Toward Scalable Terminal Task Synthesis via Skill Graphs

Zhiyuan Fan [1 2] **Tinghao Yu** [1] **Yuanjun Cai** [1] **Jiangtao Guan** [1] **Yun Yang** [1] **Dingxin Hu** [1] **Jiang Zhou** [1] **Xing Wu** [1] **Zhuo Han** [1] **Feng Zhang** [1] **Lilin Wang** [1]

Abstract

Terminal agents have demonstrated strong potential for autonomous command-line execution, yet their training remains constrained by the scarcity of high-quality and diverse execution trajectories. Existing approaches mitigate this bottleneck by synthesizing large-scale terminal task instances for trajectory sampling. However, they primarily focus on scaling the number of tasks while providing limited control over the diversity of execution trajectories that agents actually experience during training. In this paper, we present *SkillSynth*, an automated framework for terminal task synthesis built on a *scenario*-mediated skill graph. *SkillSynth* first constructs a large-scale skill graph, where scenarios serve as intermediate transition nodes that connect diverse command-line skills. It then samples paths from this graph as abstractions of real-world workflows, and uses a multi-agent harness to instantiate them into executable task instances. By grounding task synthesis in graph-sampled workflow paths, *SkillSynth* explicitly controls the diversity of minimal execution trajectories required to solve the synthesized tasks. Experiments on Terminal-Bench demonstrate the effectiveness of *SkillSynth*: on Terminal-Bench 2.0, Qwen3-32B trained on *SkillSynth* trajectories reaches 29.6% accuracy, outperforming the much larger Qwen 3 Coder 480B model (23.9%) despite using 15× fewer parameters.

1. Introduction

Terminal agents leverage the command-line interface (CLI) as a universal action space, enabling large language models (LLMs) to execute complex, long-horizon tasks across computing systems (Jimenez et al., 2024; Team, 2025; Merrill et al., 2026). Yet their capabilities remain fundamentally limited by the scarcity of high-quality, diverse trajectories for training. Since manually curating executable terminal tasks is expensive and difficult to scale (Lin et al., 2018; Merrill et al., 2026), recent work has turned to large-scale terminal task synthesis as a path toward scalable trajectory collection.

To understand what makes such trajectories useful for training, it helps to look at how terminal agents actually operate. During execution, a terminal agent does not plan over the entire environment in a single shot. Instead, it acts through step-by-step interaction with a sequence of intermediate *scenarios*, applying a *skill* at each scenario to make progress. An execution trajectory is therefore jointly characterized by two dimensions: the scenarios it traverses and the skills it exercises. Training terminal agents thus amounts to learning to apply appropriate skills across diverse partial scenarios, which in turn requires trajectories that are diverse along both dimensions.

Existing efforts, however, primarily scale terminal task instances, either by broadening domain coverage via LLM-generated taxonomies, which often diverge from real-world usage (Pi et al., 2026; Zhu et al., 2026), or by deriving task instances from real GitHub repositories and inverting healthy environments into buggy states (Wu et al., 2026; Lin et al., 2026; Chen et al., 2026), which remain narrowly scoped to software-engineering domains such as issue resolution and feature development (Yang et al., 2025b; Zhang et al., 2025; Wang et al., 2025a). These approaches provide limited explicit control over the scenario or skill composition underlying the resulting trajectories.

As empirically shown in Figure 1, these trajectories exhibit redundancy in both scenario coverage and skill usage: different task instances often expose the agent to overlapping scenarios and reuse similar skills (Wu et al., 2026).

To address this gap, we propose *SkillSynth*, a scalable framework for constructing diverse terminal tasks. *SkillSynth* first collects skills from ClawHub (OpenClaw, 2026) and public GitHub repositories, capturing practical experience distilled from real terminal usage. For each skill, *SkillSynth*

[1]Hunyuan Team, Tencent [2]The Hong Kong University of Science and Technology, Hong Kong SAR, China. Correspondence to: Zhiyuan Fan <zhiyuan.fan@connect.ust.hk>, Tinghao Yu <maxwellyu@tencent.com>, Lilin Wang <lilinwang@tencent.com>.

Published at the Second Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD) at ICML 2026. Copyright 2026 by the author(s).

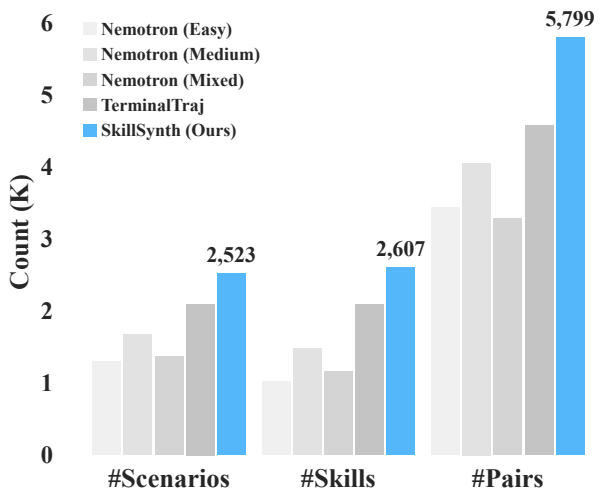

Figure 1. Diversity of synthesized trajectories across datasets, measured by the number of unique scenarios, skills, and (scenario, skill) pairs after semantic canonicalization. Each value is averaged over three independent samples of 1,000 trajectories per dataset.

derives a precondition scenario that describes when the skill is applicable, and a postcondition scenario that describes the state reached after its execution. It then links semantically compatible scenarios across skills to construct a scenario-mediated skill graph, in which scenarios serve as nodes and skills as directed transitions between them. A directed path sampled from the graph therefore specifies an agentic workflow, consisting of a sequence of skills ordered together with the intermediate scenarios they traverse. We then use a multi-agent harness to instantiate each sampled workflow into a concrete executable task instance whose intended solution realizes the sampled path, with oracle-based verification to ensure solvability and rubric-based evaluation to ensure task quality.

In a single fully automated run, *SkillSynth* constructs 3,560 verified task instances from 3,721 sampled paths, achieving a 95.7% oracle pass rate at only an average cost of $27.3 per verified task instance. Compared with tasks synthesized from single-skill seeds or randomly composed multi-skill seeds, *SkillSynth* tasks are more challenging and diverse: Claude Opus 4.6 requires 37 steps on average to solve them, and 121 tasks remain unsolved after three independent rollouts. We validate the effectiveness of *SkillSynth* by performing supervised fine-tuning on Qwen3 models at 8B, 14B, and 32B scales using sampled trajectories, and evaluating on Terminal-Bench 1.0 and 2.0. With the enhanced diversity of the sampled trajectories, the fine-tuned models achieve improved performance with higher data efficiency. Additionally, task instances synthesized by *SkillSynth* have been adopted to train Hy3 Preview (Tencent Hy Team, 2026), contributing to its enhanced agentic capabilities.

Currently, the constructed skill graph contains 82,073 scenarios after deduplication and merging, 57,214 filtered skills, and 185,529 LLM-verified bridges, from which a large space of workflow paths can be sampled for task construction. Its scenario-mediated structure makes the graph a naturally scalable infrastructure for task synthesis. As the community contributes more skills, the graph continues to expand, enabling continual synthesis of diverse terminal tasks.

Our contributions are summarized as follows:

1. We model agentic trajectories as sequences of scenarios and skills to analyze their diversity, and introduce a scenario-mediated skill graph that organizes existing skills as a foundation for sampling workflows for controllable synthesis of terminal task instances.

2. We build an end-to-end multi-agent harness that synthesizes sampled paths into executable task instances without human intervention, achieving a 95.7% oracle pass rate and producing 3,560 verified task instances in a single fully automatic run.

3. We collect trajectories on the synthesized task instances and demonstrate that graph-guided synthesis yields harder tasks and more diverse trajectories than single-skill or composed multi-skill baselines, leading to consistent gains on Terminal-Bench 1.0 and 2.0 across model scales.

