AGENT FOREST: CONTROLLED COMPLEXITY GROWTH FOR RL ENVIRONMENTS VIA A MULTI-AGENT ORCHESTRATION SYSTEM

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models are increasingly expected to act as agents that use tools, follow user constraints, observe changing environment state, and complete long-horizon tasks. Training and evaluating these agents requires large collections of challenging, verifiable tasks, yet constructing such data at scale remains difficult. Existing synthetic task-generation methods frequently rely on single-pass prompting, producing tasks that are limited in complexity, weakly grounded in executable environments, and difficult to expand while preserving consistency. We introduce Agent Forest, a multi-agent framework for controlled complexity growth that systematically transforms trusted seed tasks into more complex descendants while preserving executability and verification guarantees. Agent Forest uses five specialized agents that increase complementary dimensions of interaction complexity: broader tool use, branching dependencies, long-range memory requirements, partial observability, and task composition. Rather than simply rewriting instructions to create superficial difficulty, these agents reason over available tools and environment state to modify the underlying interaction problem itself. Because transformations can accumulate and be applied in different orders, a single seed task can generate multiple descendants with distinct complexity profiles, forming a *forest* of task variants that grows both the scale and diversity of trusted agentic datasets. We apply Agent Forest to three agentic environments, τ^2-Retail, τ^2-Airline, and General-Agent, transforming trusted seed tasks into diverse descendants with varying complexity profiles. As interaction complexity accumulates, the resulting tasks become substantially more challenging for strong solver agents. On τ^2-Retail, Qwen 3.5 397B drops from 75.4% Pass@1 on the original seed tasks to 54.7% on fully transformed descendants. On τ^2-Airline, Gemma 4 31B drops from 66.0% to 28.6%, while GLM-5.2 drops from 72.0% to 53.6%. On General-Agent, Gemma 4 31B drops from 68.8% to 25.6%. Overall, accumulated complexity growth reduces Pass@1 by 18.4–43.2 percentage points relative to the original seed tasks, demonstrating that Agent Forest can systematically generate more complex agentic tasks while preserving executability and validity.

1 INTRODUCTION

Language models are increasingly used as agents that interact with users, call tools, observe external state, and take actions that change that state. Realistic tasks often require gathering information, reasoning over tool outputs, retaining earlier constraints, recovering missing details, and coordinating actions over long horizons. Existing agent benchmarks have helped measure these abilities (Shridhar et al., 2021; Yao et al., 2023; Qin et al., 2023; Liu et al., 2025b), but constructing large and diverse collections of such tasks remains difficult. Improving agents requires training and evaluation data that extend beyond familiar task patterns and expose models to broader tool use, longer dependencies, partial observability, long-range memory, and coordination across multiple objectives. This motivates not simply *more* agentic data, but methods for generating diverse, executable, and increasingly complex agent experiences.

A natural approach is to synthesize new tasks from existing ones. Prior work has shown that language models can generate instructions, evolve tasks, and create data for agent training (Wang et al., 2023; Xu et al., 2025; Mitra et al., 2024; Xie et al., 2026; Shi et al., 2025). However, agentic task complexity is determined by the interaction itself: which tools must be used, what information is available, how actions modify state, and how later decisions depend on earlier observations. Linguistically complicating an instruction is therefore insufficient. Effective transformations should change the underlying interaction problem while preserving executability and consistency. For example, a transformation may remove information from the instruction and require the agent to recover it through tool use before a later action (Figure 1).

We study this problem as *controlled complexity growth*: increasing explicit dimensions of interaction complexity while preserving executability and verifiability. We propose Agent Forest, a multi-agent framework that transforms trusted

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

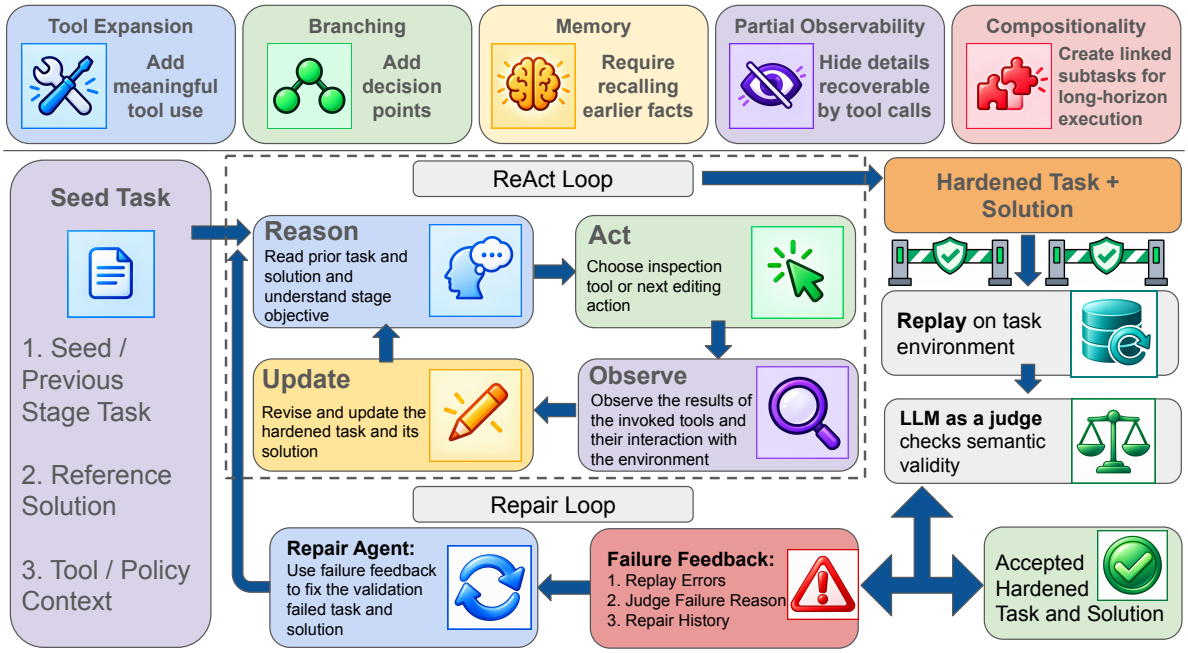


Figure 1: Overview of Agent Forest. Starting from a seed task, a specialized complexity-growth agent follows a ReAct loop, reasoning over the task, available tools, and environment state before generating a transformed descendant and updated reference solution. The resulting descendant is validated through execution replay and semantic verification. Accepted descendants become parents for subsequent transformations, enabling task lineages to grow along different complexity-growth paths and form a forest of verified descendants. Invalid descendants receive repair feedback and enter a repair loop before revalidation or rejection.

