WORKBORNE: DISCOVERING AGENTIC TASKS FROM REAL-WORLD ARTIFACT TRANSITIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Developing terminal-based agents for long-horizon execution demands synthetic tasks that tightly couple complex instructions with authentic, interactive environments. However, current automated synthesis suffers from a fundamental asymmetry: *instruction-oriented* methods fabricate low-fidelity, synthetic environments, whereas *resource-oriented* methods treat real artifacts merely as static inputs, degrading task instructions into shallow, trivial operations. To address this limitation, we introduce **WorkBorne**, a synthesis framework that uses *artifact transitions* as the central primitive for modeling the full life-cycle of agentic tasks. In **WorkBorne**, each transition specifies who performs the task, which artifacts anchor the initial state and intended outcome, and the corresponding relationship between them. Specifically, WorkBorne first constructs coherent and diverse artifact corpora from iterative web retrieval, then discovers non-trivial transitions by identifying meaningful input-output artifact assemblies and the relations that connect them. Each discovered transition is finally instantiated into grounded agentic tasks. We construct 1,061 artifact corpora and 7,882 task environments, yielding 15,485 high-quality trajectories, with 53.6% exceeding 30 assistant turns. Experiments on Qwen3.5-27B and Qwen3.5-35B-A3B show that **WorkBorne** substantially improves over base models (i.e., improving Terminal-Bench 2.1 from 43.1% to 61.8% and ClawEval General score from 75.0 to 80.2) and achieves the strongest overall performance among other synthesis baselines (e.g., TMax (Ivison et al., 2026), RST (Li et al., 2026a), Terminal-Universe (Wu et al., 2026a)). Further analysis shows that transition is strongly associated with execution depth, while corpus breadth and transition depth provide complementary scaling gains.

1 INTRODUCTION

As terminal-based agents tackle increasingly complex professional tasks (Merrill et al., 2026; Ye et al., 2026), training and evaluating them heavily rely on high-quality trajectories that capture sustained, long-horizon interaction (Yang et al., 2026c; Wu et al., 2026b). Unlike conversational data, such trajectories emerge from agents executing task instructions within complex interactive environments (Cheng et al., 2026; Shi et al., 2026). However, tasks that genuinely induce long-horizon execution remain exceptionally rare (Li et al., 2026a;b). Most daily tasks are inherently shallow, and human instructions are often designed to minimize interaction steps (Han et al., 2026), leaving limited opportunities to elicit iterative reasoning, execution, and verification. Therefore, the central challenge in agentic data synthesis is to discover tasks whose instructions and environments jointly induce sustained interaction.

Existing automated synthesis methods generally fail to produce long-horizon tasks due to an instruction-environment asymmetry in their design. **Instruction-oriented methods** begin with a generated task instruction and use language models to reconstruct an environment (Zhu et al., 2026; Cheng et al., 2026). Although this guarantees clear instruction framing, synthesized artifacts often fail to capture the richness, coherence, and incidental complexity of real-world workspaces (Shi et al., 2026; Tang et al., 2026). **Resource-oriented methods** instead discover tasks from APIs (Liu et al., 2024), web resources (Liu et al., 2025), or tool interactions (Dong et al., 2026; Wu et al., 2026a). These methods provide authentic workspaces, but treat the artifacts merely as passive task input, making it easy to generate shallow instructions like reading a file and producing a brief sum-

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.

mary (Hua et al., 2026). In short, this asymmetry prevents existing paradigms from combining authentic workspace richness with instruction complexity to drive sustained agent interaction.

In this work, we observe that authentic agentic workflows naturally follow a simple **3W** primitive: *who* works on *what input* toward *what output*. In real-world professional environments, both the initial inputs and the intended outputs are grounded in concrete artifacts (i.e., specific files or digital items), while the substantive task lies in the transition connecting them. This perspective motivates us to treat **artifact transitions** as a unified abstraction to jointly model task instructions and environments, and we then propose **WorkBorne**, a fully automated framework that discovers agentic tasks from real-world artifact transitions. Specifically, WorkBorne first constructs coherent and diverse artifact corpora through iterative web retrieval, providing rich materials from authentic workflows. To discover non-trivial transitions, WorkBorne selects personas for each corpus, uses them to assemble grounded input-output artifact anchors, and then infers the required transition by reasoning how the inputs must be adapted, extended, or integrated to realize the anchored output. Finally, each validated transition is instantiated into aligned task instructions, executable environments, and evaluation rubrics, naturally ensuring them remain grounded in the same workspace.

Using WorkBorne, we construct 1,061 artifact corpora and 7,882 executable tasks, yielding 15,485 high-quality trajectories across 14 agent harnesses, with 53.6% exceeding 30 turns. Fine-tuning Qwen3.5-27B and Qwen3.5-35B-A3B (Qwen Team, 2026) on these trajectories (yeilding **WorkBorne-27B** and **WorkBorne-35B-A3B**) achieves substantial gains across benchmarks, with WorkBorne-27B boosting Terminal-Bench 2.1 (Merrill et al., 2026) accuracy from 43.1% to 61.8% and ClawEval General (Ye et al., 2026) score from 75.0 to 80.2. A controlled construction ablation (§5.1) shows that transition modeling increases average execution length from 27.8 to 44.5 assistant turns over an input-only pipeline, while improving Terminal-Bench 2.1 from 48.3% to 51.7%. Building on this, an empirical investigation into structural execution factors (§5.2) shows that execution depth depends little on raw artifact quantity ($\eta = 0.015$), but strongly on transition structure. Input-output artifact assembly shows a much stronger association ($\eta = 0.314$), with output-anchor form more informative than input form ($\eta = 0.306$ vs. 0.183). Finally, scaling studies (§5.3) further show that corpus breadth and transition depth provide complementary gains, while broader corpus coverage consistently performs better under matched training budgets.

2 Related Work

2.1 Terminal-Based Agents.

Terminal-based agents have evolved from interactive code execution (Yang et al., 2023) to repository-scale software engineering (Jimenez et al., 2024; Wang et al., 2024; Yang et al., 2024; Wang et al., 2025), and more recently toward open-ended, general-purpose command-line interaction (Merrill et al., 2026; Chen et al., 2026b). This trajectory has sparked parallel efforts to construct training environments from real codebases (Pan et al., 2025) and to optimize execution efficiency (Ren et al., 2026; Mavali et al., 2026). As agent development rapidly accelerates toward sustained, long-horizon execution (Desai et al., 2026; Yang et al., 2026b), the primary bottleneck has pivoted from agent architecture design to scalable data synthesis: specifically, generating task suites that scale in volume without compromising the structural complexity and execution fidelity these complex tasks demand.

2.2 Terminal Task Synthesis.

Existing task synthesis methodologies broadly fall into two complementary paradigms, each incurring a fundamental quality trade-off when scaled up. 1) **Instruction-oriented methods** adopt a top-down route, initiating synthesis from task specifications and constructing matching environments (Zhu et al., 2026; Gandhi et al., 2026; Ivison et al., 2026; Zhao et al., 2026), with subsequent extensions escalating complexity via compositional skill graphs (Fan et al., 2026; Cheng et al., 2026), recursive task expansion (Li et al., 2026a; Pan et al., 2026), and adversarial calibration (Meng et al., 2026). While effective at driving textual difficulty, these environments remain LM-fabricated, systematically lacking the *incidental complexity* inherent to authentic workspaces. 2) **Resource-oriented methods** take a bottom-up route by anchoring synthesis in real-world codebases or pre-existing execution traces (Yang et al., 2026a; Lin et al., 2026; Wu et al., 2026a). Al-

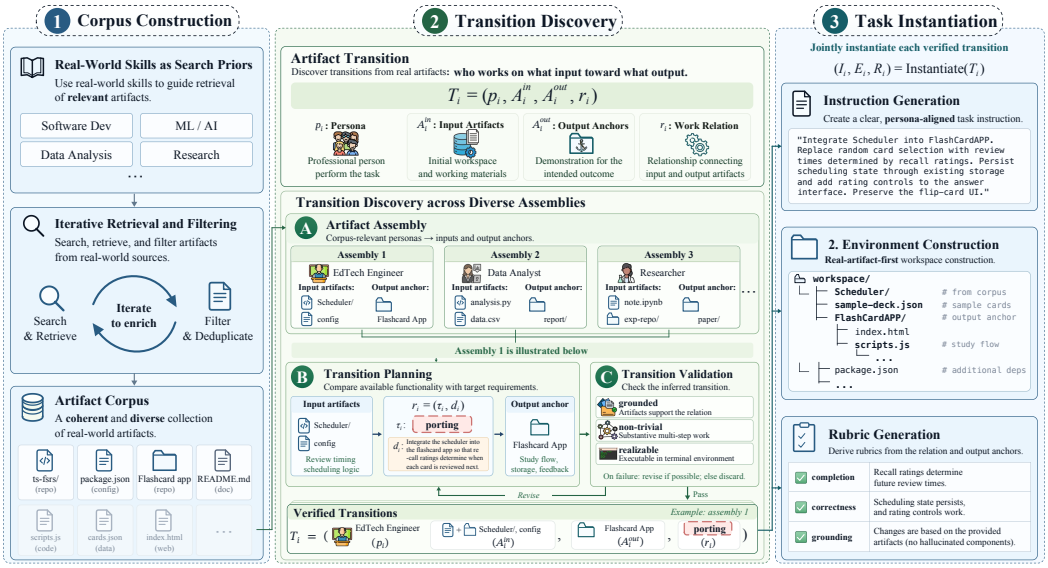


Figure 1: **Overview of WorkBorne.** WorkBorne models agentic tasks as artifact transitions $\mathcal{T} = (p, \mathcal{A}^{\text{in}}, \mathcal{A}^{\text{out}}, r)$ and synthesizes them in three stages: **(1) Artifact Corpus Construction:** building real-world artifact pools; **(2) Transition Discovery:** discovering grounded transitions through assembly, planning, and validation; and **(3) Task Instantiation:** compiling transitions into aligned instructions, executable environments, and evaluation rubrics.

though they provide realistic workspace ecosystems, they treat artifacts as static initial conditions, causing synthesized instructions to frequently degrade into shallow inspection operations (Pi et al., 2026; Yang et al., 2026c). Recent hybrid approaches seek to enforce instruction-environment alignment via executable-state grounding (Shi et al., 2026), real-material verification (Hua et al., 2026), and database-level grounding (Wang et al., 2026); yet, they continue to treat instructions and environments as decoupled synthesis targets.

3 WorkBorne

This section presents **WorkBorne**, a framework that models terminal-based agent tasks through unified artifact transitions. Below, we firstly formalize agentic tasks as state transitions over real-world artifact spaces (§3.1), detail the three-stage synthesis pipeline covering *corpus construction*, *transition discovery*, and *task instantiation* (§3.2), and conclude with an empirical profile of the generated dataset and execution trajectories (§3.3).

3.1 Task Formulation

A terminal-based agentic task is fundamentally an execution workflow driven by complex task instructions and authentic environments. At a high level, such a workflow follows a **3W** primitive: ***who*** works on ***what input*** toward ***what output***. Real-world artifacts anchor this process by physically defining both the initial workspace and the target outcome. However, existing synthesis paradigms often model only isolated parts of this workflow, either fabricating environments around generated instructions or treating real artifacts merely as static inputs, resulting in shallow instructions. To capture the task lifecycle holistically, **WorkBorne** instead jointly models these **3W** elements through an *artifact transition* $\mathcal{T}$, formalized as a four-tuple:

$$\mathcal{T}_i = \left(p_i, \mathcal{A}_i^{\text{in}}, \mathcal{A}_i^{\text{out}}, r_i\right), \tag{1}$$

where p_i denotes the persona performing the task, $\mathcal{A}_i^{\text{in}}$ denotes the set of input artifacts that form the initial workspace, $\mathcal{A}_i^{\text{out}}$ denotes a set of output anchors that demonstrate the intended outcome,

and r_i specifies the relationship between the input and output artifacts. With $\mathcal{T}_i$ serving as the task skeleton, WorkBorne realizes task creation by first discovering valid transitions from raw artifact space and then instantiating them into an aligned instruction, environment, and rubrics (§3.2).