2. Problem Formulation

Terminal agent task. We formulate a terminal agent task as a tuple $\tau = (\mathcal{E}, s_0, g, V)$. From the initial state $s_0 \in \mathcal{S}$, an agent π interacts with the executable environment $\mathcal{E}$ to achieve a natural language described goal g. At step t under partial observability, the agent receives an observation $o_t \in \mathcal{O}$ and samples an action $a_t \sim \pi(\cdot \mid o_{\leq t}, a_{<t}, g)$. The resulting low-level trajectory $\zeta = (o_0, a_0, \ldots, o_T)$ is labeled successful if the final state satisfies the external verifier, i.e., $V(s_T) = 1$.

Scenario and skill abstraction. However, low-level trajectories ζ consist only of fine-grained observations and actions, obscuring the high-level strategy the agent applies during execution. We thus lift ζ to a higher semantic abstraction through two objects. A *scenario* $\sigma_t \in \Omega$ is a decision-relevant abstraction of the observation at a decision point of execution, and by construction serves as a sufficient statistic of the interaction history up to that point for the agent's next decision. A *skill* $\kappa_t \in \mathcal{K}$ is an action subsequence applied at one scenario that produces a predictable transition to the next,

$$\kappa_t : \sigma_{t-1} \to \sigma_t, \qquad \kappa_t = (a_{i_t}, a_{i_t+1}, \ldots). \tag{1}$$

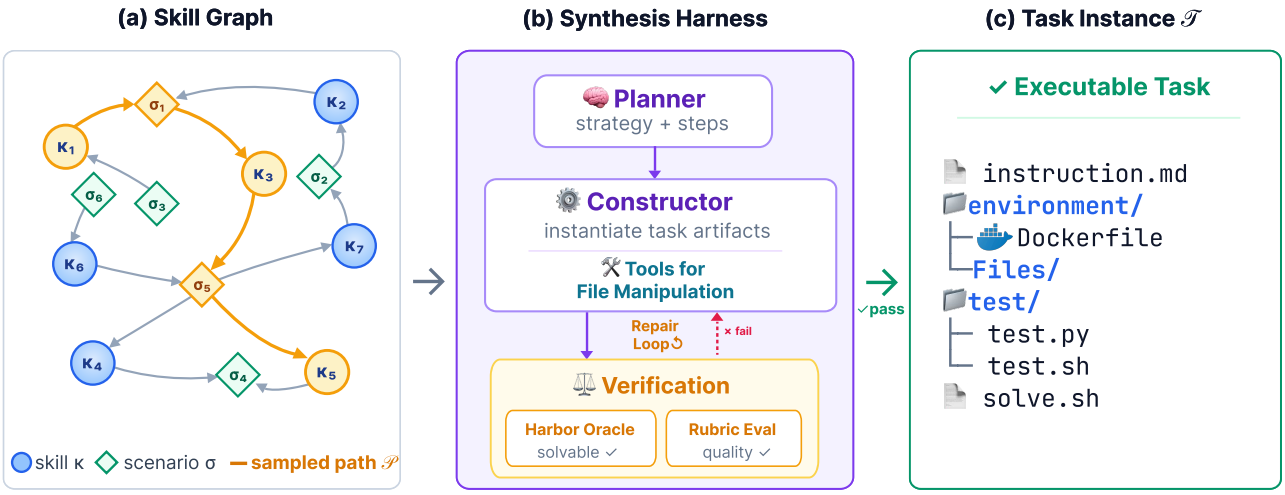

Figure 2. **Overview of *SkillSynth*.** (a) A compositional path $\mathcal{P}$ of scenarios σ and skills κ (highlighted in orange) is sampled from the scenario-mediated skill graph. (b) A multi-agent harness instantiates $\mathcal{P}$ into the components of a task instance through a planner and a tool-augmented synthesis agent, followed by dual verification that checks solvability (execution-based) and specification quality (rubric-based); failed instances re-enter the synthesis loop for iterative repair. (c) The resulting task instance $\mathcal{T}$ is fully executable and verified.

Under this identification, ζ induces an *execution trajectory*

$$\xi = \big(\sigma_0, \kappa_1, \sigma_1, \kappa_2, \ldots, \kappa_L, \sigma_L\big) \in (\Omega \times \mathcal{K})^L \times \Omega, \quad (2)$$

which jointly characterizes the scenarios traversed and the skills exercised (cf. options in hierarchical RL (Sutton et al., 1999)).

Learning objective. Based on Equation 2, an agent reduces to a policy over skills conditioned on scenarios, and training such an agent thus amounts to maximizing the expected skill-selection likelihood under the empirical distribution $\mathcal{D}$ induced by the training trajectories:

$$\mathcal{J}(\pi) = \mathbb{E}_{\xi\sim\mathcal{D}} \sum_{t=1}^{L} \log \pi(\kappa_t \mid \sigma_{t-1}, g)\,, \quad (3)$$

which is equivalent to the standard next-token-prediction loss on low-level interaction trajectories.

Equivalence to action-level training. The skill-level objective in Equation 3 reduces to the standard next-action objective under three assumptions: (A1) the mapping $\zeta \mapsto \xi$ is deterministic, (A2) each scenario σ_t is a sufficient statistic for the agent's next decision, and (A3) each skill $\kappa_t = (a_{i_t}, \ldots, a_{j_t})$ is executed autoregressively. Under (A2) and (A3), the chain rule of conditional probability gives

$$\log \pi(\kappa_t \mid \sigma_{t-1}, g) = \sum_{k=i_t}^{j_t} \log \pi(a_k \mid o_{\leq k}, a_{<k}, g). \quad (4)$$

Summing over $t = 1, \ldots, L$ and using the contiguity $i_{t+1} = j_t + 1$, the double sum collapses into a single sum over all actions. Taking expectations and applying (A1) yields

$$\begin{aligned} \mathcal{J}(\pi) &= \mathbb{E}_{\xi\sim\mathcal{D}} \sum_{t=1}^{L} \log \pi(\kappa_t \mid \sigma_{t-1}, g) \\ &= \mathbb{E}_{\zeta\sim\mathcal{D}} \sum_{t=0}^{T-1} \log \pi(a_t \mid o_{\leq t}, a_{<t}, g). \end{aligned} \quad (5)$$

This equivalence shows that the scenario–skill abstraction imposes no additional training machinery, and properties of $\mathcal{D}$ established at the scenario–skill level transfer directly to the token-level loss. For a fixed goal g, decomposing $\mathcal{J}$ over the support of $\mathcal{D}$ yields

$$\mathcal{J}(\pi) = \sum_{\sigma\in\Omega} p_{\mathcal{D}}(\sigma \mid g) \sum_{\kappa\in\mathcal{K}_\sigma} p_{\mathcal{D}}(\kappa \mid \sigma, g)\, \log \pi(\kappa \mid \sigma, g), \quad (6)$$

where $\mathcal{K}_\sigma \subseteq \mathcal{K}$ denotes skills admissible at σ. Equation 6 makes explicit that the learnable region of π is confined to the support of $\mathcal{D}$ along both factors: scenarios with $p_{\mathcal{D}}(\sigma \mid g) = 0$ are unobservable, and skills with $p_{\mathcal{D}}(\kappa \mid \sigma, g) = 0$ are unexercised. Maximizing learned capacity therefore requires training data whose induced $\mathcal{D}$ densely covers the conditional product space $\{(\sigma, \kappa) : \sigma \in \Omega,\ \kappa \in \mathcal{K}_\sigma\}$, which motivates our approach to scale synthetic task instances by maximizing the diversity of execution trajectories experienced by the agent.

3. Approach

3.1. Overview

SkillSynth synthesizes diverse terminal task instances through three stages. We first construct a scenario-mediated

skill graph, in which nodes are scenarios and directed edges represent skills that point from precondition to postcondition scenarios (§ 3.2). From this graph we sample directed paths, each specifying a compositional sequence of skills to be exercised sequentially together with the scenarios they traverse (§ 3.3). A multi-agent harness then instantiates each sampled path into a concrete, executable task instance with oracle verification and rubric-based evaluation to ensure solvability and task quality (§ 3.4).

3.2. Skill Graph Construction

Recall from § 2 that each skill $\kappa \in \mathcal{K}$ is a directed transition $\kappa : \sigma \to \sigma'$ between scenarios. We lift this pairwise structure into a directed multigraph $\mathcal{G} = (\Omega, \mathcal{K})$, where nodes are scenarios and edges are skills. A directed path in $\mathcal{G}$ corresponds to a sequential workflow in which each skill's postcondition serves as the precondition of the next. The full construction pipeline is illustrated in Figure 3.