seed tasks through five specialized agents. *Tool Expansion* broadens required tool interactions; *Branching Dependencies* introduces observation-dependent decisions; *Memory* requires earlier information to be retained and reused; *Partial Observability* hides information that must be recovered through interaction; and *Compositionality* combines multiple interdependent requirements. Rather than transforming text alone, these ReAct-style agents inspect available tools and environment state before jointly modifying the task and its reference solution.

These transformations form a *forest* rather than a single fixed pipeline. Different copies of the same seed can encounter agents in different orders, producing descendants with distinct complexity profiles (Figure 1). For example, one path may apply Tool Expansion followed by Partial Observability, while another may introduce Branching before Compositionality, exposing the resulting descendants to different combinations of interaction requirements. Complexity can therefore accumulate along a lineage, providing *depth*, while alternative transformation paths provide *diversity*. This allows a relatively small collection of trusted seeds to support a much larger family of related tasks that differ in both the amount and type of complexity they acquire.

Crucially, increased complexity is useful only if generated tasks remain valid. After each transformation, Agent Forest checks that the instruction, reference trajectory, and resulting environment state remain mutually consistent. Candidates that fail validation are repaired when possible or filtered before further propagation, preserving executability, verifiability, and a traceable lineage from each accepted descendant to its seed.

We apply Agent Forest to τ^2-Retail, τ^2-Airline (Barres et al., 2025), and General-Agent (Senghaas, 2026). Across these environments, Agent Forest substantially expands the collection of validated descendants while increasing diverse forms of interaction complexity. Fully transformed descendants reduce Pass@1 by 18.4–43.2 percentage points relative to the original seeds across strong solver agents, exposing limitations in tool use, dependency reasoning, long-range memory, information recovery, and multi-objective coordination. Overall, Agent Forest provides a scalable way to grow trusted agentic datasets into diverse families of verified, environment-grounded tasks for training and evaluating increasingly capable agents.

2 Related Work

Agentic benchmarks and tool-use environments. Recent benchmarks study language models as agents that act in interactive environments. ALFWorld connects language reasoning with embodied action (Shridhar et al., 2021), WebShop evaluates agents in a simulated shopping environment (Yao et al., 2023), ToolLLM studies large-scale API use (Qin et al., 2023), and AgentBench evaluates LLMs across several agent settings (Liu et al., 2025b). τ^2-Bench further evaluates conversational agents in dual-control environments where both the user and the agent can affect shared state (Barres et al., 2025). These benchmarks show that agent performance depends on tool use, state tracking, user interaction, and multi-step decision making. However, they mainly provide fixed task collections. Our goal is to start from such seed tasks and automatically generate diverse, verified descendants with progressively richer forms of environment-grounded interaction complexity.

LLM-based synthetic data and task evolution. A large body of work uses language models to synthesize or transform data. Self-Instruct generates new instruction-following examples from seed tasks (Wang et al., 2023), while WizardLM evolves existing instructions to make them more complex (Xu et al., 2025). More recent systems target agentic data directly: AgentSynth generates tasks for computer-use agents (Xie et al., 2026), TaskCraft creates verifiable multi-tool tasks (Shi et al., 2025), and WebRL builds self-evolving curricula for web agents (Qi et al., 2025). These works show that LLMs can expand data and induce difficulty, but the transformation is often driven mainly by language-level generation or high-level task rewriting. In contrast, Agent Forest uses ReAct-style task-modifying agents that can inspect tools, query the environment, observe state, and make grounded changes to both the task and its reference trajectory.

Multi-agent synthetic data generation and verification. Several works use multiple agents to improve synthetic data quality. MAG-V uses generation and verification agents for synthetic data and trajectory checking (Sengupta et al., 2025). Matrix studies peer-to-peer multi-agent data generation at scale (Wang et al., 2026a). ToolACE uses multi-agent synthesis and verification for function-calling data (Liu et al., 2025a). These systems demonstrate the value of multi-agent workflows, but their agents primarily generate, review, or coordinate data production. Agent Forest uses agents differently: each agent is a grounded task transformer. It observes the current task state, modifies the task along a particular difficulty dimension, updates the reference trajectory, and passes the result to the next agent. This lets difficulty accumulate through a lineage rather than being produced by a single rewrite.

Environment synthesis and verifiable agent training. Another direction expands agent training by synthesizing environments or using self-improving training loops. Agent World Model, Agent-World, and AutoForge generate environments for agentic reinforcement learning (Wang et al., 2026c; Dong et al., 2026; Cai et al., 2025). Self-evolving synthetic data with verifiable-reward RL and AgenticQwen use data flywheels and verifiable feedback to improve tool-using agents (Gao et al., 2026; Lyu et al., 2026). These works focus on creating new environments, training loops, or large-scale self-improvement pipelines. Agent Forest instead assumes a domain already has tools, state, and seed tasks, and asks how to systematically harden those tasks while preserving executable correctness.

Positioning. The main distinction of Agent Forest is that it treats synthetic agentic data generation as *environment-grounded complexity growth*. Prior work has shown that LLMs can rewrite instructions, synthesize tasks, and verify generated data. In contrast, Agent Forest transforms tasks through modular ReAct-style agents that inspect the environment, available tools, and execution context before modifying the underlying interaction problem. Each agent increases a specific dimension of interaction complexity, and different agent orderings produce different evolutionary paths through the task *forest*. Generated tasks are accepted only after execution and semantic validation, resulting not simply in a larger synthetic dataset, but in a collection of verified task lineages whose complexity growth can be controlled, measured, and traced back to trusted seed tasks.