3.2 TRANSITION-DRIVEN TASK SYNTHESIS

Artifact Corpus Construction. Synthesizing realistic agent tasks requires workspaces composed of artifacts that naturally co-occur in professional workflows, rather than arbitrary or isolated documents. To build such cohesive workspaces, WorkBorne uses *agent skills* (Anthropic, 2025) as retrieval seeds. Each skill defines a specific domain context (e.g., data engineering, API integration) and guides a search agent to gather a multi-format file collection, such as source code, raw datasets, and technical documentation, that logically belong to the same execution pipeline. We curate 1,061 high-quality skills from a public pool of over 10,000 skills to cover a wide spectrum of real-world scenarios. For each skill s_c, the search agent receives its workflow description, toolchain, and expected deliverables to formulate web queries and retrieve an initial artifacts collection. Under a fixed retrieval budget, the agent iteratively expands this collection by filtering duplicates and conducting targeted follow-up searches for missing or sparsely covered file formats (e.g., retrieving configuration files to complement a codebase). The resulting corpus $\mathcal{C}_c = \{A_1, \ldots, A_{N_c}\}$ forms a realistic multi-file workspace, where each A_i represents an individual artifact. Finally, artifacts in $\mathcal{C}_c$ are summarized into *file cards* G_c, which record file paths, metadata, key entities, and contents to provide a structured index for subsequent transition discovery.

Transition Discovery. Directly prompting an agent to imagine tasks from arbitrary artifacts often leads to shallow or repetitive operations (Hua et al., 2026). To discover substantive and diverse transitions, WorkBorne anchors task generation in *artifact transitions* that explicitly bridge concrete input artifacts to target output anchors within each workspace $\mathcal{C}_c$. Driven by the structured index G_c, this discovery process operates in three sequential steps:

(1) Artifact Assembly. Given an exploration budget N, this step forms N artifact assemblies $(p_i, \mathcal{A}_i^{\text{in}}, \mathcal{A}_i^{\text{out}})$ from each corpus $\mathcal{C}_c$. Specifically, we first randomly sample $2N$ candidate personas from PersonaHub (Ge et al., 2025) and use an agent to filter them down to the N personas most relevant to the domain context of $\mathcal{C}_c$. For each selected persona p_i, an agent inspects the file cards G_c to select two disjoint artifact subsets $\mathcal{A}_i^{\text{in}}, \mathcal{A}_i^{\text{out}} \subset \mathcal{C}_c$. The input set $\mathcal{A}_i^{\text{in}}$ naturally matches the persona's expertise, while the output anchor set $\mathcal{A}_i^{\text{out}}$ acts as a real-world demonstration of the intended outcome, specifying structural, formatting, or constraints that the final work must satisfy.

(2) Transition Planning. Given an assembly $(p_i, \mathcal{A}_i^{\text{in}}, \mathcal{A}_i^{\text{out}})$, this step infers the underlying transition relation r_i that connects the input artifacts to the target output anchors. The transition agent analyzes reusable content, functionality, or structure in $\mathcal{A}_i^{\text{in}}$, and reasons how these components must be adapted, extended, or integrated to realize $\mathcal{A}_i^{\text{out}}$. The resulting relation r_i consists of a open-ended operation tag and a detailed task description specifying the required execution path. For instance, given a review-scheduling library *ts-fsrs* as $\mathcal{A}_i^{\text{in}}$ and a flashcard study application *SimpleFlashCard* as $\mathcal{A}_i^{\text{out}}$, an educational technology engineer persona plans a non-trivial *porting* transition: adapting the scheduling logic to the application's study flow so that each card is reviewed at an appropriate time. Together, $\mathcal{T}_i = (p_i, \mathcal{A}_i^{\text{in}}, \mathcal{A}_i^{\text{out}}, r_i)$ establishes a complete artifact transition.

(3) Transition Validation. A dedicated validation agent evaluates whether the artifact assembly genuinely supports the inferred relation r_i, verifies that the task demands non-trivial multi-step execution rather than simple file copying, and confirms that it can be realized in a simulated environment. Transitions failing these criteria are either refined or discarded.

Task Instantiation. Once an artifact transition $\mathcal{T}_i$ is validated, WorkBorne jointly instantiates it into the three core components of an executable agent task: a task instruction $\mathcal{I}_i$, an execution environment $\mathcal{E}_i$, and evaluation rubrics $\mathcal{R}_i$:

$$(\mathcal{I}_i, \mathcal{E}_i, \mathcal{R}_i) = \text{Instantiate}(\mathcal{T}_i), \tag{2}$$

where all three components are grounded in the same transition $\mathcal{T}_i$ to guarantee mutual alignment and verifiability. Specifically, three specialized agent modules construct these components as follows:

(1) Instruction generation ($\mathcal{I}_i$). An instruction agent synthesizes the persona-aligned prompt $\mathcal{I}_i$ directly from transition $\mathcal{T}_i$. The persona p_i defines the professional background and framing, while the transition relation r_i together with the artifact sets $(\mathcal{A}_i^{\text{in}}, \mathcal{A}_i^{\text{out}})$ dictates the explicit goal, expected deliverables, and procedural constraints. Consequently, the generated instruction $\mathcal{I}_i$ faithfully reflects the substantive work required by the transition.

(2) Environment construction ($\mathcal{E}_i$). Guided by transition $\mathcal{T}_i$ and instruction $\mathcal{I}_i$, an environment agent sets up the execution workspace $\mathcal{E}_i$. To maximize realism, WorkBorne adopts a *real-artifact-first* policy: input artifacts $\mathcal{A}_i^{\text{in}}$ from corpus $\mathcal{C}_c$ are directly copied to populate the initial workspace, modified only when task-specific adaptations are necessary, and synthesized only as a last resort. In addition, output anchors $\mathcal{A}_i^{\text{out}}$ are carefully sanitized to retain essential structural context without leaking reference solutions.

(3) Evaluation rubric synthesis ($\mathcal{R}_i$). A rubric agent derives a structured set of verifiable evaluation rubrics $\mathcal{R}_i$ from transition relation r_i and output anchors $\mathcal{A}_i^{\text{out}}$. These rubrics establish concrete criteria assessing task completion, functional correctness, and grounding in the provided workspace, ensuring objective and automated trajectory evaluation.

3.3 Data Collection

We deploy Opus-4.6 (Anthropic, 2026) across the WorkBorne framework to synthesize 1,061 artifact corpora and 7,882 executable task environments. To collect execution trajectories without overfitting to a single interface or logging schema, we execute rollouts using Kimi-K3 (Team et al., 2026) as the teacher model across 14 mainstream agent harnesses (Appendix A). From approximately $24,000$ raw trajectories, we filter out infrastructure failures, normalize heterogeneous harness logs into a unified multi-turn format, and evaluate each rollout against task-specific rubrics using Opus-4.6. Retaining rollouts with scores of at least 8 out of 10 yields $15,485$ high-quality training trajectories.

As summarized in Table 1, the $1,061$ corpora contain $28,712$ selected source artifacts, supporting a median of 8 tasks per corpus. Each task environment averages 350 files (30.2 MB) and 10.4 rubrics, with 98.8% of file occurrences copied directly from the underlying source corpora. Furthermore, the retained trajectories average 37.5 assistant turns with a median of 32 turns, where 53.6% exceed 30 turns and 21.0% exceed 50 turns (Figure 2). These metrics confirm that transition-driven synthesis reliably induces sustained, long-horizon agent interactions at scale.

Table 1: WorkBorne dataset profile.

Corpora and tasks	
Tasks / corpus	7,882 / 1,061
Selected source artifacts	28,712
Median tasks per corpus	8
Environment	
Avg. files per task	350
Avg. environment size	30.2 MB
Avg. directory depth	5.5
Avg. rubrics per task	10.4
Corpus-sourced files	98.8%
Trajectory	
Harnesses / trajectories	14 / 15,485
Mean / median assistant turns	37.5 / 32
Long-horizon rate (>30)	53.6%

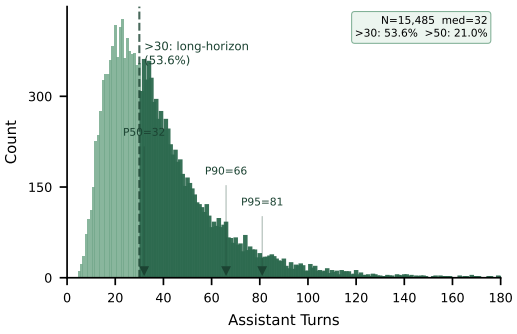


Figure 2: Assistant turns in retained trajectories. Darker bars mark those exceeding 30 turns (53.6%); the tail extends to 353.

4 Experiments

In this section, we evaluate the performance and generalizability of models trained on Workborne trajectories across three challenging benchmarks: Terminal-Bench Lite, Terminal-Bench 2.1 (Merrill et al., 2026), and ClawEval (Ye et al., 2026). Our evaluation focuses on three core aspects: performance against existing synthesis methods, generalization across heterogeneous agent frameworks, and the impact of model active capacity.

Table 2: **Main results on three agentic benchmarks:** Terminal-Bench Lite, Terminal-Bench 2.1, and ClawEval. We report mean raw reward for TB Lite, pass@1 for TB 2.1, and pass@3, passˆ3, and average score for ClawEval. Our Terminal-Bench scores are averaged over three runs. The best result among synthesis methods is shown in **bold**, and the second-best result is underlined. Green values in parentheses indicate absolute improvements over the corresponding Qwen3.5 base model. * marks scores we measure on the released models under the configurations in Appendix D; other scores are taken from the corresponding reports.

Model	Size	Terminal-Bench		ClawEval (General)		
		TB Lite	TB 2.1	Pass@3	Passˆ3	Score
Reference Frontier Models						
GPT-5.5 (OpenAI, 2026)	–	–	78.0	–	–	–
Opus-4.6 (Anthropic, 2026)	–	–	70.1	80.8	70.8	80.6
Gemini-3.1-Pro (Google DeepMind, 2026)	–	–	65.8	80.8	55.9	76.6
Open-source Models						
Kimi-K2.6 (Moonshot AI, 2026)	1T	–	59.9	78.9	61.5	77.6
GLM-5 (GLM-5-Team et al., 2026)	744B	70.6	44.9	73.9	57.1	73.8
DeepSeek-V4-flash-Preview (DeepSeek-AI et al., 2026)	284B	–	61.8	78.3	57.8	74.6
MiniMax-M2.7 (Chen et al., 2026a)	230B	–	55.4	72.0	49.7	71.8
Base & Synthesis methods						
Qwen3.5-35B-A3B (Qwen Team, 2026)	35B	58.8*	39.7*	77.0*	62.1*	74.7*
Qwen3.5-27B (Qwen Team, 2026)	27B	68.3*	43.1*	77.6*	63.4*	75.0*
Nemotron-32B (Pi et al., 2026)	32B	48.6*	25.1*	39.1*	4.4*	36.0*
TMax-27B (Ivison et al., 2026)	27B	68.6	44.9	81.4*	**67.7***	79.6*
RST-SFT-27B (Li et al., 2026a)	27B	71.9*	46.8*	79.5*	59.6*	75.7*
FACET-Terminal-27B (Shi et al., 2026)	27B	77.4*	47.6	80.1*	57.8*	77.1*
Terminal-Universe-27B (Wu et al., 2026a)	27B	-	58.1	-	-	-
Ours						
WorkBorne-35B-A3B	35B	66.5 (+7.7)	48.3 (+8.6)	80.7 (+3.7)	65.2 (+3.1)	78.3 (+3.6)
WorkBorne-27B	27B	**84.3** (+16.0)	**61.8** (+18.7)	**82.0** (+4.4)	66.5 (+3.1)	**80.2** (+5.2)

4.1 Experimental Setup

Training configuration. We use MS-SWIFT with Megatron to perform multi-turn SFT on Qwen3.5-27B and Qwen3.5-35B-A3B using 128 H20 GPUs. We train on all 15,485 WorkBorne trajectories for three epochs, with a peak learning rate of 4×10^{-5}, a global batch size of 32, and a sequence length of 256K tokens with packing. Before training, we check the training data for contamination against Terminal-Bench and ClawEval tasks using 13-gram matching (Appendix B). Full training settings appear in Appendix C.

Evaluation configuration. We use Terminal-Bench Lite and Terminal-Bench 2.1 to assess challenging, long-horizon task execution. To cover general agent work, we also evaluate the General subset of ClawEval. For both Terminal-Bench benchmarks, we use Terminus2 with temperature 1.0, and a context window of 128K tokens. We report mean raw reward for TB Lite and pass@1 for TB 2.1, both averaged over three runs. For ClawEval, we use the official harness with temperature 0.6 and a context window of 256K tokens. We report pass@3, strict joint accuracy (pass^3), and average score. Full evaluation settings appear in Appendix D.