Skill filtering. We instantiate $\mathcal{G}$ from human-written skills in ClawHub (OpenClaw, 2026) and public GitHub repositories. The skill pool $\mathcal{K}$ retains only skills that are 1) executable on a Linux terminal, 2) defined by a structured workflow rather than prompt engineering alone, 3) free of adversarial or jailbreak content (e.g., downloading files from unknown IPs, exfiltrating environment keys), and 4) producing deterministic, objectively verifiable outputs.

Scenario extraction. All LLM-based extraction and alignment calls use DeepSeek Reasoner (v3.2) to improve consistency on long, tool-heavy skill specifications. We use structured prompts that force preconditions and postconditions to be expressed as executable environment states rather than task names or broad domains. This representation is important because graph edges should encode feasible state transitions, not merely topical similarity between two skills.

Scenario inference. For each retained skill $\kappa \in \mathcal{K}$, we prompt an LLM with its full specification (Markdown description, code, and usage examples) to infer plausible precondition scenarios $\Omega_\kappa^{\text{pre}}$ and postcondition scenarios $\Omega_\kappa^{\text{post}}$, yielding atomic transitions $\{\kappa : \sigma \to \sigma' \mid \sigma \in \Omega_\kappa^{\text{pre}},\ \sigma' \in \Omega_\kappa^{\text{post}}\}$.

Scenario deduplication. All inferred scenarios are embedded and undergo clustering-based semantic deduplication, merging scenarios that describe the same state but differ in lexical form. We use a two-stage scalable hierarchical procedure: Louvain community detection first forms coarse buckets over a sparse semantic similarity graph, and complete-linkage agglomerative clustering is then applied within each bucket. This avoids the quadratic memory cost of global hierarchical clustering while preventing chain-drift merges where transitively linked but semantically distinct states collapse into one node. We evaluate nine common clustering algorithms and find that this Louvain-bucketed hierarchical method performs best empirically.

Cross-skill alignment. To connect atomic transitions into a unified graph, postconditions of one skill are aligned with preconditions of another. For each postcondition scenario, the top-1,000 most similar precondition scenarios are retrieved by embedding similarity, and an LLM judges semantic compatibility. We repeat this process in reverse (precondition $\to$ top-1,000 postconditions) using separately designed prompts to ensure bidirectional alignment quality.

We also considered two simpler construction strategies. Embedding-only alignment is cheaper but frequently connects states that are lexically similar yet operationally incompatible, such as a project with passing tests and a project with failing tests. Single-pass subgraph generation avoids pairwise alignment but produces smaller, less reusable local graphs. The retrieve-then-judge design provides a better tradeoff: embeddings keep alignment scalable, while LLM judgments preserve executable transition semantics.

Scenario merging and filtering. Aligned pre- and postcondition pairs are merged into unified scenario nodes using an LLM. Finally, we perform an LLM-based filtering pass over all resulting (scenario, skill, scenario) triples, retaining only those that form valid transitions. Through manual review of sampled graph cases, we find that each stage is necessary to ensure the overall quality of the constructed graph. Full alignment criteria and prompt details are provided in § A.

3.3. Graph-Guided Path Sampling

Given the skill graph $\mathcal{G} = (\Omega, \mathcal{K})$, we sample directed paths that serve as compositional inputs to the multi-agent synthesis harness (§ 3.4). A sampled path $\mathcal{P} = (\sigma_0, \kappa_1, \sigma_1, \kappa_2, \ldots, \kappa_L, \sigma_L)$ interleaves L skill transitions with $L + 1$ scenarios, where each $\kappa_l \in \mathcal{K}$ is a directed edge from σ_{l-1} to σ_l in $\mathcal{G}$. Paths whose length L falls within $[L_{\min}, L_{\max}]$ are retained, covering single-skill tasks ($L \in \{1, 2, 3\}$) through compositional multi-step ones ($L \geq 4$); we set $L_{\min} = 1$ and $L_{\max} = 7$ in our experiments.

A uniform random walk on $\mathcal{G}$ concentrates on high-degree scenarios and frequently traversed skills, producing redundant paths that revisit the same sub-structures. We instead sample paths with inverse-frequency weighting: for each scenario $\sigma \in \Omega$ we maintain a visit count $\nu(\sigma)$, and for each skill $\kappa \in \mathcal{K}$ a usage count $\mu(\kappa)$. The source scenario is drawn with probability $p(\sigma) \propto (\nu(\sigma) + 1)^{-1}$, and at each step of the walk the next edge is drawn from the outgoing skills of the current node with probability $p(\kappa) \propto (\mu(\kappa) + 1)^{-1}$. To enforce monotone progression, visited scenarios and skills are excluded from subsequent steps within the same path. The walk continues until it

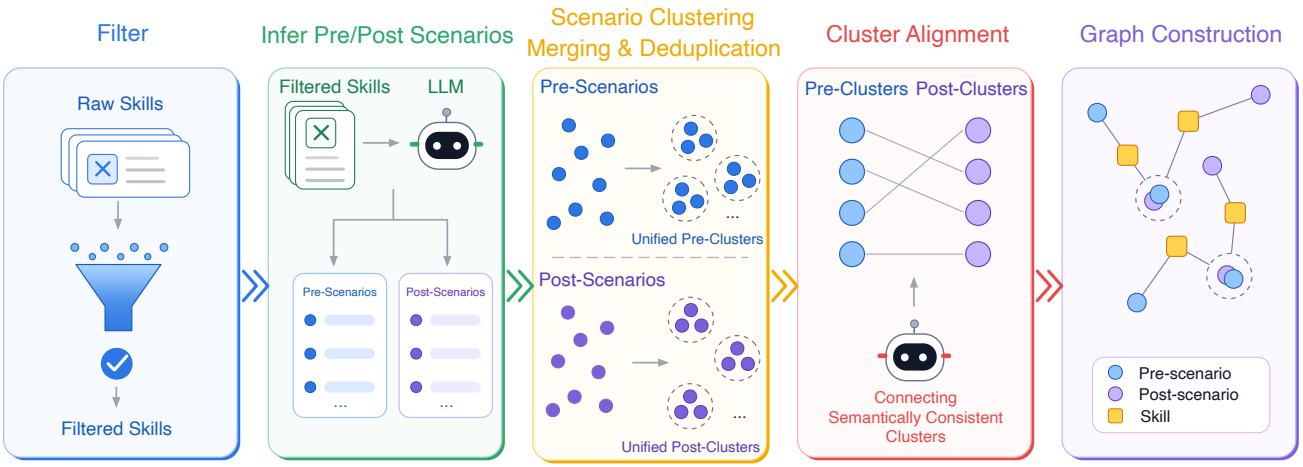

Figure 3. Overview of the skill graph construction pipeline.

reaches $L_{\max}$ or encounters a dead-end with no valid continuation. If the resulting path length falls within $[L_{\min}, L_{\max}]$ and its skill set has not been seen before, the path is accepted and both counters are incremented, progressively steering the empirical distribution toward uniform coverage over $\Omega \times \mathcal{K}$, which is the coverage criterion derived in Equation 6. The full procedure is given in Algorithm 1.

3.4. Multi-Agent Harness

Given a sampled path $\mathcal{P}$ from § 3.3, a multi-agent harness is designed to produce an executable task instance consisting of five components: a natural-language instruction, an initial filesystem snapshot, a containerized environment, verification scripts, and an oracle solution.

Directly prompting an LLM to generate all components in a single pass leads to two issues: 1) long-context generation produces outputs of inconsistent quality, and 2) the model focuses on implementation details rather than designing a coherent task instance, resulting in tasks that lack sufficient complexity. We thus decouple planning from implementation: a planner first transforms $\mathcal{P}$ into a structured plan of sub-objectives and expected outputs, then a constructor generates the whole task instance conditioned on this plan. Each task instance is checked along two complementary axes: *execution-based verification* runs the verification scripts against the oracle solution inside the environment container to ensure solvability; and *rubric-based verification* uses LLM-as-a-Judge to assess (i) alignment between instructions and tests, ensuring that tests are neither too lenient (missing required functionality) nor too strict (imposing unstated constraints), and (ii) instruction self-containedness, ensuring that no hints about the oracle solution progress leak into the instruction. If either check fails, diagnostic feedback is returned to the constructor for repair via multi-turn tool use. We permit up to $R = 3$ repair cycles, each with at most $N_{\text{tool}} = 20$ tool calls, after which the instance is accepted or discarded. An example of a sampled path and the corresponding synthesized task instruction is shown in Figure 4.