3 Method

3.1 Overview

Agent Forest takes a trusted seed set of agentic tasks and grows it into a forest of verified task descendants. Each seed task consists of an environment, a set of tools, and a reference solution. Through controlled complexity growth, tasks are transformed along multiple evolutionary paths that introduce new interaction requirements while preserving executability, verifiability, and consistency with the underlying environment.

Figure 1 provides an overview. Starting from a verified parent task, a specialized complexity-growth agent inspects the task, reference solution, available tools, and relevant environment state before proposing a transformed descendant and updated reference solution. The candidate is then evaluated through execution replay and semantic validation. Valid descendants are retained and may undergo further complexity-growth transformations, while invalid descendants receive validation feedback for repair and are discarded if they cannot be made consistent.

The framework is modular: each complexity-growth agent increases a specific dimension of interaction complexity, and multiple transformations can accumulate along a task lineage. Different choices and orderings of agents produce distinct descendants from the same seed, yielding the *forest* structure and enabling multiple paths of controlled complexity growth.

3.2 Environment Grounded Complexity Growth via ReAct Agents

A central design principle of Agent Forest is that interaction complexity should be grounded in the environment rather than introduced solely through language. Before applying any transformation, we replay the seed reference solution to verify that it executes correctly and establish a trusted starting point. We then summarize the task-relevant structure, including the tools used, entities involved, state transitions, and dependencies exhibited by the reference solution. This environment-grounded context enables subsequent hardening agents to reason about how complexity can be increased while preserving executability and consistency.

Each specialized agent operates through a **ReAct-style reasoning and acting loop** (Yao et al., 2022). Given a parent task and a complexity-growth objective, the agent can inspect the reference solution, query relevant environment records, examine tool schemas, execute tools, and observe their outputs before deciding how to transform the task. It then jointly produces an updated instruction and reference solution grounded in these interactions.

This ReAct loop is crucial because interaction complexity arises from the underlying environment, not solely from linguistic variation. Modifying tool requirements, dependencies, observability, or state transitions requires understanding what the environment actually supports. By reasoning over and interacting with the environment before transformation, an agent can introduce new dimensions of complexity while preserving executability, consistency, and verifiability.

3.3 Complexity Growth Agents

Agent Forest uses five specialized agents that share the same environment-grounded transformation procedure but target complementary dimensions of interaction complexity. **Tool Expansion** broadens the tool interactions required to solve the task by introducing additional information gathering, verification, or tool use, requiring the solver to integrate evidence across a wider set of environment interfaces. **Branching** introduces observation-dependent decisions whose outcomes influence subsequent actions, requiring the solver to adapt its behavior based on intermediate environment state. **Memory** separates when task-relevant information is acquired from when it must be used, requiring values, constraints, or preferences introduced earlier in the interaction to be retained and correctly reused across intervening actions. **Partial Observability** removes information that was directly available in the parent task while ensuring that it remains recoverable from the environment, requiring the solver to acquire missing information through interaction before acting correctly. **Compositionality** combines multiple goals, constraints, or required state changes into a single task, requiring the solver to coordinate several interdependent requirements rather than satisfying each independently.

3.4 Complexity Growth Paths

The complexity-growth agents are compositional rather than tied to a fixed curriculum. A seed task can be transformed by different subsets of agents and in different orders, allowing distinct dimensions of complexity to accumulate along different paths. For example, one path may first broaden tool use and then introduce partial observability, while another may introduce branching dependencies before composing additional requirements.

The resulting *forest* provides two complementary forms of expansion. *Depth* arises from accumulating multiple complexity-growth transformations along a task lineage, while *diversity* arises from applying different combinations and orderings of transformations to the same seed. Each accepted descendant retains the sequence of transformations that produced it, providing explicit provenance and a traceable record of how its interaction complexity evolved.

3.5 Verification and Repair

Environment-grounded generation does not guarantee that every proposed transformation is valid. A transformed task may require an unsupported interaction, contain an incorrect reference solution, or produce an environment state that

does not satisfy the modified instruction. Agent Forest therefore validates every candidate before it can be retained or further transformed.

Validation consists of two complementary checks. First, the updated reference solution is replayed from a fresh environment state to verify that the required tool interactions execute successfully. Second, a language model acting as a semantic judge checks whether the transformed instruction, executed trajectory, and resulting state change are mutually consistent.

Candidates that fail either check receive validation feedback and are passed to a repair agent. The repaired candidate is evaluated through the same validation procedure until it passes or the repair budget is exhausted. Only candidates that pass both execution and semantic validation are retained and propagated along subsequent complexity growth paths.

The complete complexity-growth procedure is summarized in Algorithm 1 in Appendix A.

4 Experimental Setup

We evaluate Agent Forest on τ^2-Retail and τ^2-Airline (Barres et al., 2025), with 114 and 50 verified seed tasks, respectively, and General-Agent (Senghaas, 2026), with 48. The τ^2 domains contain structured conversational tool-use tasks, while General-Agent contains broader stateful tasks with task-specific tools and databases.

Task Generation. For the main experiments, we cumulatively apply the complexity-growth agents in this order: **Tool Expansion** $\rightarrow$ **Branching** $\rightarrow$ **Memory** $\rightarrow$ **Partial Observability** $\rightarrow$ **Compositionality.** We denote the seeds as Stage 0 and successive descendants as Stages 1–5. This ordering is fixed for controlled stage-wise comparison, although Agent Forest supports alternative agent orderings and complexity-growth paths. Across all environments, Qwen 3.5 397B (Qwen Team, 2026) serves as the generation model within the ReAct-based framework (Yao et al., 2022), and GPT-OSS 120B (Agarwal et al., 2025) serves as the semantic judge. Generated descendants undergo the verification and repair procedure in Section 3; only candidates with executable reference solutions that pass semantic validation are retained. We characterize accepted descendants by the tool calls, database state-changing writes, and distinct tools in their verified reference solutions.