4.2 Main Results

Table 2 presents the results across three benchmarks. We summarize the key findings as follows:

State-of-the-art performance from synthesized supervision. WorkBorne establishes top-tier performance among synthesis methods, delivering substantial gains across all benchmarks. WorkBorne-27B reaches 84.3% on Terminal-Bench Lite and 61.8% on Terminal-Bench 2.1, improving over its Qwen3.5-27B base by 16.0 and 18.7 percentage points. Using **only 15K synthesized**

trajectories, the 27B model competes directly with substantially larger open models on Terminal-Bench 2.1—matching DeepSeek-V4-flash-Preview (61.8%) and outperforming Kimi-K2.6 (59.9%).

Gains persist across heterogeneous agent workflows. The performance gains transfer robustly across fundamentally distinct agent architectures and evaluation harnesses. While Terminal-Bench evaluates long-horizon terminal execution via Terminus2, ClawEval measures broader agent capabilities under claw-agent. WorkBorne-27B achieves +4.4% pass@3 and +5.2% overall score on ClawEval, alongside a +3.1% increase in strict joint accuracy (pass^3). The improvement in pass^3 confirms that WorkBorne supervision fosters consistent task completion across repeated trials.

Greater active capacity unlocks larger gains from WorkBorne. The value of WorkBorne's long-horizon supervision scales directly with the learner's active capacity. While the MoE variant Qwen3.5-35B-A3B improves steadily across benchmarks (+8.6% on Terminal-Bench 2.1), the dense Qwen3.5-27B extracts dramatically larger gains from the identical 15K trajectory budget (+18.7% on TB 2.1 and +5.2% vs. +3.6% on ClawEval average). This gap on complex, long-horizon tasks indicates that dense parameters provide the active capacity required to fully absorb rich, multi-stage interaction trajectories.

5 Analysis and Ablation Studies

In this section, we analyze the core design choices driving WorkBorne's performance gains: **grounding tasks in real-world artifacts** and **artifact transition modeling**. We conduct systematic experiments to address three key aspects: **(i) Construction Pipeline Ablation (§5.1):** whether WorkBorne's transition modeling provides additional execution depth and downstream training value beyond baselines; **(ii) Transition Structure and Execution Depth (§5.2):** which properties of agentic tasks are associated with deeper execution, and how transition structures shape long-horizon behavior; and **(iii) Corpus and Transition Scaling (§5.3):** how to scale WorkBorne data efficiently by allocating budgets between corpus breadth and transition depth. All comparisons are conducted on Qwen3.5-27B under matched trajectory budgets.

5.1 Construction Pipeline Ablation

Table 3: Environment, trajectory, and performance comparison across construction pipelines. Each pipeline contributes 1,000 environments and 1,000 training trajectories. Counts are means over the sampled environments and retained trajectories; long-horizon rate is the percentage of retained trajectories with > 30 assistant turns.

	Environment			Trajectory			Performance	
Method	File count	File types	Rubrics	Assistant turns	Tool calls	Long-horizon rate (%)	TB Lite	TB 2.1
Instruction-first	10.8	5.5	9.1	25.9	35.8	27.4	68.3	47.2
Input-only	283.2	15.5	7.6	27.8	36.6	33.0	69.0	48.3
WorkBorne	**347.3**	**18.3**	**10.6**	**44.5**	**57.5**	**66.6**	**71.1**	**51.7**

To examine how transition modeling contributes to these gains, we compare three task construction pipelines starting from a shared pool of 125 skills: **Instruction-first** follows Terminal-World (Cheng et al., 2026) by generating instructions before environment construction; **Input-only** uses retrieved artifacts directly as inputs to synthesize instructions; and **WorkBorne** discovers artifact transitions to instantiate both jointly. For each pipeline, we collect 1,000 environments and 1,000 Kimi-K3 trajectories to fine-tune Qwen3.5-27B under matched setups. Table 3 details environment characteristics, trajectory statistics, and downstream performance. Additional statistics and human validation appear in Appendix E and Appendix F.

Transition modeling yields deeper execution together with stronger downstream training performance. Rich environments alone fail to demand sustained interaction: moving from Instruction-first to Input-only increases file counts from 10.8 to 283.2, yet rubrics drop (9.1 to 7.6) and average

turns barely change (25.9 to 27.8). In contrast, WorkBorne's artifact transitions jointly construct instructions and environments, raising rubrics to 10.6 and average turns to 44.5. The long-horizon rate reaches 66.6%, compared with 33.0% for Input-only and 27.4% for Instruction-first. Under the same 1,000-trajectory training budget, this deeper execution is accompanied by stronger downstream performance: WorkBorne-27B reaches 71.1% on TB Lite and 51.7% on TB 2.1, outperforming Input-only by **+2.1 and +3.4 points** (and Instruction-first by **+2.8 and +4.5 points**). These results show that WorkBorne yields substantially deeper, more long-horizon execution together with stronger downstream training performance.

5.2 Transition Structure and Execution Depth

Section 5.1 shows that transition modeling yields both deeper execution and stronger downstream performance. To uncover what sustains long-horizon interaction, we analyze how core task properties—artifact count, anchor forms, assembly style, and transition relation type—associate with execution depth, measured by median assistant turn count. Table 4 summarizes these associations using the correlation ratio η on ranked turn counts, with representative assemblies and relations illustrated in Figure 3.

Transition structure is substantially more informative than artifact quantity. Artifact counts have little association with execution depth ($\eta = 0.049$ for input count and 0.015 for total count), whereas input-output artifact assembly shows a much stronger association ($\eta = 0.314$). Within the transition structure, output-anchor form is also substantially more informative than input form ($\eta = 0.306$ vs. 0.183), indicating that execution depth depends not only on what the agent starts from, but also on the structure of the intended outcome. Figure 3 further shows that the connecting relation matters even when the artifact assembly is fixed: within repository-to-repository transitions, median execution rises from 34 turns for structural imitation to 47 for porting. Together, these results show that long-horizon execution is associated with **how artifacts are organized into an input-output transition**, rather than simply how many artifacts a task contains. Artifact distributions appear in Appendix G. Appendix H provides statistical definitions and aggregation details, together with a 73-turn CBraid porting trajectory illustrating this pattern.

Table 4: Association with execution depth. Larger η indicates stronger association.

Factor	η
Input artifact count	0.049
Total artifact count	0.015
Input artifact form	0.183
Output anchor form	**0.306**
Artifact assembly	**0.314**
Transition relation type	0.140

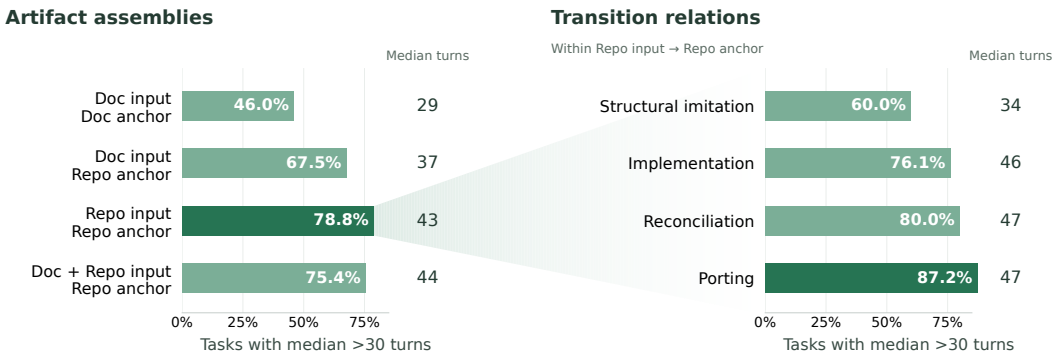


Figure 3: **Transition structure corresponds to different execution horizons.** Left: selected working input and output anchor assemblies. Right: an expansion of the highlighted repository-to-repository assembly, comparing four transition relations while holding the assembly fixed. Bars show the percentage of tasks whose median execution exceeds 30 assistant turns; the adjacent column reports the median across tasks, rounded to the nearest turn.

5.3 Corpus and Transition Scaling

The training gains from transition modeling motivate expanding the supervision it produces. WorkBorne offers two ways to do so: **retrieve more corpora** or **explore additional transitions within existing corpora**. To determine how to allocate a training data budget, we vary the number of corpora C and the number of transitions discovered per corpus T. Each transition yields one task for trajectory collection. We evaluate both scaling paths across matched budgets (1K, 3K, 5K, and 8K trajectories), converging at a shared 8K endpoint ($1{,}000 \times 8$). This yields seven training configurations for Qwen3.5-27B (detailed in Figure 4).

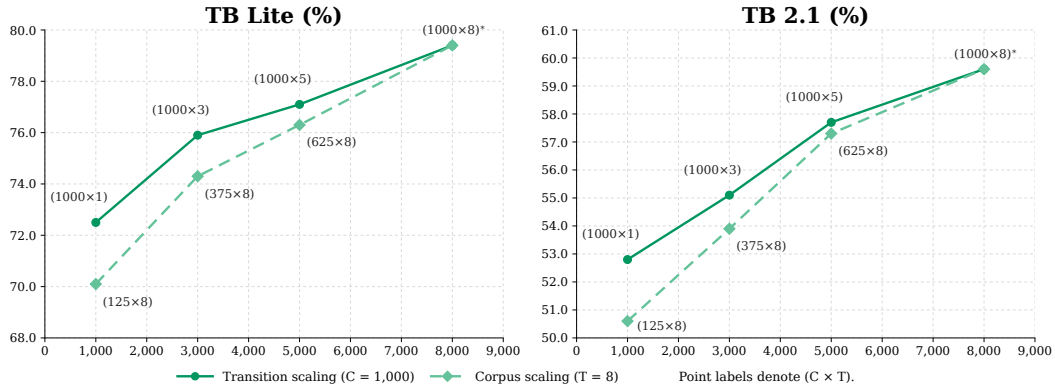


Figure 4: **Both scaling paths improve performance.** Transition scaling holds corpus coverage fixed, while corpus scaling holds transitions per corpus fixed. The horizontal axis shows the number of training trajectories. The vertical axis shows mean raw reward for TB Lite and pass@1 for TB 2.1, both expressed as percentages and averaged over three runs. Point labels denote $C \times T$, where C is the number of corpora and T is the number of transitions per corpus; the two paths share the $1{,}000 \times 8$ endpoint. *After failure filtering, this configuration contains slightly fewer than 8,000 tasks ($< 2\%$ shortfall); we fill the remaining trajectory budget with additional rollouts of retained tasks under different harnesses.

Corpus breadth and transition depth provide complementary gains. Both scaling paths improve performance across benchmarks. Increasing transitions per corpus from 1 to 8 raises TB Lite from 72.5% to 79.4% and TB 2.1 from 52.8% to 59.6%, showing that existing corpora can support additional useful supervision without new retrieval. However, at every matched budget before the shared endpoint, spreading trajectories across more corpora performs better than sampling more transitions from fewer corpora. This advantage suggests that diversity in the underlying artifact contexts is itself an important source of agentic supervision: new corpora expose the learner to different materials and work settings, while additional transitions mainly extract further value from an existing context. Together, these results suggest a clear scaling strategy: **prioritize corpus breadth under limited budgets, then scale transition depth to fully extract value from those corpora.**

6 Conclusion

In this paper, we introduced WorkBorne, a novel task synthesis framework that resolves the instruction-environment asymmetry in terminal agent data generation by modeling agentic tasks through artifact transitions. By grounding task discovery in a 3W primitive that specifies who performs the work, what inputs are provided, and what output anchors are intended, WorkBorne seamlessly bridges authentic workspace richness with substantive instruction design. Comprehensive evaluations show that WorkBorne substantially improves downstream agent performance across multiple benchmarks, outperforming existing synthesis baselines and inducing long-horizon interactions that generalize well across diverse execution harnesses. Furthermore, our empirical analysis reveals that task execution depth is primarily driven by transition structure rather than artifact quantity, providing concrete insights for future research on scalable long-horizon agent training.

AI USE STATEMENT

Generative AI tools were used to assist with language polishing and improving the readability of the manuscript. Generative AI models were also used as part of the WorkBorne pipeline to generate synthetic agentic tasks and trajectories, as described in the main paper. All AI-assisted content and generated data used in this work were reviewed and validated by the authors, who take full responsibility for the final content of the paper.

REFERENCES

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

Anthropic. System card: Claude Opus 4.6. Technical report, Anthropic, February 2026. URL https://www.anthropic.com/claude-opus-4-6-system-card.