Algorithm 1 Inverse-Frequency Path Sampling with Monotone Progression

Require: Skill graph $\mathcal{G} = (\Omega, \mathcal{K})$, length range $[L_{\min}, L_{\max}]$, sampling budget N
Ensure: Set of unique sampled paths Π

Initialize $\nu(\sigma) \leftarrow 0$ for all $\sigma \in \Omega$, $\mu(\kappa) \leftarrow 0$ for all $\kappa \in \mathcal{K}$, $\Pi \leftarrow \emptyset$, $\mathcal{S} \leftarrow \emptyset$
for $b = 1, \ldots, N$ **do**
  Sample $\sigma_0 \in \Omega$ with probability $\propto (\nu(\sigma) + 1)^{-1}$
  $\mathcal{P} \leftarrow (\sigma_0)$, $\mathcal{V}_\kappa \leftarrow \emptyset$, $\mathcal{V}_\sigma \leftarrow \{\sigma_0\}$, $l \leftarrow 0$
  while $l < L_{\max}$ **do**
    $\mathcal{N}(\sigma_l) \leftarrow \{\kappa \in \mathcal{K} \ : \ \kappa\colon \sigma_l \to \sigma'$ for some $\sigma' \in \Omega \setminus \mathcal{V}_\sigma,\ \kappa \notin \mathcal{V}_\kappa\}$
    if $\mathcal{N}(\sigma_l) = \emptyset$ **then**
      break {dead-end; no monotone continuation}
    end if
    Sample $\kappa_{l+1} \in \mathcal{N}(\sigma_l)$ with probability $\propto (\mu(\kappa) + 1)^{-1}$
    Sample $\sigma_{l+1} \in \kappa_{l+1}.\text{post} \setminus \mathcal{V}_\sigma$ with probability $\propto (\nu(\sigma) + 1)^{-1}$
    Append $(\kappa_{l+1}, \sigma_{l+1})$ to $\mathcal{P}$
    $\mathcal{V}_\kappa \leftarrow \mathcal{V}_\kappa \cup \{\kappa_{l+1}\}$, $\mathcal{V}_\sigma \leftarrow \mathcal{V}_\sigma \cup \{\sigma_{l+1}\}$
    $l \leftarrow l + 1$
  end while
  if $L_{\min} \le l \le L_{\max}$ **and** $\text{skills}(\mathcal{P}) \notin \mathcal{S}$ **then**
    $\Pi \leftarrow \Pi \cup \{\mathcal{P}\}$; $\mathcal{S} \leftarrow \mathcal{S} \cup \{\text{skills}(\mathcal{P})\}$
    $\nu(\sigma) \leftarrow \nu(\sigma) + 1$ for each σ in $\mathcal{P}$
    $\mu(\kappa) \leftarrow \mu(\kappa) + 1$ for each κ in $\mathcal{P}$
  end if
end for
return Π

4. Experiments

4.1. Experimental Setup

We sample 3,721 paths from the constructed skill graph to synthesize task instances and validate the effectiveness of *SkillSynth*.

Base Model. We select the Qwen3 dense series (Yang et al., 2025a) (8B, 14B, and 32B) as our base models to study the effect of model scale, with Qwen3-32B serving as our primary model for ablation studies. All models are trained with full-parameter supervised fine-tuning for 5 epochs using AdamW, a peak learning rate of 2×10^{-5}, weight decay of 1×10^{-4}, a cosine schedule with 10% warmup, bfloat16 precision, and gradient clipping with maximum norm 1.0. We use a micro-batch size of 1 per GPU and gradient accumulation to reach the desired global batch size. Additional training details are provided in § B.

Evaluation. We use Terminal-Bench 1.0 and 2.0 (Merrill et al., 2026) as our evaluation benchmarks, comprising 80 and 89 community-curated tasks, respectively, with the latter serving as a harder successor to the former. All reported results are the mean accuracy over three independent runs, along with 95% confidence intervals. We adopt Terminus 2 as our agent scaffold, which interacts with the environment solely through a single headless terminal without additional harness constraints. For evaluation infrastructure, we employ Harbor (Harbor Framework Team, 2026), the official orchestration framework of Terminal-Bench 2.0, to parallelize trajectory sampling across 128 concurrent Docker (Merkel, 2014) environments for enhanced sampling efficiency.

4.2. Multi-agent Harness Quality

The multi-agent harness achieves high yield. As shown in Table 1, 95.7% of synthesized instances pass the oracle check, with 92.0% passing both quality checks. The multi-turn verify-then-repair loop recovers 721 task instances that failed in the first round, which shows that interactive repair is necessary to achieve high yield at scale. In total, a single fully automatic run produces 3,560 usable task instances. Scaling further simply requires sampling more paths from the skill graph and re-running the multi-agent harness.

Misalignment between instructions and tests dominates rubric failures. Of failed rubric checks, 77% stem from test scripts that either over-specify or under-specify relative to the instruction, potentially producing inaccurate evaluation of agent trajectories and erroneous reward signals for reinforcement learning. We retain these task instances for supervised fine-tuning to preserve trajectory diversity, but discard them for reinforcement learning to avoid erroneous reward signals.

Poor first-round generations are hard to recover. For oracle failures, the dominant cause is corrupted filesystem snapshots: due to the randomness of LLM generation, the first synthesis round may produce buggy stages that remain unrecoverable even after multiple repair cycles. Although the multi-agent architecture substantially outperforms a single-stage pipeline, these cases suggest a floor on repair effectiveness. Re-running failed paths with a higher sampling temperature is a simple yet effective strategy to help recover these paths and preserve the diversity of the synthesized task set.

Task difficulty distribution. We categorize the 3,560 usable task instances by difficulty using Hy3 Preview (Tencent Hy Team, 2026). Each task is attempted three times, and we assign difficulty levels based on the number of successful attempts. Table 2 reports the distribution. 37% of tasks fall in the learnable range (1/3 or 2/3 success rate), while 38% (0/3) represent potentially challenging problems for future exploration.

4.3. Main Results

For trajectory collection, we use MiniMax M2.7 as the teacher model due to its cost efficiency, sampling three trajectories per task instance for a total of 10,680 trajectories. We retain both successful and failed trajectories for training (Pi et al., 2026) to preserve diversity and coverage, as failed trajectories contain useful reasoning patterns for agentic problem-solving, such as error diagnosis and recovery strategies across diverse scenarios.

Table 3 reports experimental results after training Qwen3-8B, Qwen3-14B, and Qwen3-32B on collected trajectories. All three models improve over their respective baselines, with gains scaling with model size. Qwen3-32B + SS outperforms the larger Qwen 3 Coder 480B on TB 2.0 despite using 15× fewer parameters, suggesting that targeted data construction and domain-specific training can effectively improve the terminal agentic capabilities of smaller models. The gain is most pronounced on TB 2.0, where tasks require longer-horizon environment manipulation and benefit more from trajectories synthesized from compositional skill paths.

4.4. Ablation Study

To isolate the contributions of skill graph construction and path sampling, we compare *SkillSynth* against two baselines that use the same multi-agent harness for task synthesis. We first construct a skill pool from all skills appearing in the sampled paths, then create: 1) Single-skill: 3,721 skills randomly drawn from the pool, each used as a standalone seed for task synthesis. 2) Multi-skill: 3,721 randomly composed combinations of 2–7 skills from the pool, without graph-guided ordering.