Downstream Agent Evaluation. We evaluate task-level Pass@1 and report tool calls, database state-changing writes, and distinct tools in solver trajectories. On τ^2-Retail, Qwen 3.5 397B (Qwen Team, 2026) is evaluated with a ReAct solver (Yao et al., 2022). On τ^2-Airline, Gemma 4 31B (Gemma Team et al., 2026) and GLM-5.2 (GLM-5-Team et al., 2026) are evaluated under the same ReAct protocol (Yao et al., 2022). On General-Agent, Gemma 4 31B (Gemma Team et al., 2026) is evaluated using OpenHands (Wang et al., 2026b).

Evaluation Protocol. We report two complementary views. **All-Accepted Evaluation** evaluates every accepted descendant available at each stage, so the underlying set of lineages may change when transformations fail. **Complete-Lineage Evaluation** instead restricts evaluation to seed lineages that survive all five transformations and compares their corresponding Stage 0–5 descendants, holding the underlying task population fixed and separating accumulated complexity growth from stage-wise lineage filtering.

5 Results

5.1 Agent Forest Expands Verified Agentic Data

Table 1 summarizes the descendants produced by Agent Forest across all three environments. Starting from 114 τ^2-Retail, 50 τ^2-Airline, and 48 General-Agent seed tasks, the framework generates 443, 169, and 211 validated descendants, respectively. This corresponds to $3.89\times$, $3.38\times$, and $4.40\times$ more verified tasks than contained in the original seed collections, yielding 823 accepted descendants from 212 seeds overall. These results demonstrate that controlled complexity growth can substantially expand trusted agentic datasets while preserving executability and semantic validity.

The verification pipeline discards transformations that do not remain executable or semantically consistent rather than propagating them to later stages. Nevertheless, substantial numbers of task lineages survive the full complexity-growth process: 53 tasks for τ^2-Retail, 28 for τ^2-Airline, and 39 for General-Agent remain after all five transformations. Acceptance behavior also varies across environments and transformation types. For example, Compositionality accounts for the largest number of discarded descendants on Retail, while Tool Expansion produces the largest initial reduction on Airline. In contrast, General-Agent maintains comparatively high acceptance rates throughout the process. These results

Table 1: **Pipeline yield and reference-solution statistics** across the three datasets under cumulative complexity growth from Agent Forest. **Qwen 3.5 397B** is used as the **ReAct** generation agent and **GPT-OSS 120B** as the **LLM-as-judge.** Statistics are computed from the verified reference solutions of accepted tasks.

Stage	Agent	Pipeline Yield			Reference Solution Statistics		
		Input	Accepted	Discarded	Avg. Tool Calls	Avg. Writes	Avg. Distinct Tools
				τ^2-Retail			
0	Seed	114	114	0	4.82	1.54	3.34
1	+Tool Expansion	114	**108**	6	8.61	1.55	5.69
2	+Branching	108	**101**	7	9.97	1.57	6.09
3	+Memory	101	**92**	9	11.49	1.64	6.23
4	+Partial Observability	92	**89**	3	11.54	1.65	6.22
5	+Compositionality	89	**53**	36	12.62	2.45	7.15
				τ^2-Airline			
0	Seed	50	50	0	2.84	0.98	1.70
1	+Tool Expansion	50	**38**	12	5.45	0.84	3.76
2	+Branching	38	**37**	1	7.08	0.97	5.00
3	+Memory	37	**33**	4	7.91	1.42	6.03
4	+Partial Observability	33	**33**	0	9.33	1.42	6.03
5	+Compositionality	33	**28**	5	10.64	2.11	6.89
				General-Agent			
0	Seed	48	48	0	31.10	21.41	7.51
1	+Tool Expansion	48	**46**	2	38.04	22.35	10.12
2	+Branching	46	**44**	2	42.00	23.37	10.35
3	+Memory	44	**41**	3	43.80	24.31	10.37
4	+Partial Observability	41	**41**	0	43.82	24.31	10.39
5	+Compositionality	41	**39**	2	48.35	25.67	11.29

show that dataset expansion is coupled with explicit verification and filtering rather than unconditional accumulation of generated tasks.

Beyond increasing dataset size, complexity growth systematically changes the interaction requirements of accepted descendants. On τ^2-Retail, average reference-solution tool calls increase from 4.82 for the seed tasks to 12.62 after all five transformations, while the number of distinct tools increases from 3.34 to 7.15. On τ^2-Airline, tool calls increase from 2.84 to 10.64 and distinct tools from 1.70 to 6.89. General-Agent begins with substantially longer interactions, yet complexity growth further increases average tool calls from 31.10 to 48.35, state-changing writes from 21.41 to 25.67, and distinct tools from 7.51 to 11.29.

The individual transformations leave distinct signatures on the resulting descendants. Tool Expansion produces the largest increase in tool breadth across all three environments, whereas later transformations increasingly modify dependency structure, information availability, and task composition. Importantly, increases in interaction complexity do not necessarily correspond to longer execution trajectories. For example, on General-Agent, Partial Observability changes average reference-solution tool calls only from 43.80 to 43.82 while leaving writes unchanged at 24.31, illustrating that a transformation can substantially alter the information required to solve a task without materially increasing the number of actions required. This highlights that complexity growth reflects changes in the structure of the interaction problem, not merely increases in trajectory length.

Overall, Agent Forest expands a small collection of trusted seed tasks into a substantially larger set of verified descendants while introducing multiple dimensions of environment-grounded interaction complexity. The reference-solution statistics show that descendants require broader tool use, richer environment interactions, and more complex dependency structures than their corresponding seeds. We next evaluate whether these increases in interaction complexity translate into increased difficulty for downstream solver agents.