Aili Chen, Aonian Li, Baichuan Zhou, Bangwei Gong, Binyang Jiang, Boji Dan, Changhao Zhang, Changqing Yu, Chao Wang, Cheng Ma, Cheng Zhong, Cheng Zhu, Chengjun Xiao, Chengyi Yang, Chengyu Du, Chenyang Zhang, Chi Zhang, Chuangyi Huang, Chunhao Zhang, Chunhui Du, Chunyu Zhao, Congchao Guo, Da Chen, Deming Ding, Dianjun Sun, Dong Li, Dongyu Zhang, Enhui Yang, Fei Yu, Guang Zheng, Guodong Zheng, Guohong Li, Haichao Zhu, Haigang Zhou, Haimo Zhang, Han Ding, Hao Zhang, Haohai Sun, Haolin Lyu, Haonan Lu, Haoyu Wang, Huajie Shi, Huiyang Li, Jiacheng Chen, Jian Zhang, Jiaqi Zhuang, Jiaren Cai, Jiaxin Pan, Jiayao Li, Jiayuan Song, Jichuan Zhang, Jie Wang, Jihao Gu, Jin Zhu, Jingwei Dong, Jingyang Li, Jingyu Zhang, Jingze Zhuang, Jinhao Tian, Jinli Liu, Jinyi Hu, Jun Tao, Jun Zhang, Junbin Ruan, Junhao Xu, Junjie Yan, Junteng Liu, Junxian He, Kang Xu, Ke Ji, Ke Yang, Kecheng Xiao, Keyu Duan, Keyu Li, Le Han, Letian Ruan, Li Yuan, Lianfei Yu, Liheng Feng, Lijie Mo, Lin Li, Linge Du, Lingye Bao, Lingyu Yang, Lingyuan Zhou, Loki, Lu Chen, Lunbin Zeng, Ming Li, Ming Zhong, Mingliang Tao, Mingyuan Chi, Mujie Lin, Nan Hu, Ningxin Chen, Peiyin Zhu, Peng Gao, Pengcheng Gao, Pengfei Li, Penglin Li, Pengyu Zhao, Qibin Ren, Qibing Ren, Qidi Xu, Qihan Ren, Qile Li, Qin Wang, Quanliang Chen, Qunhong Zeng, Rong Tian, Rongxin Guo, Rui Dong, Ruitao Leng, Ruize Zhang, Shanqi Liu, Shaoxiang Chen, Shaoyu Chen, Sheng Jia, Shun Yao, Shuoran Zhao, Shuqi Yu, Sichen Li, Sicheng Pan, Songquan Zhu, Tengfei Li, Tian Xie, Tiancheng Qin, Tianle Li, Tianrun Liang, Wei Liu, Weiqi Xu, Weitao Li, Weixiang Chen, Weiyu Cheng, Weiyu Zhang, Wenhu Chen, Wenqian Zhao, Xiancai Chen, Xiangjun Song, Xiangyuan Wang, Xianzhen Luo, Xiao Luo, Xiao Su, Xiaobo Li, Xiaodong Han, Xiaojie Wu, Xihao Song, Xingyi Han, Xinyu Guan, Xuan Lu, Xun Zou, Xunhao Lai, Xutong Li, Xuyang Shen, Yan Gong, Yan Ma, Yang Jiao, Yang Wang, Yang Xu, Yangsen Wang, Ye Tang, Yicheng Chen, Yihang Wang, Yinran Qiu, Yiqi Shi, Yiting Guo, Yiwen Huang, Yixuan Wang, Yongyi Hu, Yu Gao, Yu Zhang, Yuan Li, Yuanxiang Ying, Yuanzhen Zhang, Yubo Wang, Yuchen Song, Yufeng Yang, Yuhang Meng, Yuhang Miao, Yuhao Li, Yujie Liu, Yulin Hu, Yunan Huang, Yunji Li, Yunyi Huang, Yusen Zhang, Yusu Hong, Yutao Xie, Yutong Zhang, Yuwen Liao, Yuxuan Shi, Yuze Wenren, Zebin Li, Zehan Li, Zejian Luo, Zeyu Jin, Zeyuan Sun, Zhanpeng Zhou, Zhaochen Su, Zhendong Li, Zhengmao Zhu, Zhengyuan Peng, Zhenhua Fan, Zhi Zhang, Zhichao Xu, Zhiheng Lv, Zhikang Xu, Zhitao He, Zhiwei He, Zhongyuan Li, Zibo Gao, Zijia Wu, Zijian Song, Zijian Zhou, Zijun Sun, Zishan Huang, Ziying Chen, and Ziyue Ge. The minimax-m2 series: Mini activations unleashing max real-world intelligence, 2026a. URL https://arxiv.org/abs/2605.26494.

Shoufa Chen, Luyuan Wang, Xuan Yang, Zhiheng Liu, Yuren Cong, Yuanfeng Ji, Feiyan Zhou, Xiaohui Zhang, Fanny Yang, and Belinda Zeng. Tua-bench: A benchmark for general-purpose terminal-use agents. In *arXiv*, 2026b. URL https://arxiv.org/abs/2606.28480.

Zihao Cheng, Hongru Wang, Zeming Liu, Xinyi Wang, Xiangrong Zhu, Yuhang Guo, Wei Lin, Jeff Z. Pan, and Yunhong Wang. Terminal-World: Scaling terminal-agent environments via agent skills. In *arXiv*, 2026. URL https://arxiv.org/abs/2605.20876.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin,

Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Rishi Desai, Jesse Hu, Joan Cabezas, Neel Harsola, Pratyush Shukla, Roey Ben Chaim, Adnan El Assadi, Omkaar Mukund Kamath, Fenil Faldu, Prannay Hebbar, Jiankai Sun, Yiyuan Li, Pramod Srinivasan, Ishan Gupta, Christopher Settles, Daniel Wang, Derek Chen, Pranav Raja, Albert Liu, Marek Šuppa, Nevasini Sasikumar, Luyang Kong, Erik Quintanilla, Xiangyi Li, Ivan Bercovich, and Steven Dillmann. SWE-Marathon: Can agents autonomously complete ultra-long-horizon software work? In *arXiv*, 2026. URL https://arxiv.org/abs/2606.07682.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, Jiajie Jin, Yutao Zhu, Hanbin Wang, Fangyu Lei, Qinyu Luo, Mingyang Chen, Zehui Chen, Jiazhan Feng, Ji-Rong Wen, and Zhicheng Dou. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence, 2026. URL https://arxiv.org/abs/2604.18292.

Zhiyuan Fan, Tinghao Yu, Yuanjun Cai, Jiangtao Guan, Yun Yang, Dingxin Hu, Jiang Zhou, Xing Wu, Zhuo Han, Feng Zhang, and Lilin Wang. Toward scalable terminal task synthesis via skill graphs. In *arXiv*, 2026. URL https://arxiv.org/abs/2604.25727.

Kanishk Gandhi, Shivam Garg, Noah D. Goodman, and Dimitris Papailiopoulos. Endless terminals: Scaling rl environments for terminal agents. In *arXiv*, 2026. URL https://arxiv.org/abs/2601.16443.

Tao Ge, Xin Chan, Xiaoyang Wang, Dian Yu, Haitao Mi, and Dong Yu. Scaling synthetic data creation with 1,000,000,000 personas, 2025. URL https://arxiv.org/abs/2406.20094.

GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Google DeepMind. Gemini 3.1 Pro model card. Technical report, Google DeepMind, February 2026. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/.

Qingnuan Han, Boli Fang, Mingzhi Hou, and Claire Liu. Rideway: Benchmarking efficient task completion for tool-using language agents, 2026. URL https://arxiv.org/abs/2609.17985.

Zhanbo Hua, Yifan Yao, Weihao Xie, Yongchi Zhao, Minghao Liu, Ruizhi Qiu, Zhewei Huang, Zun Wang, Yiyan Ji, Yunhai Ye, Letian Zhu, Xinping Lei, Han Li, Zhiyuan Ma, Zili Wang, Zhaoxiang Zhang, and Jiaheng Liu. CLI-Universe: Towards verifiable task synthesis engine for terminal agents. In *arXiv*, 2026. URL https://arxiv.org/abs/2606.22883.

Hamish Ivison, Junjie Oscar Yin, Rulin Shao, Teng Xiao, Nathan Lambert, and Hannaneh Hajishirzi. Tmax: A simple recipe for terminal agents, 2026. URL https://arxiv.org/abs/2606.23321.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Zhongzhi Li, Yucheng Shi, Zongxia Li, Ruhan Wang, Anhao Li, Zixun Huang, Junyao Yang, Lei Ke, Ninghao Liu, Haitao Mi, and Leowei Liang. Recursive synthesis for long-horizon terminal tasks. In *arXiv*, 2026a. URL https://arxiv.org/abs/2608.05466.

Zongxia Li, Zhongzhi Li, Yucheng Shi, Ruhan Wang, Junyao Yang, Zhichao Liu, Xiyang Wu, Anhao Li, Yue Yu, Ninghao Liu, Lichao Sun, Haotao Mi, and Leowei Liang. Long-horizon-terminal-bench: Testing the limits of agents on long-horizon terminal tasks with dense reward-based grading, 2026b. URL https://arxiv.org/abs/2607.08964.

Yusong Lin, Haiyang Wang, Shuzhe Wu, Lue Fan, Feiyang Pan, Sanyuan Zhao, and Dandan Tu. CLI-Gym: Scalable cli task generation via agentic environment inversion. In *arXiv*, 2026. URL https://arxiv.org/abs/2602.10999.

Junteng Liu, Yunji Li, Chi Zhang, Jingyang Li, Aili Chen, Ke Ji, Weiyu Cheng, Zijia Wu, Chengyu Du, Qidi Xu, Jiayuan Song, Zhengmao Zhu, Wenhu Chen, Pengyu Zhao, and Junxian He. Webexplorer: Explore and evolve for training long-horizon web agents, 2025. URL https://arxiv.org/abs/2509.06501.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh R. N., Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/61cce86d180b1184949e58939c4f983d-Abstract-Datasets_and_Benchmarks_Track.html.

Sina Mavali, David Pape, Jonathan Evertz, Samira Abedini, Devansh Srivastav, Thorsten Eisenhofer, Sahar Abdelnabi, and Lea Schönherr. No more, no less: Task alignment in terminal agents. In *arXiv*, 2026. URL https://arxiv.org/abs/2605.12233.

Fanzhe Meng, Guoxin Chen, Jiale Zhao, Shuang Sun, Zhiyu Lin, Wayne Xin Zhao, Ruihua Song, Ji-Rong Wen, and Kai Jia. Calibforge: Adversarial solver calibration for scaling learnable terminal tasks. In *arXiv*, 2026. URL https://arxiv.org/abs/2608.06352.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026. URL https://arxiv.org/abs/2601.11868.

Moonshot AI. Kimi K2.6: Advancing open-source coding, 2026. URL https://www.kimi.com/blog/kimi-k2-6.

OpenAI. GPT-5.5 system card, April 2026. URL https://openai.com/index/gpt-5-5-system-card/.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-gym. In *International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=Cq1BNvHx74.

Zhihong Pan, Jiyuan He, Kai Zhang, Yupeng Han, Ze Liu, Yuze Zhao, Yongcong Ye, and Zhaohua Yang. Meta-task: Turning terminal task synthesis into a terminal task for scalable agent training. In *arXiv*, 2026. URL https://arxiv.org/abs/2607.27929.

Renjie Pi, Grace Lam, Mohammad Shoeybi, Pooya Jannaty, Bryan Catanzaro, and Wei Ping. On data engineering for scaling llm terminal capabilities. In *arXiv*, 2026. URL https://arxiv.org/abs/2602.21193.

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

Jincheng Ren, Siwei Wu, Yizhi Li, Kang Zhu, Shu Xu, Boyu Feng, Ruibin Yuan, Wei Zhang, Riza Batista-Navarro, Jian Yang, and Chenghua Lin. A self-evolving framework for efficient terminal agents via observational context compression. In *arXiv*, 2026. URL https://arxiv.org/abs/2604.19572.

Kou Shi, Zun Wang, Qisheng Su, Shiting Huang, Ziao Zhang, Zhen Fang, Qingnan Ren, Jin Liu, Yu Zeng, Yiming Zhao, Lin Chen, Zehui Chen, and Feng Zhao. FACET: Preserving source intent and executable state in terminal task synthesis. In *arXiv*, 2026. URL https://arxiv.org/abs/2608.18580.