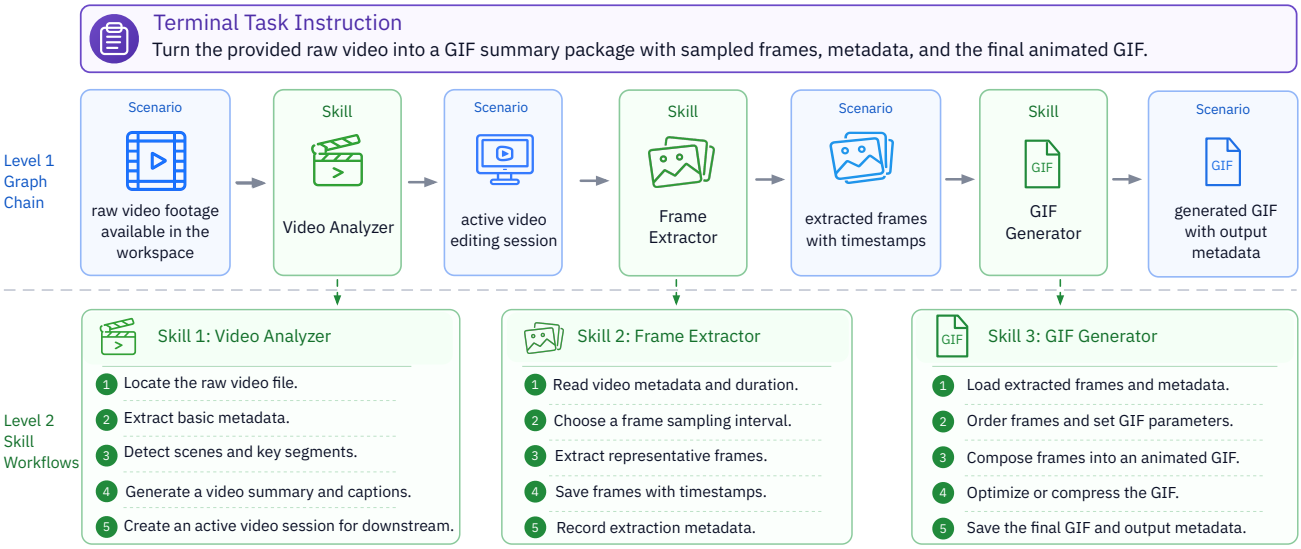

Figure 4. A video-domain example of a path sampled from the skill graph and the corresponding synthesized task instruction. To accomplish the task, the agent needs to apply multiple skills, each of which expands into a multi-step internal workflow.

Table 1. Summary of multi-agent harness outcomes and repair statistics.

Task Verification Results		
Category	**Count**	**Ratio (%)**
All passed	3,423	92.0
Oracle passed only	137	3.7
Failed	161	4.3
Total	3,721	100.0
Repair Statistics		
Avg. repair cycles	2.31	
Avg. tool calls	11	
Recovered tasks	721	

Table 2. Difficulty distribution of the 3,560 usable task instances. Lower success counts indicate harder tasks.

Successes (out of 3)	Percentage	# Tasks
0/3	38%	1,352
1/3	18%	637
2/3	19%	679
3/3	25%	892
Total	100%	3,560

Table 4 reports both task difficulty and downstream model performance. *SkillSynth* tasks are harder than both baselines. On the training side, models trained on *SkillSynth* trajectories outperform those trained on single-skill trajectories by 8.4 points on TB 1.0 and 8.3 points on TB 2.0, and outperform multi-skill trajectories by 3.0 points on TB 1.0 and 3.8 points on TB 2.0.

Random composition lacks workflow coherence. We find multi-skill baselines produce lower-quality tasks because randomly composed skills lack sequential dependencies and cannot form coherent workflows. The multi-agent harness tends to generate simplified task instances that contain multiple fine-grained requirements but require few execution steps.

4.5. Diversity Analysis

We analyze diversity at two levels: 1) the structural diversity of the skill graph itself and sampled paths, and 2) the diversity of the collected trajectories.

Skill graph and sampled paths. The constructed skill graph spans a wide range of domains, including long-tail areas such as Audio & Speech and 3D & Simulation (see § C for the full domain distribution). The graph is highly connected at the macro level: 85.6% of scenarios belong to the largest weakly connected component, while 6,251 smaller components capture specialized, self-contained workflows. From the whole graph, we enumerate 16,632,220 paths requiring seven or more skills, confirming a vast combinatorial space for task synthesis. Algorithm 1 ensures that the sampled paths are diverse at the skill and scenario levels.

Trajectories. To quantify execution diversity, we randomly sample 1,000 trajectories separately and use DeepSeek Reasoner (v3.2) to extract the scenarios and skills encountered by the agent during execution. Using the same prompt and model across all strategies ensures statistically comparable extraction granularity. We then embed all extracted sce-

Table 3. Experimental results on Terminal-Bench 1.0 and 2.0 with the Terminus 2 scaffold. +SS denotes models trained with *SkillSynth* generated trajectories.

Proprietary			Open-Source		
Model	**TB 1.0**	**TB 2.0**	**Model**	**TB 1.0**	**TB 2.0**
GPT-5.3-Codex	–	64.7 ± 2.7	GLM 5	–	52.4 ± 2.6
Claude Opus 4.6	–	62.9 ± 2.7	Kimi K2.5	–	43.2 ± 2.9
Claude Opus 4.5	–	57.8 ± 2.5	MiniMax m2.5	–	42.2 ± 2.6
Gemini 3 Pro	–	56.9 ± 2.5	DeepSeek-V3.2	–	39.6 ± 2.8
GPT-5.2	–	54.0 ± 2.9	Kimi K2 Thinking	–	35.7 ± 2.8
Claude Sonnet 4.5	51.0 ± 1.6	42.8 ± 2.8	GLM 4.7	–	33.4 ± 2.8
GPT-5.1	–	47.6 ± 2.8	Qwen 3 Coder 480B	–	23.9 ± 2.8
Claude Opus 4.1	43.8 ± 2.8	38.0 ± 2.6			
Claude Haiku 4.5	41.8 ± 2.6	28.3 ± 2.9			
GPT-5	41.3 ± 2.2	35.2 ± 3.1	**Ours**		
Claude Opus 4	39.0 ± 0.8	–	Qwen3-8B + SS	17.1 ± 1.8	13.5 ± 2.8
Claude Sonnet 4	36.4 ± 1.2	–	Qwen3-14B + SS	22.9 ± 1.8	19.9 ± 1.6
GPT-5-Mini	30.8 ± 3.9	24.0 ± 2.5	Qwen3-32B + SS	33.8 ± 3.1	29.6 ± 1.6

Table 4. Ablation study on synthesis seed selection strategy.

	Task Distribution				Accuracy	
Strategy	0/3	1/3	2/3	3/3	TB 1.0	TB 2.0
Single-skill	16%	23%	34%	27%	25.4 ± 1.8	21.3 ± 2.8
Multi-skill	27%	24%	21%	28%	30.8 ± 1.8	25.8 ± 2.8
SkillSynth (Ours)	38%	18%	19%	25%	33.8 ± 3.1	29.6 ± 1.6

narios and skills using Harrier-OSS-v1-27B and perform clustering-based deduplication to obtain unique scenarios, skills, and scenario-skill pairs. Trajectories sampled from *SkillSynth* tasks exhibit 31% higher unique scenario-skill coverage than single-skill baselines and 19% higher than randomly composed multi-skill baselines on average. The improved diversity translates directly into higher data efficiency and performance, as shown in Table 4. Prompts are provided in § D.

4.6. Error Analysis

To understand failure modes, three annotators independently analyzed 20 failed trajectories each. Based on the results, we design an error analysis skill for an LLM agent, which then analyzes all failed trajectories accordingly. As shown in Table 5, the dominant failure modes are partial implementation (42.2%) and over-trust in inline self-test (29.5%). In both cases, the agent substitutes self-narrated verification for specification-grounded testing and confidently marks the task complete. These findings suggest that terminal agents require stronger alignment with task instructions and more flexible exploration strategies during execution.

4.7. Discussion and Future Work

SkillSynth synthesizes a large number of task instances that reflect real-world workflows, covering diverse scenarios and skills. Future work includes 1) scaling *SkillSynth* to larger data regimes while continuously refining the multi-agent harness to adapt to more cost-effective models, 2) sampling subgraphs rather than chains from the skill graph, which requires parallel execution of multiple skills and increases task complexity.