Table 2: **Solver evaluation on τ^2-Retail. Qwen 3.5 397B** is evaluated with the **ReAct** agent harness across the cumulative complexity-growth stages. We report results across all accepted tasks available at each stage and on the fixed subset of 53 tasks that successfully pass through all five transformations. Correct/Total and Pass@1 report task-level success, while Avg. Tool Calls, Avg. Writes, and Avg. Distinct Tools are computed from the solver trajectories.

Stage	Agent	Correct/Total	Pass@1	Avg. Tool Calls	Avg. Writes	Avg. Distinct Tools
All-Accepted Evaluation						
0	Seed	86/114	75.4%	7.08	1.51	4.68
1	+Tool Expansion	76/108	70.4%	6.69	1.44	4.51
2	+Branching	72/101	71.3%	6.60	1.42	4.55
3	+Memory	62/92	67.4%	7.22	1.58	4.76
4	+Partial Observability	58/89	65.2%	7.30	1.61	4.76
5	+Compositionality	29/53	54.7%	8.15	2.36	5.51
Complete-Lineage Evaluation						
0	Seed	41/53	77.4%	6.45	1.23	4.49
1	+Tool Expansion	36/53	67.9%	6.06	1.26	4.28
2	+Branching	38/53	71.7%	6.00	1.23	4.28
3	+Memory	37/53	69.8%	6.96	1.38	4.75
4	+Partial Observability	36/53	67.9%	6.45	1.30	4.49
5	+Compositionality	29/53	54.7%	8.15	2.36	5.51

5.2 Complexity Growth Increases Solver Difficulty

5.2.1 τ^2-Retail

Table 2 shows that increasing interaction complexity substantially reduces solver success on τ^2-Retail. Across all accepted descendants, Qwen 3.5 397B drops from 75.4% Pass@1 on the seed tasks to 54.7% after all five complexity-growth transformations, an absolute decrease of 20.7 percentage points. At the same time, solver trajectories become increasingly involved: average tool calls increase from 7.08 to 8.15, state-changing writes from 1.51 to 2.36, and distinct tools from 4.68 to 5.51. These results indicate that complexity-grown descendants require broader tool use and greater coordination across environment interactions while producing substantially lower end-to-end success rates.

The Complete-Lineage Evaluation shows the same pattern on the fixed set of 53 task lineages that survive all five transformations. Pass@1 decreases from 77.4% on the corresponding seed tasks to 54.7% after full complexity growth, a 22.7 percentage-point reduction. Over the same comparison, average tool calls increase from 6.45 to 8.15, writes from 1.23 to 2.36, and distinct tools from 4.49 to 5.51. This confirms that the performance degradation persists when the underlying task population is held fixed, providing direct evidence that accumulated complexity growth, rather than lineage filtering, increases task difficulty.

5.2.2 τ^2-Airline

Table 3 shows that accumulated complexity growth substantially increases solver difficulty on τ^2-Airline. Across all accepted descendants, Gemma 4 31B drops from 66.0% Pass@1 on the seed tasks to 28.6% after all five transformations, while GLM-5.2 drops from 72.0% to 53.6%. The degradation is observed across both solver models, indicating that the generated complexity is not specific to a particular model architecture, although Gemma is affected more strongly.

As interaction complexity increases, both models are forced into richer interactions with the environment. Gemma increases from 6.64 to 8.89 average tool calls and from 3.96 to 5.18 distinct tools, while GLM-5.2 increases from 7.44 to 10.46 tool calls and from 4.26 to 5.68 distinct tools. Thus, accumulated complexity growth increases both the amount and breadth of interaction required from the solver while simultaneously reducing successful task completion, suggesting that the generated descendants demand more coordinated multi-step reasoning and tool use rather than simply more actions.

The Complete-Lineage Evaluation shows the same trend on a fixed set of 28 task lineages. Gemma decreases from 75.0% to 28.6% Pass@1, while GLM-5.2 decreases from 78.6% to 53.6%. This confirms that the performance degradation persists when the underlying task population is held fixed, providing further evidence that accumulated interaction complexity, rather than changes in task composition or lineage filtering, increases solver difficulty.

Table 3: **Solver evaluation on τ^2-Airline. Gemma 4 31B** and **GLM-5.2** are evaluated as **ReAct** solver agents across successive complexity-growth stages. We report results on all accepted descendants available at each stage and on the fixed subset of 28 task lineages that successfully survive all five transformations. Correct/Total and Pass@1 measure task-level success, while Avg. Tool Calls, Avg. Writes, and Avg. Distinct Tools are computed from solver trajectories.

		Gemma 4 31B					GLM-5.2				
Stage	**Agent**	**Correct/ Total**	**Pass@1**	**Avg. Tool Calls**	**Avg. Writes**	**Avg. Distinct Tools**	**Correct/ Total**	**Pass@1**	**Avg. Tool Calls**	**Avg. Writes**	**Avg. Distinct Tools**
All-Accepted Evaluation											
0	Seed	33/50	66.0%	6.64	0.70	3.96	36/50	72.0%	7.44	0.98	4.26
1	+Tool Expansion	23/38	60.5%	7.18	0.66	4.05	26/38	68.4%	9.21	1.03	4.76
2	+Branching	24/37	64.9%	6.68	0.73	3.97	22/37	59.5%	6.73	0.73	4.00
3	+Memory	18/33	54.5%	7.45	1.18	5.03	19/33	57.6%	6.79	1.18	4.58
4	+Partial Observability	19/33	57.6%	9.88	1.06	4.79	17/33	51.5%	8.85	1.18	4.64
5	+Compositionality	8/28	28.6%	8.89	1.43	5.18	15/28	53.6%	10.46	1.93	5.68
Complete-Lineage Evaluation											
0	Seed	21/28	75.0%	5.43	0.54	3.75	22/28	78.6%	5.68	0.82	4.00
1	+Tool Expansion	18/28	64.3%	6.36	0.68	3.96	21/28	75.0%	7.36	0.93	4.75
2	+Branching	20/28	71.4%	5.57	0.79	3.75	19/28	67.9%	6.32	0.82	4.21
3	+Memory	15/28	53.6%	7.18	1.14	4.89	16/28	57.1%	6.79	1.25	4.71
4	+Partial Observability	16/28	57.1%	9.96	1.07	4.82	14/28	50.0%	8.50	1.18	4.61
5	+Compositionality	8/28	28.6%	8.89	1.43	5.18	15/28	53.6%	10.46	1.93	5.68