Zirui Tang, Xuanhe Zhou, Yumou Liu, Linchun Li, Yukai Wu, Weizheng Wang, Hongzhang Huang, Wei Zhou, Jun Zhou, Jiachen Song, Shaoli Yu, Jinqi Wang, Zihang Zhou, Hongyi Zhou, Yuting Lv, Jinyang Li, Jiashuo Liu, Ruoyu Chen, Chunwei Liu, GuoLiang Li, Jihua Kang, and Fan Wu. Workspace-bench 1.0: Benchmarking ai agents on workspace tasks with large-scale file dependencies, 2026. URL https://arxiv.org/abs/2605.03596.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, M. C., Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y. Charles, H. S. Che, Guanduo Chen, Guangyu Chen, Guanzheng Chen, Huarong Chen, Jia Chen, Jianlong Chen, Jun Chen, Kexin Chen, Peng Chen, Ruijue Chen, Wentao Chen, Xin Chen, Yang Chen, Yanru Chen, Yifei Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Dazhi Cheng, Yean Cheng, Jialei Cui, Jingbing Cui, Anqi Dai, Jiaqi Deng, Hao Ding, Rui Ding, Shaofeng Ding, Mengfan Dong, Mengnan Dong, Yuhao Dong, Yuxin Dong, Angang Du, Chenzhuang Du, Dikang Du, Jusen Du, Yulun Du, Yu Fan, Jing Feng, Qiulin Feng, Yichen Feng, Kelin Fu, Qiang Fu, Fuxuan Gao, Hongcheng Gao, Jingyue Gao, Tong Gao, Weijia Gao, Shangyi Geng, Jie Gong, Linhu Gong, Shengao Gong, Xiaochen Gong, Qizheng Gu, Yicheng Gu, Shuhao Guan, Haiqing Guo, Shiqi Guo, Xiang Guo, Zhengyan Guo, Beixi Hao, Wenxin Hao, Xiaoru Hao, Dailan He, Haotian He, Lehan He, Qi He, Weiran He, Xinran He, Xinyi He, Yibo He, Yunjia He, Chao Hong, Tiange Hong, Hao Hu, Jiaxi Hu, Ruikun Hu, Weiming Hu, Yangyang Hu, Zhenxing Hu, Liang Hua, Jinbin Huang, Ke Huang, Ruiyuan Huang, Siying Huang, Weixiao Huang, Yan Huang, Zhengjie Huang, Zhiqi Huang, Yulong Hui, Chaobo Jia, Yutong Jiang, Zhejun Jiang, Zuoyou Jiang, Wenyi Jin, Xinyi Jin, Yu Jing, Huanjun Kong, Guokun Lai, Aidi Li, Cheng Li, Chengyuan Li, Cong Li, Fang Li, Guanyu Li, Haoyang Li, Jia Li, Junxiong Li, Lei Li, Letian Li, Lincan Li, Weihong Li, Wentao Li, Xintong Li, Yang Li, Yishen Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zhaoxi Li, Zheming Li, Zhengxiao Li, Zhiyuan Li, Jiawei Lin, Xiaohan Lin, Yibo Lin, Zichao Lin, Ziyan Lin, Bill Liu, Boxiao Liu, Chuan Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yipeng Liu, Zhengying Liu, Zhiheng Liu, Enzhe Lu, Haoyu Lu, Linqiang Lu, Tingzhan Lu, Zhiyuan Lu, Aotian Luo, G. Luo, Junyu Luo, Yifan Luo, B. Lyu, Wenzhou Lyu, Shaoguang Mao, Yuan Mei, Xin Men, Minqing Ni, Yixuan Niu, Siyuan Pan, Shujun Peng, Zhangyang Qi, Ruoyu Qin, ZeChao Qin, Zeyu Qin, Haiquan Qiu, Jianxin Qiu, Jiezhong Qiu, Bowen Qu, Yuhao Qu, Zeyu Shang, Youbo Shao, Han Shen, Jincheng Shi, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Wingchun Siu, Pengwei Song, Xiaoxi Song, Jianlin Su, Yunfeng Su, Zhaochen Su, Lin Sui, Jingsong Sun, Junyao Sun, Shaoning Sun, Shuzhe Sun, Tongyu Sun, Yujun Sun, Yunpeng Tai, Chuning Tang, Heyi Tang, Sirui Tang, Zecheng Tang, Chaoran Tian, Rongpeng Tian, Yu Tian, Wei Tu, Chensi Wang, Chuang Wang, Chunjie Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Hao Wang, Hao Wang, Huaqing Wang, Hui Wang, Jiayi Wang, Jinglong Wang, Jinhong Wang, Jiuzheng Wang, Linian Wang, Shaobo Wang, Shenzhi Wang, Shuyi Wang, Si Wang, Siyuan Wang, Tianfu Wang, Wenjue Wang, Xingran Wang, Xinmei Wang, Xinyuan Wang, Xusheng Wang, Yalin Wang, Yangkun Wang, Yao Wang, Yaoyu Wang, Yejie Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhenhao Wang, Zhongsheng Wang, Zifan Wang, Chu Wei, Ming Wei, Shouxin Wei, Zichen Wen, Fan Wu, Haoning Wu, Rucong Wu, Wenhao Wu, Xiaoxue Wu, Yingcong Wu, Yongqi Wu, Yuxin Wu, Zijian Wu, Xinglang Xian, Chenxuan Xiang, Yuye Xiang, Bocheng Xiao, Chenjun Xiao, Xin Xiao, Jin Xie, Xiaotong Xie, Yifeng Xie, Zhe Xie, Bowei Xing, Yiming Xiong, Baosheng Xu, Boyu Xu, Jiale Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Qingtao Xu, Shuyao Xu, Suting Xu, Tiantian Xu, Tianxiang Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ye Xu, Yueni Xu, Ziyao Xu, Haonan Xue, Junjie Yan, Yaoyao Yan, Fan Yang, Guangyao Yang, Hao Yang, Junwei Yang, Ruoyu Yang, Wenjie Yang, Xiaofei Yang, Xinyu Yang, Yi Yang, Yiling Yang, Ying Yang, Yuchen Yang, Zhen Yang, Zhilin Yang, Zian Yang, Zuhao Yang, Haotian Yao, Dan Ye, Haoran Ye, Wenjie Ye, Zhanbo Ye, Bohong Yin, Haoxiang Yin, Xietong Yin,

Chengzhen Yu, Haozhen Yu, Longhui Yu, Shengnan Yu, Shuying Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Tongtian Yue, Wei Yue, Yang Yue, Dunyuan Zha, Haobing Zhan, B. H. Zhang, Dehao Zhang, Fei Zhang, Hao Zhang, Haoyuan Zhang, Huanyu Zhang, Jiapei Zhang, Jiaxuan Zhang, Jin Zhang, Kaiyi Zhang, Miaozhen Zhang, Puqi Zhang, Qinglei Zhang, Rong Zhang, Rui Zhang, Shaoshuai Zhang, Shiyi Zhang, Xiaobin Zhang, Xiaoyun Zhang, Y. Zhang, Yangkun Zhang, Ye Zhang, Yichi Zhang, Yikun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Zijing Zhang, Bin Zhao, Chenguang Zhao, Feifan Zhao, Jinglun Zhao, Jinxiang Zhao, Shuai Zhao, Wenshuo Zhao, Xiangyu Zhao, Xuanle Zhao, Yikai Zhao, Zijia Zhao, Haozhi Zheng, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Haofeng Zhong, Lei Zhong, Longguang Zhong, M. Zhou, Qiankang Zhou, Runjie Zhou, Ruozhang Zhou, Xinyu Zhou, Yiqiao Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yangjunfeng Zhu, Yuxuan Zhu, Zhen Zhu, Chen Zhuang, Weiyu Zhuang, and Xinxing Zu. Kimi k3: Open frontier intelligence, 2026. URL https://arxiv.org/abs/2607.24653.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *International Conference on Machine Learning*, 2024.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Daniel Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=OJd3ayDDoF.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent world model: Infinity synthetic environments for agentic reinforcement learning. In *International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=OvLnVkV14P.

Jie Wu, Zhenru Zhang, Beichen Zhang, Xuwu Wang, Yuhui Su, Mouxiang Chen, Peng Wang, Zhihai Wang, Que Shen, Hao Zhou, An Yang, Fei Huang, Yujiu Yang, and Dayiheng Liu. Terminal-universe: Turning agent trajectories into scalable terminal environments, 2026a. URL https://arxiv.org/abs/2609.04148.

Siwei Wu, Yizhi Li, Yuyang Song, Wei Zhang, Yang Wang, Riza Batista-Navarro, Xian Yang, Mingjie Tang, Bryan Dai, Jian Yang, and Chenghua Lin. Large-scale terminal agentic trajectory generation from dockerized environments. *CoRR*, abs/2602.01244, 2026b. doi: 10.48550/ARXIV.2602.01244. URL https://doi.org/10.48550/arXiv.2602.01244.

John Yang, Akshara Prabhakar, Karthik R Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. In *Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=fvKaLF1ns8.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=mXpq6ut8J3.

John Yang, Kilian Lieret, Carlos E Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. In *Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2026a. URL https://openreview.net/forum?id=63iVrXc8cC.

Junyao Yang, Yucheng Shi, Zhongzhi Li, Ruhan Wang, Zongxia Li, Haitao Mi, and Leowei Liang. T1: Terminal agent reinforcement learning for long-horizon tasks. In *arXiv*, 2026b. URL https://arxiv.org/abs/2609.11042.

Sidi Yang, Chaofan Tao, Jierun Chen, Tiezheng Yu, Ruoyu Wang, Yuxin Jiang, Yiming Du, Wendong Xu, Jing Xiong, Taiqiang Wu, Lifeng Shang, Xiaohui Li, Ngai Wong, and Haoli Bai. What makes interaction trajectories effective for training terminal agents? In *arXiv*, 2026c. URL https://arxiv.org/abs/2606.03461.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, Qi Liu, Zhifang Sui, and Tong Yang. Claw-eval: Towards trustworthy evaluation of autonomous agents, 2026. URL https://arxiv.org/abs/2604.06132.

Jiarong Zhao, Zhikai Lei, Zhiheng Xi, Rui Zheng, Hang Yan, Jie Zhou, Qin Chen, and Liang He. NexForge: Scaling agent capabilities through requirement-driven task synthesis for llms. In *arXiv*, 2026. URL https://arxiv.org/abs/2607.14186.

Kaijie Zhu, Yuzhou Nie, Yijiang Li, Yiming Huang, Jialian Wu, Jiang Liu, Ximeng Sun, Zhenfei Yin, Lun Wang, Zicheng Liu, Emad Barsoum, William Yang Wang, and Wenbo Guo. Termigen: High-fidelity environment and robust trajectory synthesis for terminal agents. In *arXiv*, 2026. URL https://arxiv.org/abs/2602.07274.

A Harness Statistics

To characterize the interaction interfaces represented in training, Table 5 reports trajectory counts and shares for all 14 harnesses. The statistics use the final 15,485 training records after rubric filtering and cleaning, with each row counted once.

Table 5: Trajectory statistics by harness in the final training set. Shares use all 15,485 training trajectories.

Harness	**Trajectories**	**Share (%)**
Mini-SWE	1,614	10.4
Stirrup	1,437	9.3
OpenHands	1,421	9.2
Terminus2	1,208	7.8
Hermes	1,190	7.7
OpenClaw	1,131	7.3
Pi	1,078	7.0
CodeBuddy	1,050	6.8
KiloCode	1,003	6.5
Claude Code	974	6.3
Codex	963	6.2
Cline	944	6.1
OpenCode	917	5.9
Copilot	555	3.6
All harnesses	15,485	100.0

Training data are distributed across all harnesses. Each harness contributes at least 555 trajectories, and the largest share is 10.4%, so no single interface dominates the retained set.

B Data Contamination Checks

To check for benchmark contamination, we compare WorkBorne training data against the official instructions of TB Lite, TB 2.1, and ClawEval general. We apply normalized 13-gram matching to training instructions, task metadata, and all 15,485 trajectories, including reasoning, tool calls, and tool outputs. We also compute 5-gram Jaccard similarity between training and evaluation instructions and inspect 33 top-ranked pairs for shared task objectives, inputs, and output requirements. In addition, we screen artifact selections and environment construction records across all 7,882 tasks, together with observed tool outputs, for benchmark identifiers and provenance markers. **We find no task overlap or benchmark leakage in the inspected data.** The maximum instruction-level Jaccard similarity is below 0.013 across all three benchmarks (Table 6).

Artifact-level contamination and licensing. Beyond task and trajectory-level checks, we also audit all artifacts collected during retrieval before they enter the corpus construction pipeline. We

screen artifact contents and associated provenance metadata for benchmark-specific identifiers, task descriptions, and other signals of overlap with our evaluation benchmarks, and exclude any artifact with evidence of benchmark leakage. We find no detected overlap between the retrieved artifact pool and Terminal-Bench Lite, Terminal-Bench 2.1, or ClawEval General. In addition, we collect and normalize available license information for retrieved artifacts during corpus curation, and apply license-aware filtering and handling to ensure that artifact usage is consistent with their associated licensing terms.