Limitations. Our current graph sampler focuses on linear paths, which matches many terminal workflows but cannot fully capture tasks with branching plans, parallel subtasks, or multiple independent artifacts that must later be reconciled. The multi-agent harness also relies on automatic oracle and rubric checks, so its quality is bounded by the coverage of generated tests and the reliability of judge feedback. In practice, the most useful failures are not random invalid tasks but partially specified tasks whose tests miss one requirement; these cases motivate tighter coupling between instruction generation, oracle construction, and adversarial test augmentation. Another limitation is that our graph is built from available public skills, which provides broad coverage but may underrepresent proprietary operational workflows. Expanding the skill pool with organization-specific skills would make the same synthesis pipeline applicable to specialized engineering environments.

Scaling direction. The results suggest that skill-graph structure is useful not only for synthesizing tasks, but also for scheduling future data collection. Because each sampled path exposes the model to a known combination of scenarios

Table 5. Error analysis of failed trajectories.

Error Category	Ratio	Typical Example
Partial Implementation	42.2%	Agent runs `pytest` on self-generated tests, gets all tests passing, and submits but silently skips one or more clauses from the instruction.
Inline Self-Test Over-trust	29.5%	Agent never runs `pytest`; instead verifies with hand-picked `python3 -c` assertions on happy paths, missing edge cases.
Premature Termination	12.9%	Agent finishes writing code and exits without running any form of verification.
API/Flag Hallucination	7.0%	Agent repeatedly invokes non-existent modules or flags, triggering $\geq$3 errors such as `No module named` or `unexpected kwarg`.
Debug Fixation Loop	5.1%	Agent repeats the same command $\geq$10 times with minor edits, never switching strategy, and exhausts the episode budget.
Error Rationalization	3.3%	Agent observes failing tests but rationalizes them as "pre-existing" or "flaky" and completes anyway.

and skills, the graph can support curriculum construction, targeted oversampling of weak regions, and continual refresh as new skills appear. This points to a closed-loop version of *SkillSynth* in which failure analysis identifies undertrained graph regions, the sampler proposes new paths around those regions, and the harness synthesizes additional verified tasks for the next training round.

5. Related Work

Terminal Agents. Recent work has increasingly explored LLMs as terminal agents that interact with real-world computing systems through command-line interfaces (Yang et al., 2023; Jimenez et al., 2024). To evaluate such capabilities, Terminal-Bench (Merrill et al., 2026) provides a set of hand-crafted tasks spanning diverse domains, requiring agents to complete end-to-end workflows within containerized Docker environments. Meanwhile, a series of agent scaffolds (Anthropic, 2025; OpenAI, 2025; Google, 2025; Wang et al., 2025b; JetBrains, 2025) have been developed to enhance LLMs' planning, execution, and tool calling capabilities in terminal settings. Despite strong proprietary results, open-source models remain substantially behind, motivating scalable synthesis of diverse terminal execution trajectories for effective training.

Synthetic Data for Terminal Agents. To reduce the cost of curating terminal training data, prior efforts scale synthetic task instances along different dimensions. Some approaches expand domain coverage through LLM-generated taxonomies, such as Endless Terminals (Gandhi et al., 2026), TermiGen (Zhu et al., 2026), and Nemotron-Terminal (Pi et al., 2026), enabling task instance generation across diverse domains. Others scale the number of task instances, for example by collecting real-world Docker environments from GitHub (Wu et al., 2026), or by inverting healthy environments into buggy states to derive new task instances (Lin et al., 2026). While these approaches increase the number of task instances, they provide limited control over the diversity of execution trajectories experienced by agents during task solving. We address this limitation by synthesizing task instances from a skill graph, enabling explicit control over the trajectories encountered during training.

Skill Topology. A separate line of work organizes reusable agent skills into structured topologies, following the reusable skill library introduced by Voyager (Wang et al., 2023) and drawing on large skill repositories such as ClawHub (OpenClaw, 2026). AgentSkillOS (Li et al., 2026) organizes skills into a hierarchical capability tree and composes them through DAG-based orchestration graphs, while SkillNet (Liang et al., 2026) models skills as nodes in a relational graph with explicit inter-skill connections. However, these approaches primarily focus on skill organization and retrieval, rather than modeling how skills are composed during execution. In contrast, we organize skills through scenarios, where skill transitions are induced by shared executable states and temporal ordering in trajectories, enabling structured synthesis of terminal tasks with controllable scenario and skill diversity.

6. Conclusion

In this paper, we model agentic trajectories as sequences of scenarios and skills to analyze their diversity, and propose *SkillSynth*, a framework that synthesizes diverse terminal task instances by sampling compositional paths from a constructed skill graph. *SkillSynth* achieves a high synthesis pass rate at relatively low cost, enabling scalable production of verified task instances. Experimental results show that training effectiveness depends on trajectory diversity, not merely on task volume. As communities like ClawHub continue to grow, *SkillSynth* provides a scalable way to transform emerging workflows into diverse task instances, enabling models to continuously learn real-world experience.

Impact Statement

This work aims to improve terminal agents by synthesizing verified task instances from public skills and repositories. The skills used to construct our data are sourced from publicly available GitHub repositories and ClawHub, and are used in accordance with their respective open-source licenses. More capable terminal agents may also increase the risk of unsafe automation if deployed without appropriate sandboxing, permission controls, and monitoring; our synthesis and evaluation pipeline therefore emphasizes containerized execution and verification.

References

Anthropic. Claude code: Best practices for agentic coding. https://www.anthropic.com/engineering/claude-code-best-practices, Apr 2025.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, October 2008. ISSN 1742-5468. doi: 10.1088/1742-5468/2008/10/p10008. URL http://dx.doi.org/10.1088/1742-5468/2008/10/P10008.

Chen, M., Zhang, L., Feng, Y., Wang, X., Zhao, W., Cao, R., Yang, J., Chen, J., Li, M., Ma, Z., Ge, H., Zhang, Z., Cui, Z., Liu, D., Zhou, J., Sun, J., Lin, J., and Hui, B. Swe-universe: Scale real-world verifiable environments to millions, 2026. URL https://arxiv.org/abs/2602.02361.

Gandhi, K., Garg, S., Goodman, N. D., and Papailiopoulos, D. Endless terminals: Scaling rl environments for terminal agents, 2026. URL https://arxiv.org/abs/2601.16443.

Google. Gemini cli: Your open-source ai agent. https://blog.google/technology/developers/gemini-cli-open-source-ai-agent/, 2025.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments, January 2026. URL https://github.com/harbor-framework/harbor.

JetBrains. Junie cli: Llm-agnostic coding agent built for real-world development. https://junie.jetbrains.com/, 2025.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. Swe-bench: Can language models resolve real-world github issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Johnson, S. C. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967. doi: 10.1007/BF02289588.

Li, H., Mu, C., Chen, J., Ren, S., Cui, Z., Zhang, Y., Bai, L., and Hu, S. Organizing, orchestrating, and benchmarking agent skills at ecosystem scale, 2026. URL https://arxiv.org/abs/2603.02176.

Liang, Y., Zhong, R., Xu, H., Jiang, C., Zhong, Y., Fang, R., Gu, J.-C., Deng, S., Yao, Y., Wang, M., Qiao, S., Xu, X., Wu, T., Wang, K., Liu, Y., Bi, Z., Lou, J., Jiang, Y. E., Zhu, H., Yu, G., Hong, H., Huang, L., Xue, H., Wang, C., Wang, Y., Shan, Z., Chen, X., Tu, Z., Xiong, F., Xie, X., Zhang, P., Gui, Z., Liang, L., Zhou, J., Wu, C., Shang, J., Gong, Y., Lin, J., Xu, C., Deng, H., Zhang, W., Ding, K., Zhang, Q., Huang, F., Zhang, N., Pan, J. Z., Qi, G., Wang, H., and Chen, H. Skillnet: Create, evaluate, and connect ai skills, 2026. URL https://arxiv.org/abs/2603.04448.

Lin, X. V., Wang, C., Zettlemoyer, L., and Ernst, M. D. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system, 2018. URL https://arxiv.org/abs/1802.08979.

Lin, Y., Wang, H., Wu, S., Fan, L., Pan, F., Zhao, S., and Tu, D. Cli-gym: Scalable cli task generation via agentic environment inversion, 2026. URL https://arxiv.org/abs/2602.10999.

Merkel, D. Docker: Lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014 (239):2, 2014.