Table 4: **Solver evaluation on General-Agent. Gemma 4 31B** is evaluated with the **OpenHands** agent harness across successive complexity-growth stages. We report results on all accepted descendants available at each stage and on the fixed subset of 39 task lineages that successfully survive all five transformations. Correct/Total and Pass@1 measure task-level success, while Avg. Tool Calls, Avg. Writes, and Avg. Distinct Tools are computed from solver trajectories.

Stage	**Agent**	**Correct/Total**	**Pass@1**	**Avg. Tool Calls**	**Avg. Writes**	**Avg. Distinct Tools**
All-Accepted Evaluation						
0	Seed	33/48	68.8%	40.19	8.56	7.79
1	+Tool Expansion	20/46	43.5%	41.43	7.91	9.35
2	+Branching	21/44	47.7%	41.23	7.64	9.45
3	+Memory	18/41	43.9%	44.34	9.56	9.83
4	+Partial Observability	10/41	24.4%	46.73	9.02	9.54
5	+Compositionality	10/39	25.6%	40.41	7.72	9.62
Complete-Lineage Evaluation						
0	Seed	28/39	71.8%	36.56	7.03	7.74
1	+Tool Expansion	17/39	43.6%	41.79	7.77	9.15
2	+Branching	18/39	46.2%	41.92	7.74	9.23
3	+Memory	17/39	43.6%	44.77	9.44	9.69
4	+Partial Observability	9/39	23.1%	46.87	8.62	9.33
5	+Compositionality	10/39	25.6%	40.41	7.72	9.62

5.2.3 GENERAL-AGENT

Table 4 shows that accumulated complexity growth substantially reduces solver success on General-Agent, despite the seed tasks already requiring long, tool-intensive interactions. Across all accepted descendants, Gemma 4 31B drops from 68.8% Pass@1 on the seed tasks to 25.6% after all five transformations, an absolute decrease of 43.2 percentage points. At the same time, solver trajectories exhibit broader environment interaction, with average distinct tool usage increasing from 7.79 to 9.62. The large performance drop despite relatively modest changes in tool breadth suggests that later transformations increase difficulty through richer dependencies, information requirements, and coordination demands rather than simply requiring more tool use.

The intermediate stages further show that increasing interaction complexity leads to more involved environment interaction without preserving success. Under Partial Observability, for example, Gemma reaches 46.73 average tool

calls while Pass@1 falls to 24.4%, indicating that additional interaction alone is insufficient when critical information must be recovered, retained, and coordinated correctly throughout the task.

The Complete-Lineage Evaluation shows the same trend on the fixed set of 39 task lineages. Pass@1 decreases from 71.8% on the corresponding seed tasks to 25.6% after all five transformations, a 46.2 percentage-point reduction. This confirms that the observed degradation persists when the underlying task population is held fixed, providing strong evidence that accumulated interaction complexity, rather than changes in task composition, increases solver difficulty. More broadly, the results demonstrate that Agent Forest can substantially increase difficulty even in environments whose original tasks already require extensive planning, tool use, and state management.

5.3 Complexity-Growth Agents Expose Distinct Failure Modes

The five complexity-growth agents target different dimensions of interaction complexity and expose distinct weaknesses in solver behavior.

Tool Expansion stresses evidence integration across a broader tool space. Tool Expansion increases the number of distinct tools required from 3.34 to 5.69 on τ^2-Retail, 1.70 to 3.76 on τ^2-Airline, and 7.51 to 10.12 on General-Agent (Table 1). Solvers use more tools but do not always integrate the resulting evidence successfully. On General-Agent, Gemma's distinct tool usage increases from 7.79 to 9.35 while Pass@1 falls from 68.8% to 43.5% (Table 4), suggesting that broader tool use also requires effective evidence integration.

Branching Dependencies stress state-conditioned decision making. Later actions must depend on earlier observations. In failed trajectories, models may retrieve the correct branch condition but follow an earlier plan, choose the wrong alternative, or associate the observation with the wrong entity, indicating difficulty binding observed state to subsequent decisions.

Memory stresses long-range information binding. Memory separates when information is acquired from when it must later be used. Models may retrieve the correct value early but lose or misassociate it after intervening actions, producing incorrect identifiers, constraints, or state updates. On τ^2-Airline, Gemma and GLM-5.2 achieve 54.5% and 57.6% Pass@1 after Memory despite continued tool interaction (Table 3).

Partial Observability stresses information recovery rather than trajectory length. On General-Agent, average reference-solution tool calls change only from 43.80 to 43.82, writes remain at 24.31, and distinct tools change from 10.37 to 10.39 (Table 1). Yet Gemma Pass@1 drops from 43.9% to 24.4%, and from 43.6% to 23.1% on the fixed set of 39 task lineages (Table 4). Thus, information recovery can substantially increase difficulty without materially increasing trajectory length.

Compositionality stresses global coordination. Compositionality combines multiple requirements into a single workflow, where solvers may omit downstream requirements or leave state changes inconsistent. After full cumulative hardening, Pass@1 reaches 54.7% on τ^2-Retail, 28.6% and 53.6% for Gemma and GLM-5.2 on τ^2-Airline, and 25.6% on General-Agent (Tables 2, 3, and 4).

Overall, different complexity dimensions expose distinct capability limitations. Across transformations, agents continue to interact extensively with their environments while fully hardened tasks show substantially lower success than the original seeds. The bottleneck shifts across evidence integration, state-conditioned decisions, long-range information use, information recovery, and multi-objective coordination, showing that interaction complexity is multidimensional and that different transformations expose complementary weaknesses in modern tool-using agents.