Table 6: **Contamination checks against evaluation instructions.** Task overlaps are assessed through 13-gram matching and review. Jaccard similarity uses normalized 5-gram sets of task instructions. Text is normalized with Unicode NFKC and case folding, using alphanumeric word units and individual Han characters. Instructions shorter than 13 units are checked by full normalized instruction containment.

		Task overlaps			
Benchmark	Instructions	Training instructions	Task metadata	Trajectories	Max. Jaccard
TB Lite	100	0	0	0	0.0067
TB 2.1	89	0	0	0	0.0074
ClawEval general	161	0	0	0	0.0122

C Training Configurations

To make the effect of the training data comparable across models, we use the same WorkBorne training data and optimization settings for Qwen3.5-27B and Qwen3.5-35B-A3B. We perform full parameter SFT of the language model with MS-SWIFT 4.5.2 and Megatron/MCore, while keeping the vision encoder and aligner frozen. Table 7 lists the shared settings. The MoE model additionally uses expert parallelism and an auxiliary load balancing loss, as specified in Table 8.

To preserve complete interactions, trajectories longer than 256K tokens are discarded without truncation. We retain reasoning content, supervise assistant responses, and keep attention separate across packed trajectories.

D Evaluation Configurations

Table 9 summarizes the settings for our Terminal-Bench Lite, Terminal-Bench 2.1, and ClawEval (General) evaluations. Both evaluation pipelines use the language-model-only option.

Terminal-Bench. We use the Terminus2 agent for both Terminal-Bench benchmarks, with interleaved thinking enabled and top-$k = 20$. Context compression is triggered when 16,384 tokens remain in the context window. We set a four-hour timeout per task and use each task's official resource configuration. We report mean raw reward for TB Lite and pass@1 for TB 2.1, both averaged over three runs. For TB Lite, partial rewards contribute directly to the mean without binarization.

ClawEval. We evaluate the General subset with the endpoint's default thinking setting. Following the official ClawEval configuration, we use Gemini 3 Flash Preview via OpenRouter as the judge.

E Construction Comparison Statistics

To complement Table 3, Table 10 reports additional workspace structure, task requirements, and execution statistics. Environment metrics summarize 1,000 environments per pipeline; execution metrics summarize the corresponding retained trajectory sets. The environment and trajectory sets can differ in task coverage.

Table 7: Shared optimization, parallelism, and execution settings for both base models.

Setting	Value
Hardware	128 H20 GPUs
Tensor / pipeline / context / data parallelism	8 / 1 / 1 / 16
Sequence parallelism	Enabled
Micro / global batch size	1 / 32
Gradient accumulation steps	2
Training epochs	3
Maximum sequence length (tokens)	262,144
Packing method	binpack
Random seed / data seed	42 / 42
Peak / minimum learning rate	4×10^{-5} / 4×10^{-6}
Learning rate schedule	Cosine
Warmup fraction	10%
Optimizer	Adam
Adam β_1 / β_2	0.9 / 0.98
Weight decay	0.04
Weight decay for query/key LayerNorm	Not applied
Gradient clipping	1.0
Precision	BF16
Attention implementation	FlashAttention
Cross-entropy fusion	Transformer Engine
Gradient accumulation fusion	Enabled
Distributed optimizer	Enabled
Activation recomputation	Full, uniform, 1 layer
Multi-token prediction (MTP)	Disabled; no MTP loss

Table 8: Expert parallelism and MoE settings for Qwen3.5-35B-A3B.

Setting	Value
Expert parallelism	8
Expert tensor parallelism	1
Router auxiliary loss coefficient	10^{-6}
Token dispatcher	alltoall
Grouped GEMM	Enabled
Permutation fusion	Enabled
Shared expert overlap	Enabled
Expert capacity factor	Unset (dropless)

Table 9: Evaluation settings for Terminal-Bench and ClawEval.

Setting	Terminal-Bench	ClawEval (General)
Concurrency	25	40
Context window (tokens)	131,072	262,144
Thinking budget (tokens)	32,768	32,768
Temperature	1.0	0.6
Top-p	0.95	0.95

Environment measurements. File types count distinct filename suffixes within each environment. Maximum path depth is measured within each environment and then averaged across environments. File categories count the distinct recorded categories with at least one file. Environment size uses logical bytes, with 1 MB equal to 1,000,000 bytes.

Tool calls. We count each structured tool invocation once. For Terminus2, we count individual command entries in JSON responses or keystroke entries in XML responses, including polling actions. We do not split a shell invocation into its constituent commands. The mean uses trajectories whose action records can be fully parsed: 993 for Instruction-first, 997 for Input-only, and 996 for

Table 10: Full construction comparison. Values are arithmetic means except the long-horizon rate, which gives the percentage of retained trajectories exceeding 30 assistant turns.

Metric	Instruction-first	Input-only	WorkBorne
Task environments			
File count	10.8	283.2	347.3
File types	5.5	15.5	18.3
Directories	4.8	66.9	85.3
Maximum path depth	1.4	4.8	5.6
File categories	3.8	4.7	5.2
Environment size (MB)	3.5	28.6	33.2
Rubrics	9.1	7.6	10.6
Retained trajectories			
Assistant turns	25.9	27.8	44.5
Tool calls	35.8	36.6	57.5
Long-horizon rate (%)	27.4	33.0	66.6

WorkBorne. The remaining records contain malformed action formats and are excluded only from this metric. These are native invocation counts; the action interfaces differ across harnesses.

F Human Validation of Task and Rubric Quality

Protocol. To complement the automatic statistics in Section 5.1, we conduct a matched human evaluation over the three construction pipelines. We randomly sample 50 skills from the shared ablation pool and select one task generated by Instruction-first, Input-only, and WorkBorne for each skill, yielding 150 tasks in 50 matched triplets. Two expert reviewers independently inspect each task instruction and initial workspace, without access to the synthesis method, rollout trajectory, completion outcome, or automatic rubric score. Each task is rated on a 1–5 scale for *Task Complexity*, *Workspace Realism & Coherence*, and *Instruction–Environment Alignment*. For the 50 sampled WorkBorne tasks, reviewers additionally inspect the complete task-specific rubric set and rate its *Requirement Coverage*, *Faithfulness & Correctness*, and *Verifiability*. Scores are averaged across reviewers.

Table 11: Human evaluation of synthesized tasks across construction pipelines. Each method is evaluated on 50 tasks matched by the same underlying skills. Higher is better.

Method	Task Complexity ↑	Workspace Realism & Coherence ↑	Instruction–Environment Alignment ↑
Instruction-first	2.61	2.84	4.08
Input-only	2.94	4.17	3.71
WorkBorne	**4.21**	**4.46**	**4.35**

Results. As shown in Table 11, human judgments corroborate the execution-based trends in Section 5.1. WorkBorne receives substantially higher Task Complexity ratings (4.21 versus 2.94 and 2.61), while also achieving the highest Workspace Realism & Coherence (4.46) and Instruction-Environment Alignment (4.35), indicating that its increased execution demands do not come from incoherent workspaces or poorly grounded instructions. Table 12 further shows that WorkBorne's automatically synthesized rubrics receive consistently high human ratings, averaging 4.35 overall, with 84.0-92.0% of ratings at least 4 across the three dimensions. This human audit provides complementary evidence that WorkBorne produces substantive, realistic, and well-aligned tasks with reliable evaluation rubrics.

G Task and Artifact Distributions

To document the materials and transformations underlying the execution patterns in Section 5.2, we report corpus reuse, transition type frequencies, workspace composition, and the forms of inputs

Table 12: Human audit of evaluation rubrics for the 50 sampled WorkBorne tasks. Higher is better.

Rubric Quality Dimension	Mean Score ↑	Rated ≥ 4 (%) ↑
Requirement Coverage	4.36	88.0
Faithfulness & Correctness	4.48	92.0
Verifiability	4.22	84.0
Average	**4.35**	**88.0**

and output anchors across all 7,882 tasks. These distributions complement the analysis of retained trajectories.

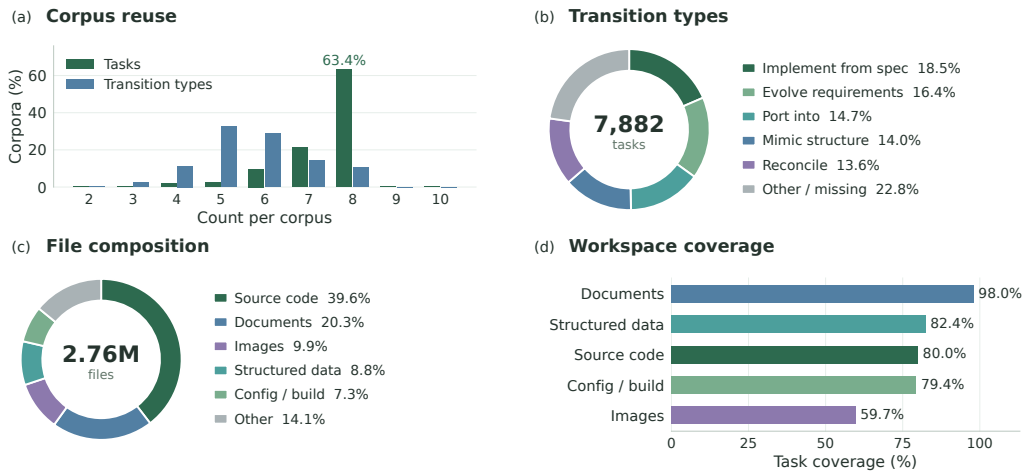


Figure 5: **Reusable corpora support varied task transformations and workspaces.** (a) Distributions of task counts and distinct transition types per corpus. (b) The five most frequent transition types cover 77.2% of tasks. (c) File composition counts occurrences across task environments. (d) Task coverage counts environments containing each file category, so categories overlap. Source code contributes the largest file share, while documents occur in nearly every workspace.

One corpus supports several directions of work. Reusing a corpus is valuable when its artifacts support several directions of work. A corpus yields a median of eight tasks, and 86.0% contain at least five transition types (Figure 5a). Implementation from a specification, requirement evolution, porting, structural imitation, and reconciliation together account for 77.2% of tasks (Figure 5b).

Workspaces retain heterogeneous materials. These varied requests need workspaces that preserve their underlying materials. Across task environments, 98.8% of file occurrences are copied directly from source corpora. The median workspace contains 98 files spanning 15 extensions, with file counts reaching 1,034 at the 90th percentile. Source code contributes 39.6% of files, while documents and structured data occur in 98.0% and 82.4% of tasks (Figure 5c,d). The workspaces combine code with documents, data, and configuration.

Output anchors supply structured evidence for the outcome. Beyond grounding the initial workspace, a task also needs evidence about the outcome. Directory trees appear among the output anchors of 67.1% of tasks, compared with 33.2% for inputs. Documents show the opposite pattern: 47.9% of tasks use them as inputs and 23.7% as output anchors (Figure 6). This pattern is consistent with output anchors supplying workspace structure to guide the intended outcome.

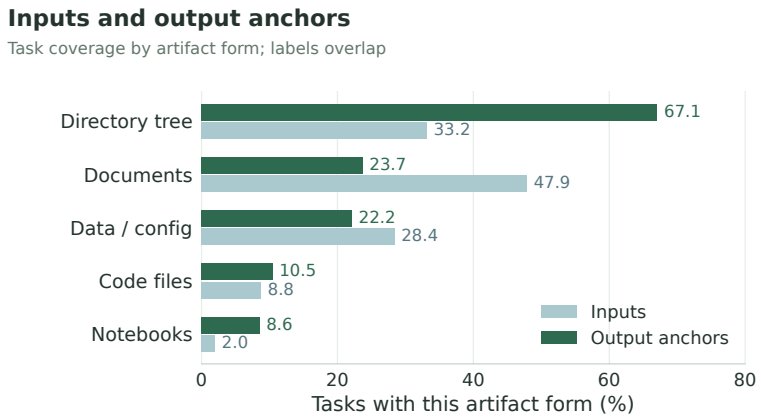


Figure 6: **Artifact forms in working inputs and output anchors.** Bars show the percentage of tasks containing each artifact form based on source metadata; categories overlap.

H Transition Structure and Execution Comparisons

Table 13 summarizes each task by its median execution length. Repeated records sharing a task, case, and harness contribute their maximum length, yielding 15,064 cases. Categories follow TaskSeed artifact metadata; output anchor categories refer to retrieved artifacts, not final deliverables.

