underrepresent the richer structure of skills required in more complex real-world applications. In practice, useful skills may need to encode longer procedural dependencies, conditional branches, tool-specific conventions, and domain-specific verification rules. Future work can extend OPENCLAW-SKILL toward more hierarchical and compositional skill documents while preserving the retrieval and transfer benefits of subtask-level abstraction.