6 Conclusion

We introduced Agent Forest, a framework for expanding trusted agentic datasets through verified, environment-grounded complexity growth. Across τ^2-Retail, τ^2-Airline, and General-Agent, Agent Forest produces larger collections of validated descendants that increase multiple dimensions of interaction complexity and reduce Pass@1 by 18.4–43.2 percentage points across strong solver agents. These tasks expose recurring weaknesses in evidence integration, state-conditioned decision making, long-range information use, information recovery, and global coordination. Overall, Agent Forest shows that trusted seed datasets can be systematically transformed into diverse, executable task families for both stronger agent evaluation and richer agent training.

AI USE STATEMENT

Large language models were used as backbone models within the Agent Forest framework for dataset generation, including task transformation and semantic validation, as described in Sections 3 and 4. Generative AI tools were also used to assist with code development and debugging. During manuscript preparation, LLMs were used for writing assistance, primarily for sentence-level refinement and grammatical polishing. All AI-generated suggestions were carefully reviewed by the authors, who take full responsibility for the final content of this work.

ETHICS STATEMENT

This work studies automated task generation and agent evaluation in benchmark environments and does not involve human subjects or the collection of personal user data. Agent Forest generates verified task descendants through controlled complexity growth while using execution replay and semantic validation to preserve executability and consistency with the underlying environments, as described in Section 3. As with other methods for synthetic agentic data generation, the resulting descendants may inherit limitations, biases, or coverage gaps present in the original environments and seed tasks. Our experimental setup, generation process, and evaluation protocols are described in Section 4.

REPRODUCIBILITY STATEMENT

We describe the Agent Forest framework, complexity-growth agents, environment-grounded ReAct procedure, verification and repair pipeline, and complexity-growth paths in Section 3. Section 4 details the datasets, generation models, evaluation models, transformation sequence, and evaluation protocols used in our experiments. Algorithm and prompts used for the Agent Forest framework are provided in Appendix Section A and B. Code and generated task artifacts required to reproduce the complexity-growth and evaluation pipeline will be publicly released upon acceptance.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Shihao Cai, Runnan Fang, Jialong Wu, Baixuan Li, Xinyu Wang, Yong Jiang, Liangcai Su, Liwen Zhang, Wenbiao Yin, Zhen Zhang, Fuli Feng, Pengjun Xie, and Xiaobin Wang. Autoforge: Automated environment synthesis for agentic reinforcement learning, 2025. URL https://arxiv.org/abs/2512.22857.

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.

Jiaxuan Gao, Jiaao Chen, Chuyi He, Shusheng Xu, Di Jin, and Yi Wu. From self-evolving synthetic data to verifiable-reward rl: Post-training multi-turn interactive tool-using agents, 2026. URL https://arxiv.org/abs/2601.22607.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, Mayank Chaturvedi, Aditya Chawla, Victor Cotruta, Alice Coucke, Phil Culliton, Robert Dadashi, Lucas Dixon, Mohamed Elhawaty, Utku Evci, Clément Farabet, Johan Ferret, Filippo Galgani, Sertan Girgin, Jean-Bastien Grill, Maarten Grootendorst, Jiaxian Guo, Cassidy Hardin, Yanzhang He, Steven M. Hernandez, Omri Homburger, Léonard Hussenot, Juyeong Ji, Armand Joulin, Aishwarya Kamath, Parnian Kassraie, Olivier Lacombe, Preethi Lahoti, Gaël Liu, Gus Martins, Luciano Martins, Tatiana Matejovicova, Ramona Merhej, Nikola Momchev, Sneha Mondal, Ryan Mullins, Sindhu Raghuram Panyam, Shreya Pathak, Sarah Perrin, André Susano Pinto, Etienne Pot, Angéline Pouget, Alexandre Ramé, Sabela Ramos, Douglas Reid, David Rim, Morgane Rivière, Karsten Roth, Louis Rouillard, Omar Sanseviero, Pier Giuseppe Sessa, Shane Settle, Danila Sinopalnikov, Sara Smoot, Piotr Stanczyk, Andreas Steiner, Lawrence Stewart, Ilya Tolstikhin, Michael Tschannen, Anton Tsitsulin, Nino Vieillard, Renjie Wu, Pingmei Xu, Haichuan Yang, Edouard Yvinec, Biao Zhang,