Merrill, M. A., Shaw, A. G., Carlini, N., Li, B., Raj, H., Bercovich, I., Shi, L., Shin, J. Y., Walshe, T., Buchanan, E. K., Shen, J., Ye, G., Lin, H., Poulos, J., Wang, M., Nezhurina, M., Jitsev, J., Lu, D., Mastromichalakis, O. M., Xu, Z., Chen, Z., Liu, Y., Zhang, R., Chen, L. L., Kashyap, A., Uslu, J.-L., Li, J., Wu, J., Yan, M., Bian, S., Sharma, V., Sun, K., Dillmann, S., Anand, A., Lanpouthakoun, A., Koopah, B., Hu, C., Guha, E., Dreiman, G. H. S., Zhu, J., Krauth, K., Zhong, L., Muennighoff, N., Amanfu, R., Tan, S., Pimpalgaonkar, S., Aggarwal, T., Lin, X., Lan, X., Zhao, X., Liang, Y., Wang, Y., Wang, Z., Zhou, C., Heineman, D., Liu, H., Trivedi, H., Yang, J., Lin, J., Shetty, M., Yang, M., Omi, N., Raoof, N., Li, S., Zhuo, T. Y., Lin, W., Dai, Y., Wang, Y., Chai, W., Zhou, S., Wahdany, D., She, Z., Hu, J., Dong, Z., Zhu, Y., Cui, S., Saiyed, A., Kolbeinsson, A., Hu, J., Rytting, C. M., Marten, R., Wang, Y., Dimakis, A., Konwinski, A., and Schmidt, L. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026. URL https://arxiv.org/abs/2601.11868.

OpenAI. Introducing codex. https://openai.com/index/introducing-codex/, May 2025.

OpenClaw. Clawhub: Open skill registry for AI agents, 2026. URL https://clawhub.ai.

Pi, R., Lam, G., Shoeybi, M., Jannaty, P., Catanzaro, B., and Ping, W. On data engineering for scaling llm terminal capabilities, 2026. URL https://arxiv.org/abs/2602.21193.

Sutton, R. S., Precup, D., and Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2): 181–211, 1999.

Team, O.-A. OpenThoughts-Agent. https://www.openthoughts.ai/blog/agent, December 2025.

Tencent Hy Team. Hy3 preview: The first step in rebuilding the hy model, 2026. URL https://hy.tencent.com/research/hy3.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An open-ended embodied agent with large language models, 2023. URL https://arxiv.org/abs/2305.16291.

Wang, L., Ramalho, L., Celestino, A., Pham, P. A., Liu, Y., Sinha, U. K., Portillo, A., Osunwa, O., and Maduekwe, G. Swe-bench++: A framework for the scalable generation of software engineering benchmarks from open-source repositories, 2025a. URL https://arxiv.org/abs/2512.17419.

Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J., Tran, H. H., Li, F., Ma, R., Zheng, M., Qian, B., Shao, Y., Muennighoff, N., Zhang, Y., Hui, B., Lin, J., Brennan, R., Peng, H., Ji, H., and Neubig, G. Openhands: An open platform for ai software developers as generalist agents, 2025b. URL https://arxiv.org/abs/2407.16741.

Wu, S., Li, Y., Song, Y., Zhang, W., Wang, Y., Batista-Navarro, R., Yang, X., Tang, M., Dai, B., Yang, J., and Lin, C. Large-scale terminal agentic trajectory generation from dockerized environments, 2026. URL https://arxiv.org/abs/2602.01244.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

Yang, J., Prabhakar, A., Narasimhan, K., and Yao, S. Intercode: Standardizing and benchmarking interactive coding with execution feedback, 2023. URL https://arxiv.org/abs/2306.14898.

Yang, J., Lieret, K., Jimenez, C. E., Wettig, A., Khandpur, K., Zhang, Y., Hui, B., Press, O., Schmidt, L., and Yang, D. Swe-smith: Scaling data for software engineering agents, 2025b. URL https://arxiv.org/abs/2504.21798.

Zhang, L., He, S., Zhang, C., Kang, Y., Li, B., Xie, C., Wang, J., Wang, M., Huang, Y., Fu, S., Nallipogu, E., Lin, Q., Dang, Y., Rajmohan, S., and Zhang, D. Swe-bench goes live!, 2025. URL https://arxiv.org/abs/2505.23419.

Zhu, K., Nie, Y., Li, Y., Huang, Y., Wu, J., Liu, J., Sun, X., Yin, Z., Wang, L., Liu, Z., Barsoum, E., Wang, W. Y., and Guo, W. Termigen: High-fidelity environment and robust trajectory synthesis for terminal agents, 2026. URL https://arxiv.org/abs/2602.07274.

A. Skill Graph Construction Discussion

All LLM calls use DeepSeek Reasoner (v3.2) to ensure extraction quality and reliability. We also explored several alternative graph construction strategies, including embedding-based scenario alignment and single-pass subgraph generation; we found that the LLM-based pairwise alignment approach yields higher-quality graphs.

We cluster scenarios using a two-stage scalable hierarchical procedure. First, we construct a sparse semantic similarity graph over normalized scenario embeddings and apply Louvain community detection (Blondel et al., 2008) to obtain coarse buckets. Then, within each bucket, we run complete-linkage agglomerative clustering (Johnson, 1967) with cosine distance to avoid the quadratic memory cost of global hierarchical clustering while preserving the desirable property of complete linkage: every pair within a cluster must be mutually close, preventing chain-drift artifacts where transitively linked but semantically distant scenarios get merged (e.g., A is close to B and B is close to C, but A and C are semantically different.).

For the clustering hyperparameters, we sweep the agglomerative distance threshold on held-out scenario samples and manually inspect the resulting clusters. We choose the final threshold by balancing merge quality and over-fragmentation: the selected value should merge clear paraphrases and near-equivalent states while keeping semantically distinct states, especially negations and pre/post condition changes, in separate clusters.

B. Supervised Fine-Tuning

We perform supervised fine-tuning with AdamW, using $\beta_1 = 0.9$ and $\beta_2 = 0.95$, together with a cosine learning rate schedule and a warmup ratio of 10%. The peak learning rate is set to 2×10^{-5} and the weight decay is 1×10^{-4} unless otherwise noted. We train all models for 5 epochs in bfloat16 precision. To stabilize optimization on long terminal trajectories, we apply gradient clipping with a maximum norm of 1.0. Training uses a micro-batch size of 1 per GPU and gradient accumulation to achieve the desired global batch size.

C. Skill Graph Analysis

Table 6 summarizes the structural properties of the skill graph $\mathcal{G}$ constructed in § 3.2.

The degree distribution (Figure 6) is heavy-tailed: the median node degree is 2, while a small number of hub scenarios reach degree 752, corresponding to generic intermediate states compatible with many skills. This motivates the inverse-frequency sampling in § 3.3. The connected component distribution (Table 6, right) is dominated by a single giant component of 118,806 nodes (85.6%), confirming that the cross-skill alignment stage chains the majority of skills into a single traversable subgraph. Figure 5 shows that the graph covers both common domains, such as coding agents, document processing, DevOps, and security, as well as long-tail domains, such as audio/speech, 3D simulation and IoT/hardware workflows.

D. Prompts

The prompts used to extract skills and scenarios from trajectories, and to cluster embeddings to obtain unique instances, are provided below. We use DeepSeek Reasoner (v3.2) for extraction and Microsoft/Harrier-OSS-v1-27B for embedding.

Listing 1. Prompt: Trajectory Analysis

```
You are analyzing a terminal agent
trajectory to extract its semantic
structure.

TRAJECTORY:
{{{observation_action_sequence}}}

Your task: Segment this trajectory into a
sequence of (scenario, skill) pairs, where
:
- A SCENARIO is the semantic state of the
environment at a decision point
  (e.g., "Python project with failing unit
 tests", "empty Git repository",
  "server container with missing
dependency").
- A SKILL is a coherent multi-step
workflow the agent executes to transition
  from one scenario to the next (e.g., "
run pytest and capture output",
  "install missing package via pip", "
create a Dockerfile for the service").

Output a JSON array of pairs:
```json
[
  {
    "step_range": [start_index, end_index
],
    "scenario": "<brief semantic
description of the state before this skill
>",
    "skill": "<brief description of the
workflow executed in this range>"
  },
  ...
]
```

Keep scenario and skill descriptions under
 15 words each.
Return ONLY the JSON array, no other text.
```

Listing 2. Prompt: Embedding Scenarios

Table 6. Left: Structural statistics of the skill graph $\mathcal{G}$. Right: Connected component size distribution.