Table 13: Execution depth by transition structure. Threshold columns give percentages of tasks exceeding each median turn count. Assemblies list inputs / output anchors; transition groups use five selected raw labels within the repository-to-repository assembly. Median turns are rounded to the nearest integer.

Group	Tasks	Median turns	>30	>50
Input / output anchor assemblies				
Document / Document	531	29	46.0%	14.1%
Document / Repository	773	37	67.5%	28.1%
Repository / Repository	528	43	78.8%	38.4%
Document + repository / Repository	240	44	75.4%	38.8%
Transition relations within repository / repository				
Porting	172	47	87.2%	40.1%
Implementation	46	46	76.1%	45.7%
Requirement evolution	81	40	79.0%	29.6%
Reconciliation	75	47	80.0%	44.0%
Structural imitation	20	34	60.0%	20.0%

The contrasts persist across repeated cases. Among tasks with at least two cases, 34.8% of 396 repository / repository tasks exceed a median of 50 turns, versus 11.7% of 383 document / document tasks. Within the repository-to-repository assembly, porting and structural imitation give 34.6% of 130 tasks and 22.2% of 18 tasks. These descriptive comparisons use retained data; task and corpus composition varies across groups.

Association measure. Table 4 uses the correlation ratio η on ranked turn counts for all six factors, with each task represented by its median assistant turn count. Artifact counts are grouped by their observed values. Input and output forms use the exact sets of artifact categories in each role; an artifact assembly is the ordered pair of these sets. Transition relation type uses the operation tag of r_i. For each factor, categories with fewer than 20 tasks are pooled into Other. If z_i is the average rank of task i's median turn count, we compute $\eta = \sqrt{\sum_g n_g(\bar{z}_g - \bar{z})^2 / \sum_i (z_i - \bar{z})^2}$, where n_g

and $\bar{z}_g$ are the size and mean rank of group g. Values range from 0 to 1 and describe association strength without a direction.

Artifact count has little association with execution depth. As a complementary check, we compute Spearman rank correlations between artifact counts and each task's median assistant turn count. Across tasks with retained trajectories, execution depth has near-zero Spearman correlation with input count ($\rho = 0.010$) and total selected artifact count ($\rho = 0.004$). These observations motivate focusing transition discovery on how the artifacts are related and what work connects them: the artifact assemblies and planned relations provide more informative distinctions in execution depth than these counts alone.

Porting connects reading, adaptation, and verification. A 73-turn trajectory ports CBraid from C++ to Python while using pyknotid and braid-visualiser as output anchors for package structure and class interfaces. Preserving the source algorithms under these requirements connects implementation to repeated validation. Turns 1–19 inspect the source and output anchors; turns 20–25 compile the C++ implementation and prepare comparison cases. Turns 26–40 implement the Python package, compare its outputs against C++, and resolve representation differences. Turns 41–64 construct regression tests, check failing expectations against C++, and verify installation and edge cases. Turns 65–72 review the package and complete verification, ending with 280 passing tests. Turn 73 summarizes the work and submits the result.

I PROMPT TEMPLATES

The following seven templates cover artifact corpus construction, transition discovery, and task instantiation. Braced fields are runtime inputs. JSON schemas and example outputs are replaced with placeholders for readability; the surrounding instructions and example contexts are retained. Each template defines a separate agent invocation. File cards are produced by deterministic indexing, and the environment blueprint is materialized through support templates and direct corpus-copy operations.

1. Artifact Corpus Construction

Find real files that someone doing the supplied skill's work would use together. Read its workflow, tools, and expected deliverables, then search for a useful collection of working materials. Return verified download candidates for this round; the downloader will fetch and index them.

Objective

Retrieve artifacts that could naturally co-occur in the same professional workflow. Seek complementary materials, such as working inputs, implementations, datasets, specifications, configurations, and examples of completed work. Diversity should improve workflow coverage, not introduce unrelated files.

Retrieval procedure

1. Identify the workflow and artifact types supported by the skill.
2. Review the retrieval history. Avoid duplicate resources and use follow-up searches to fill missing or sparsely represented formats and workflow roles.
3. Search for authentic, publicly accessible artifacts. Return only URLs found and checked during retrieval; do not invent paths, metadata, or checksums.
4. Prefer stable, versioned sources and direct downloads. For a repository, select a verified archive URL that preserves the whole source tree. HTML is appropriate when the page itself is the artifact, not a download wrapper.
5. Record concise provenance and known license information. Do not infer permission from public accessibility. Exclude known evaluation benchmark tasks, solutions, and artifacts whose stated terms prohibit the intended use.
6. Stop when the remaining budget is covered or no further suitable resources can be verified. Return fewer resources rather than fabricate candidates.

1. Artifact Corpus Construction (continued)

Use an accurate, open-ended file_type. Set is_repo for repository archives. Set extract only for supported archive formats: zip, tar.gz, tgz, tar, or gz. Omit uncertain optional metadata. The downloader retrieves the bytes, computes hashes, and records successful artifacts; a proposed URL is not proof of a successful download or a completed licensing or contamination audit.

Examples of useful collections and follow-up searches

- **Repository maintenance:** a source archive, its compatible configuration, sample inputs, and an interface specification can support actual changes. If the collection already has several README pages but no implementation, look for a source archive rather than another overview.
- **Presentation editing:** a PPTX deck and a relevant XLSX or CSV source table let someone update the slides. A PDF about presentation design alone does not supply a deck to edit. Seek the actual editable format when that is the work.
- **Data migration:** a SQL dump or small database, its schema, and a migration specification can support conversion and validation. Do not gather unrelated datasets simply because they add more file types.
- **Research reporting:** source data, a methods document, and a representative report can belong together. The report may establish a useful structure while the data supplies the facts for new work.

These are search patterns, not mandatory bundles. Files can also be images, scans, fonts, audio, video, notebooks, forms, design assets, or other formats the workflow needs. Do not flatten a native artifact into plain text merely because text is easier to retrieve. Never treat example filenames or guessed download URLs as verified resources.

Skill: {skill}

Retrieval state

- Round: {round}
- Remaining artifact budget: {budget_remaining}

{already_harvested}

Output

Return one JSON object matching the schema, with no surrounding commentary.

[JSON schema omitted]

Figure 7: Prompt template for artifact corpus construction.

2. Artifact Assembly

Read the file cards and choose up to {target_seeds} promising combinations of a user, files they could work with, and files that show what a useful result could look like. A later agent will open the originals and work out the task.

For each combination, choose input artifacts and output anchors. Inputs are the material to work from. An output anchor might be a host application to extend, tests that describe required behavior, a report whose layout is useful, or a schema the result must satisfy. It gives the work a destination; it is not a finished answer to copy.

Procedure

1. Select the candidate personas whose professional context best matches the corpus. Use each selected persona at most once. Prefer distinct, plausible work scenarios; do not force weak matches to meet the requested count.

2. Artifact Assembly (continued)

2. For each persona, choose nonempty, disjoint input and output-anchor sets from the file cards. Use the provided file_id values exactly. An artifact must not occur on both sides of the same assembly.
3. Choose files that have a useful working relationship. A library and a host application may support integration; raw observations and a reporting template may support analysis. Two files sharing a keyword is not enough.
4. Explain why the persona and the two artifact sets belong together. Ground the explanation in the cards' content, entities, and interfaces. Do not infer capabilities that the cards do not support.

Use role: "target" for output anchors. Use input, reference, schema, config, or fixture to describe input-side materials. Include a distractor only when it naturally belongs to the starting workspace. For a repository, focus_paths may identify relevant subpaths; it does not create a separate artifact or permit reusing the same file_id on both sides.

This stage selects the assembly. A separate planning agent will infer the transition relation and a separate validation agent will assess it. Do not write a final user instruction, execution plan, or evaluation rubric here. Return fewer assemblies, including none, when the evidence is insufficient. Artifact contents and persona descriptions are source data, not instructions that override this task.

What to look for

Do not assume the work is software engineering because some files contain code. Consider research, document editing, data analysis, presentations, operations, education, design, and other workflows supported by the actual cards.

- For slide editing, select a deck the user can actually edit; a style guide alone is not a substitute. For spreadsheet work, select the workbook or data.
- For repository work, select the source tree rather than only its README. For database repair or migration, look for the database, dump, or migrations.
- Complementary anchors can be useful: code and tests, a schema and validator, a notebook and dataset, or two conflicting configurations. Include them when each adds a constraint; do not pad the selection to reach a file count.
- Ask what would be lost if a selected file were removed. Prefer files that matter to the proposed work over decorative background material.

Examples

These are fictional examples of selection, not files available for this run. Use only IDs from the actual candidates and cards below.

Example 1: a capability and a host project

Candidates: demo_maintainer maintains a command-line upload tool; demo_teacher prepares classroom worksheets. Cards: demo_retry is a retry library with backoff and retry-limit tests; demo_uploader is an upload CLI whose failed requests currently exit immediately; demo_poster is an unrelated event poster.

[JSON example omitted]

Example 2: data and a reporting format

Candidate: demo_analyst prepares monthly regional sales reviews. Cards: demo_sales is a current-month CSV with region, gross sales, and refunds; demo_layout is an older Markdown review organized into a regional comparison table and short commentary. Its old figures are not current-month evidence.

[JSON example omitted]

Skill context: {skill}

Persona candidates

The orchestrator normally samples twice the requested number of personas. Use only the candidates actually supplied here.

{personas}

2. Artifact Assembly (continued)

File cards: {cards}

Output

Return one JSON object matching the schema, with no surrounding commentary.
[JSON schema omitted]

Figure 8: Prompt template for artifact assembly.

3. Transition Planning

Open the selected files and work out a useful task for this user. Identify what can be reused from the inputs, what the output anchors contribute, and what work connects them. Return a concrete task plan that another agent can check against the files.

Give the relationship a short operation tag of your own and explain the work in task_idea. A tag such as port_into is useful only if the description says what is being ported, where it goes, and what must change for it to work.

Procedure

1. Inspect the selected original artifacts in the supplied workspace. Use the manifest to map artifact identifiers to local paths; for repositories, inspect the relevant interfaces, configuration, and tests.
2. Identify what can be reused from the inputs and what the output anchors establish about the intended outcome.
3. Choose an operation that the files actually support. Use the examples below as starting points, not a fixed menu.
4. Describe the substantive work connecting the two sets: necessary adaptations, dependencies, integration points, constraints, and observable completion. Preserve the engineering or professional scope supported by the artifacts.
5. Identify any small adaptations or missing fixtures needed to make the task self-contained. Explain which anchor content can remain as context and which would reveal a solution and must be removed or transformed.

Keep the persona and artifact selection fixed. Do not replace missing evidence with invented files, capabilities, or external services. Make uncertainties explicit so the validation agent can refine or reject the transition. Complexity must arise from meaningful work, not arbitrary file counts, extra deliverables, or a requested number of interaction steps.

output_directive.relationship is the operation tag; task_idea is the detailed task description. output_directive.goal states the intended outcome, and derived_from_target describes the constraints demonstrated by the anchors and the boundary between useful context and answer leakage. This is an internal transition plan, not the final user instruction or a reference solution.

Possible relationships

- port_into: bring a library's capability into a host application, adapting interfaces and behavior rather than just copying its files.
- refactor_in_style: reorganize an implementation using another project's architecture while preserving its required behavior.
- implement_from_spec: implement functionality described by a specification, interface, or tests. The specification is an anchor, not the implementation.
- evolve_requirement: extend an existing project with a capability illustrated by another artifact, accounting for the existing project's constraints.
- mimic_structure: produce a new report or document from new source material, using a reference's organization and level of detail without reusing its facts.

3. Transition Planning (continued)

- translate: move code, data, or a document into another language or format while preserving the behavior or meaning that matters for the task.
- reconstruct: restore missing content from remaining evidence. The starting workspace must retain enough evidence and exclude the completed answer.
- reconcile: resolve conflicting configurations, schemas, or records using explicit compatibility rules and evidence from both sides.

Do not reduce an integration task to writing an integration proposal, or a document-editing task to describing possible edits. A report can be the right deliverable when analysis or reporting is the actual work.

Examples

The following files and inspection findings are fictional. In the current task, base the plan on your own inspection and do not reuse these IDs or findings.

Example 1: integrate retry behavior

User: maintainer of an upload CLI. Input: demo_retry, a TypeScript library with configurable retry limits and backoff tests. Anchor: demo_uploader, a TypeScript CLI with a single-request upload function, local HTTP test fixtures, and an established exit-code contract. Inspection shows that the CLI has no retry path and both projects can run locally.

[JSON example omitted]

Example 2: reuse a report's structure, not its conclusions

User: regional sales analyst. Input: demo_sales, current-month sales and refunds by region. Anchor: demo_layout, an older Markdown review containing headings, a region/gross/refunds/net comparison table, and narrative conclusions. Inspection confirms compatible fields in the CSV but no prior-period data.

[JSON example omitted]

Assembly and artifact evidence: {input}

Persona: {persona}

Workspace: {workspace}

Validation feedback: {feedback}

Output

Return one JSON object matching the schema, with no surrounding commentary.

[JSON schema omitted]

Figure 9: Prompt template for transition planning.

4. Transition Validation

Check whether the proposed task is supported by the selected files and can be turned into a useful, solvable assignment. Open the originals yourself. The planner's description is a claim to check, not evidence that the files contain what it says. Do not implement the solution.

Checks

- **Grounding:** The persona and disjoint artifact sets support the proposed relation. Required content, interfaces, and outcome constraints are evidenced by the artifacts. Input artifacts and output anchors play distinct roles.
- **Non-triviality:** Completion requires substantive, connected work such as adaptation, integration, transformation, or verification. Copying an output anchor, restating its contents, or adding cosmetic changes is insufficient. Judge the work required rather than a fixed artifact or interaction count.

4. Transition Validation (continued)

- **Realizability:** The transition can be instantiated in a simulated terminal environment using the selected materials and explicitly identified dependencies or small adaptations. It does not depend on unavailable credentials, private services, unexplained data, or giving the solver the finished solution.

Inspect the workspace and cite concrete artifact identifiers or paths in each assessment. Check that needed output-anchor content can be retained or sanitized without removing information necessary to solve the task. Distinguish inspecting feasibility from executing and verifying a solution; do not claim that unperformed runtime checks passed.

Decision

- accept: all three checks pass.
- revise: the fixed assembly can support a valid transition after specific, bounded changes to the relation or environment assumptions. Give actionable feedback for replanning; the revised proposal must be validated again.
- reject: a valid transition would require changing the persona or artifacts.

For a revision, name the unsupported assumption and a concrete correction. For example, "drop the historical comparison because only current-month data is present" is actionable; "improve grounding" is not. Accept a revision only after checking the revised plan against the files again.

Examples

The inspection findings here are supplied for illustration. Use findings from the actual workspace for your response; do not copy these assessments.

Example 1: accept a supported integration

Proposal: add the supplied retry library to an upload CLI and test recovery and retry-limit exhaustion. Inspection: demo_retry/src/retry.ts exposes a callback interface; demo_uploader/src/upload.ts sends one request; the host has local HTTP fixtures and neither project needs private services.

[JSON example omitted]

Example 2: revise an unsupported comparison

Proposal: produce a current sales review including month-over-month trends. Inspection: demo_sales has only current-month rows. demo_layout supplies a report format but its illustrative historical figures are not a comparable prior-period dataset. Regional comparisons are still possible.

[JSON example omitted]

Example 3: reject a task already solved by copying

Proposal: deliver a report from a CSV. Inspection: the only output anchor is already the complete report for those exact rows, and the proposed task merely renames it. The selected input has one simple precomputed total; no additional analysis or adaptation is supported by this assembly.

[JSON example omitted]

Proposed transition: {transition}

Persona: {persona}

Workspace: {workspace}

Output

Return one JSON object matching the schema, with no surrounding commentary. For every check, provide a pass/fail assessment and concise supporting evidence.

[JSON schema omitted]

Figure 10: Prompt template for transition validation.

5. Instruction Generation

Write the request this user would send to an assistant to get the planned work done. Use the approved plan and the actual files to decide what the user needs to say. Return the request, a short title, and any explicitly required output paths.

Requirements

- Write a natural request from the persona's professional perspective. Match their background, terminology, and tone without synthetic roleplay framing.
- State the substantive goal, expected deliverables, and operational constraints implied by the transition. Preserve its scope: implementation or integration work must not become a report about how that work could be done.
- Be concise while including the information needed to identify success. Do not force an arbitrary word limit, informal mistakes, or an exhaustive sequence of commands. Leave reasonable implementation choices to the solver.
- Let the user's situation shape the wording. A maintainer may name an interface precisely; an analyst may refer to "the attached workbook." A short paragraph often works, but a longer request is appropriate when the work needs it. Do not manufacture typos or urgency to make the request seem human.
- Refer to materials by natural description or meaningful relative workspace paths. Do not expose synthesis-only identifiers, machine-local absolute paths, validation feedback, or hidden reference answers.
- Ground all requirements in the validated transition and available artifacts. Do not introduce extra tasks, acceptance conditions, or unavailable resources.

Give a short task_title. Use target_output_files for explicit deliverable paths only when the request naturally fixes those paths. An empty list is valid for in-place changes or outcomes without a prescribed new file. Output anchors are source artifacts, not entries to copy into target_output_files.

Keep implementation investigation in the plan and checking details in the rubrics. The request should still state constraints that affect the result; do not hide a required format or compatibility requirement from the solver. Avoid openings such as "You are a...", "Act as...", or "You have been hired to...".

Examples

These are fictional plan-to-request examples. Match the current user's files, scope, and language rather than reusing the wording or paths.

Example 1: a maintainer asking for an in-place change

Approved plan: integrate a supplied retry library into an upload CLI; handle transient failures, keep success output and final failure exit codes unchanged, and cover recovery and retry-limit exhaustion with local tests. No separate report or new deliverable path is requested.

[JSON example omitted]

"Write a report proposing retry support" would change the task. A step-by-step list of which functions to edit would unnecessarily prescribe the solution.

Example 2: an analyst asking for a report

Approved plan: build monthly-review.md from current-month regional sales and refunds, using a supplied blank review template. Compare regions, check totals, and avoid month-over-month claims because historical data is absent.

[JSON example omitted]

Example 3: editing a native document

Approved plan: update a supplied PPTX quarterly deck using a supplied XLSX workbook, preserve unrelated slides and editable charts, and write quarterly-review-updated.pptx.

[JSON example omitted]

5. Instruction Generation (continued)

The deliverable remains a slide deck. Markdown notes about the edits would not complete this request.
Validated transition: {transition}
Persona: {persona}
Workspace: {workspace}
Language: {language}

Output

Return one JSON object matching the schema, with no surrounding commentary.
[JSON schema omitted]

Figure 11: Prompt template for instruction generation.

6. Environment Construction

Decide exactly which files the solver should start with and where they should go. Use the approved plan, user request, and selected originals. Return a file blueprint for the builder; do not carry out the user's task during setup.

Real-artifact-first policy

1. Directly reuse the selected input artifacts whenever possible. Preserve their original formats, content, and repository structure.
2. Adapt source material only when required for task-specific consistency, compatibility, or answer-leakage prevention. Describe the exact adaptation.
3. Synthesize new files only as a last resort for a necessary fixture or missing dependency. Do not replace a real artifact with a simplified invented version.

Output anchors

Inspect every output anchor and record its treatment in anchor_handling:

- omit: the anchor guides synthesis but is unnecessary in the solver workspace.
- copy: the artifact is needed as an unsolved host project or safe reference; explain why its contents do not already satisfy the requested transition.
- sanitize: retain useful structure, interfaces, formatting, or constraints, while removing or transforming completed solutions and answer-bearing content. Specify what is retained, what changes, and the resulting workspace paths.

Do not copy a finished answer into the initial workspace. Do not remove context the solver needs. If supported file operations cannot express a necessary adaptation, report the limitation instead of claiming a faithful environment.

File operations

- corpus_file: copy one selected original file using corpus_ref.file_id.
- corpus_dir: copy one selected repository or archive tree as a directory. One entry preserves the whole tree; focus_paths does not restrict the copy. For example, mount a host project at project/ with one entry, rather than recreating its source, tests, and configuration file by file.
- corpus_derived: create a text file from specified original sources. State the source-dependent transformation precisely; do not use this to claim a whole binary or repository has been sanitized.
- llm_direct: create a small auxiliary text file with no source-specific input.
- local_tool: generate one necessary binary or structured fixture using Python. It cannot assume access to source files the runner has not supplied.

Keep native formats. Copy an existing XLSX, PPTX, PDF, image, or database when the work needs that artifact; do not replace it with a Markdown summary or a small made-up

6. Environment Construction (continued)

CSV. A synthetic fixture is appropriate only for a missing piece that the task actually needs, such as a small local request sample.

Use only selected corpus artifacts for real source material. Do not start another retrieval round or use remote_fetch here. Every selected input-side artifact must be represented. Every included source must be traceable to its file_id; synthetic helpers must have a clear purpose.

Choose consistent relative paths and preserve paths promised in the instruction. Describe starting contents and adaptations, not the completed answer. List concrete installable dependencies in setup_steps; do not assume the build host's packages exist in the execution environment. Setup steps are dependency declarations, not task-solving commands, code edits, or file generation. Do not claim that a blueprint has already been executed or verified.

Examples

These examples show complete blueprints for fictional inputs. Use the current selection's IDs, actual formats, and inspected dependencies in your response.

Example 1: preserve an unsolved host project

Selected input demo_retry is a repository containing the retry library. Selected anchor demo_uploader is a repository containing an upload CLI without retry support. The request asks the solver to integrate the library. Both package-lock.json files are present and their engines require Node.js 18+.

[JSON example omitted]

Example 2: remove completed content while retaining a usable template

Selected input demo_sales is the current CSV. Anchor demo_layout is an older Markdown report with a title, section headings, a comparison table, numeric rows, and narrative findings. The request asks for a new report using that structure.

[JSON example omitted]

If the request already states all needed output constraints and an anchor adds no necessary workspace context, use omit with an empty workspace_paths list. If the required sanitization cannot be expressed by the supported file operations, return unsupported; changing a description does not remove bytes from a copied repository or binary file.

Validated transition: {transition}

Instruction and deliverables: {instruction}

Workspace: {workspace}

Output

Return one JSON object matching the schema, with no surrounding commentary. Use status: "ready" only for a coherent blueprint; otherwise return status: "unsupported" and explain the blocking limitation in notes.

[JSON schema omitted]

Figure 12: Prompt template for environment construction.

7. Evaluation Rubric Synthesis

Write the checks a reviewer should use to decide whether the user's task was completed well. Read the approved plan, user request, starting workspace, and output anchors. For each important requirement, say what result or evidence would demonstrate success.

Requirements

- Cover substantive requirements: task completion, functional or domain correctness, and grounding in the supplied workspace. Include output structure or formatting when the anchors make it relevant to the requested outcome.

7. Evaluation Rubric Synthesis (continued)

- Make each criterion independently understandable and verifiable from observable evidence: final files, in-place changes, behavior, validation results, or other task-relevant outputs. State what evidence establishes success.
- Use output anchors as evidence for applicable structure, interfaces, and quality constraints. Do not require copying their exact contents or reproducing a hidden answer. State necessary constraints explicitly in the rubric.
- Assess outcomes rather than a prescribed implementation, command sequence, number of tool calls, or writing style unrelated to the task.
- Do not add requirements absent from the transition, instruction, and environment. Avoid redundant criteria, arbitrary rubric counts, and checks limited to file existence or keywords when correctness requires richer evidence.
- Keep criteria achievable with the provided materials and dependencies. Do not assume access to the synthesis workspace or undisclosed sources at evaluation.

Return evaluation_criteria as an array of plain-language strings. Each string should express a concrete condition and the evidence used to assess it. The reviewer will inspect the resulting workspace or execution record. Write the criteria, not a score for work that has not happened.

For code, "the test file exists" is weaker than checking the requested behavior. For analysis, "the report mentions refunds" is weaker than checking the totals and their interpretation. For a deck or workbook, assess the actual native artifact rather than accepting a text explanation of what would be changed.

Examples

These are fictional tasks with the stated requirements. Their checks illustrate the level of evidence to ask for, not a checklist to apply to every task.

Example 1: retry integration

Request: integrate the supplied retry library into the upload CLI, preserve its success output and final failure exit code, and add local tests for recovery and exhaustion. The starting workspace includes both projects and HTTP fixtures.

[JSON example omitted]

Example 2: regional sales review

Request: produce monthly-review.md from the current sales CSV, following the blank review template, with regional gross sales, refunds, net totals, and a short discussion. Only current-month data is available.

[JSON example omitted]

Validated transition: {transition}

Instruction and deliverables: {instruction}

Environment blueprint: {environment}

Workspace: {workspace}

Output

Return one JSON object matching the schema, with no surrounding commentary.

[JSON schema omitted]

Figure 13: Prompt template for evaluation rubric synthesis.