Metric	Value
Scenario nodes $\lvert\mathcal{S}\rvert$	82,073
Skill-labeled transitions $\lvert\mathcal{E}\rvert$	57,214
Source-only scenarios	18,749 (22.8%)
Sink-only scenarios	12,299 (15.0%)
Bridge scenarios	46,699 (56.9%)
Mean / Median / Max degree	4.32 / 2 / 752
Connected components	6,251
Giant component	118,806 (85.6%)

Component size	Count
2	482
3–5	5,561
6–10	183
11–50	24
>50	1
Total	**6,251**

```
Instruct: Retrieve scenarios that describe
 the same real-world condition.
Query: {{{scenario}}}
```

Listing 3. Prompt: Embedding Skills

```
Instruct: Retrieve skills that perform the
 same workflow.
Query: {{{skill}}}
```

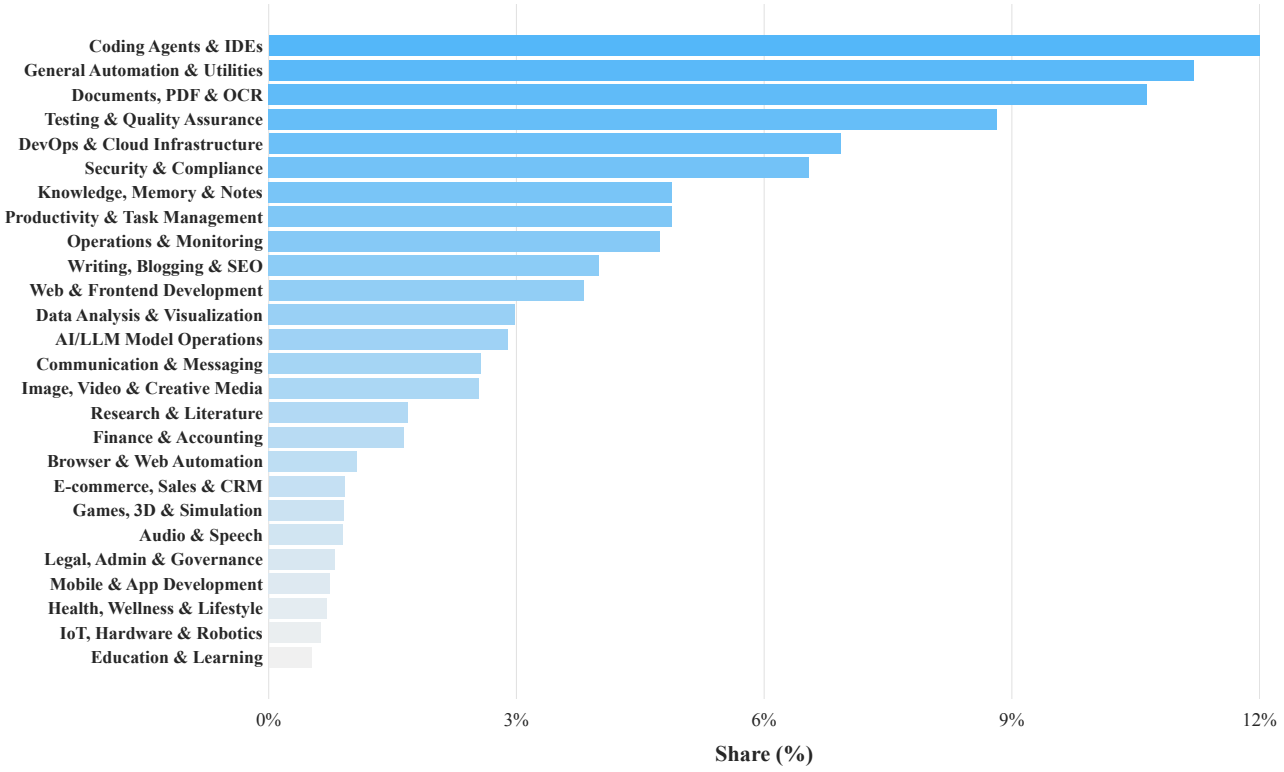

Figure 5. Skill category distribution.

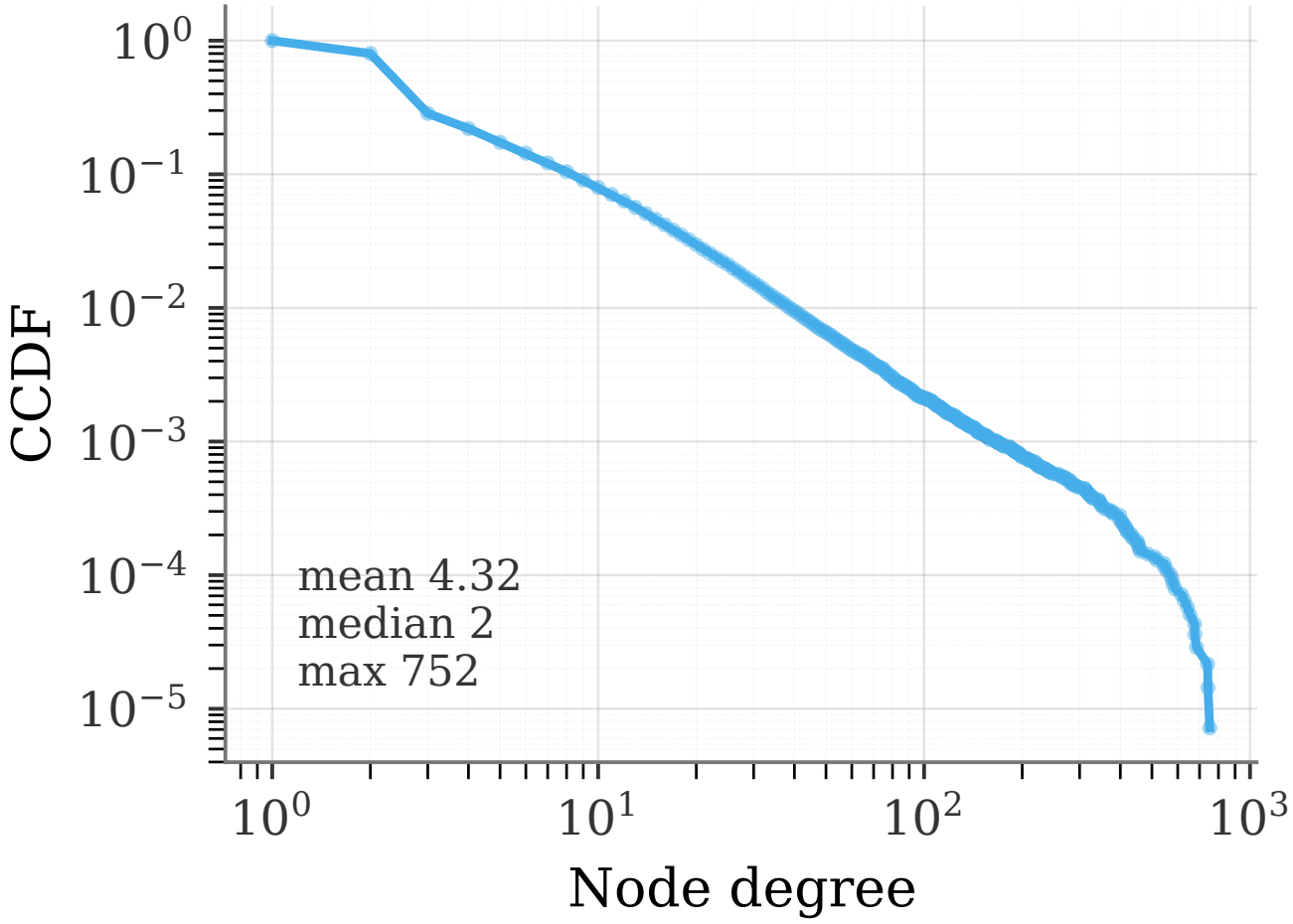

Figure 6. Degree distribution of the skill graph.