Li Zhang, Joe Zou, Nicolas Aagnes, Abdelrahman Abdelhamed, Jakub Adamek, Shivani Agrawal, Shubham Agrawal, Ibrahim Alabdulmohsin, Jean Baptiste Alayrac, Uri Alon, Chandramouli Amarnath, Ankesh Anand, Chrysovalantis Anastasiou, Setareh Ariafar, François-Xavier Aubet, Kyriakos Axiotis, Federico Barbero, Joelle Barral, Alexei Bendebury, Urs Bergmann, Stanley Bileschi, Kat Black, Mathieu Blondel, Sebastian Borgeaud, Arthur Bražinskas, Ryan Burnell, Robert Busa-Fekete, Mu Cai, Daniele Calandriello, Glenn Cameron, Charlotte Caucheteux, Rahma Chaabouni, Garima Chadha, Jetha Chan, Blake Jianhang Chen, Jesse Chen, Lin Chen, Xu Chen, Derek Cheng, Tzu hsiang Chien, Nikolai Chinaev, Yi Chou, Zhaohui Chu, Benjamin Coleman, Pooja Consul, Sam Conway-Rahman, Scott Crowell, Dylan Cutler, Vivek Dani, Samira Daruki, Anil Das, Daniel Deutsch, Nishanth Dikkala, Li Ding, Qiuhan Ding, Shenil Dodhia, Konstantin Donhauser, Tulsee Doshi, Anca Dragan, Alex Druinsky, Sahil Dua, Zoltan Egyed, Danielle Eisenbud, Daniel Eppens, Cindy Fan, Bahare Fatemi, Yassir Fathullah, Vlad Feinberg, Milen Ferev, Sebastian Flennerhag, Takumi Fujimoto, João Gabriel Oliveira, Isaac Galatzer-Levy, João Gante, Simon Geisler, Soham Ghosal, Antonious M. Girgis, Tamara von Glehn, Alec Go, Alhaad Gokhale, Alex Grills, Yiming Gu, Mayank Gupta, Pramod Gupta, Guru Guruganesh, Raia Hadsell, Hamza Harkous, Jitendra Harlalka, Demis Hassabis, Anja Hauth, Joe Heyward, Arian Hosseini, Chih-Yang Hsia, I-Hung Hsu, Xiaopeng Huang, Yangsibo Huang, Kevin Hui, Adrian Hutter, Te I, Fotis Iliopoulos, Advait Jain, Ganesh Jawahar, Ziwei Ji, Qilin Jin, Melvin Johnson, Kandarp Joshi, Arun Kandoor, Wang-Cheng Kang, Koray Kavukcuoglu, Mehran Kazemi, Kathleen Kenealy, Amr Khalifa, Phoebe Kirk, Ivan Korotkov, Suraj Kothawade, Vitaly Kovalev, Neel Kovelamudi, Adam Kraft, Ravin Kumar, Vivek Kumar, Harish Kuppam, Justin Lannin, Chen-Yu Lee, Seungji Lee, Dmitry Lepikhin, Alon Levkovitch, Dongdong Li, Qiujia Li, Valentin Liévin, Ethan Lin, Ziqian Lin, Casper Liu, Tianlin Liu, Tianqi Liu, Xin Liu, Ivan Lobov, Mayank Lunayach, Min Ma, Gagan Madan, Andrii Maksai, Eric Malmi, Michal Matuszak, Daniel McDuff, Gaurav Menghani, Maciej Mikuła, Daniil Mirylenka, Karolis Misiunas, Vedant Misra, Andreea Mitran, Kareem Mohamed, Maksim Mukha, Eric Noland, James O'Donnell, Brendan O'Donoghue, Kate Olszewska, Bernett Orlando, Wanqiong Pan, Rina Panigrahy, Unnati Parekh, Nicolas Perez-Nieves, Chunjong Park, Eric Paskie, Liqian Peng, Bryce Petrini, Slav Petrov, Jonas Pfeiffer, Bilal Piot, Martyna Plomecka, Siim Poder, Octavio Ponce, Arijit Pramanik, David Racz, Anish Rajan, Michelle Ramanovich, Anand Rao, Marvin Ritter, Vitor Rodrigues, Evan Rosen, Mikołaj Rybiński, Noveen Sachdeva, Michaël E. Sander, Rohit Sathyanarayana, Sagar Savla, Samuel Schmidgall, Tal Schuster, George Scrivener, Benoit Seguin, Andrew Sellergren, Aliaksei Severyn, Izhak Shafran, Dhruv Shah, Bobak Shahriari, Yuan Shangguan, Ashish Shenoy, Pradeep Shenoy, Rakesh Shivanna, Pauline Sho, Lucas Spangher, Wojciech Stokowiec, Tim Strother, Yao Su, Yinghao Sun, Mukund Sundararajan, Andrea Tacchetti, Mor Hazan Taege, Pouya Tafti, Jean Tarbouriech, Chetan Tekur, Shantanu Thakoor, Rahul Thapa, Madeleine Traverse, Lenart Treven, Tao Tu, Chien Te Tung, Çağlar Ünlü, Petar Veličković, Malini Pooni Venkat, Sagar Gubbi Venkatesh, Vidya Venkiteswaran, Francesco Visin, Alex Vitvitskyi, Kiran Vodrahalli, Weiyi Wang, Xin Wang, Tris Warkentin, Jan Wassenberg, John Wieting, Cindy Wu, Lechao Xiao, Hao Xu, Yuhui Xu, Fuzhao Xue, Arun Yadav, Jun Yan, Antoine Yang, Lin Yang, Ming-Hsuan Yang, Ziyu Ying, Jae Hyeon Yoo, Morteza Zadimoghaddam, Sajjad Zafar, Fred Zhang, Jiageng Zhang, Jianyi Zhang, Xiaofan Zhang, Chao Zhao, David Zhou, and Chen Zou. Gemma 4 technical report, 2026. URL https://arxiv.org/abs/2607.02770.

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.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. Toolace: Winning the points of llm function calling, 2025a. URL https://arxiv.org/abs/2409.00920.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents, 2025b. URL https://arxiv.org/abs/2308.03688.

Yuanjie Lyu, Chengyu Wang, Haonan Zheng, Yuanhao Yue, Junbing Yan, Ming Wang, and Jun Huang. Agenticqwen: Training small agentic language models with dual data flywheels for industrial-scale tool use, 2026. URL https://arxiv.org/abs/2604.21590.

Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei ge Chen, Olga Vrousgos, Corby Rosset, Fillipe Silva, Hamed Khanpour, Yash Lara, and Ahmed Awadallah. Agentinstruct: Toward generative teaching with agentic flows, 2024. URL https://arxiv.org/abs/2407.03502.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, Tianjie Zhang, Wei Xu, Jie Tang, and Yuxiao Dong. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning, 2025. URL https://arxiv.org/abs/2411.02337.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. URL https://arxiv.org/abs/2307.16789.

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

Mika Senghaas. General agent: A self-evolving, synthetic agent environment. *Prime Intellect Blog*, May 2026. https://www.primeintellect.ai/blog/general-agent.

Saptarshi Sengupta, Harsh Vashistha, Kristal Curtis, Akshay Mallipeddi, Abhinav Mathur, Joseph Ross, and Liang Gou. Mag-v: A multi-agent framework for synthetic data generation and verification, 2025. URL https://arxiv.org/abs/2412.04494.

Dingfeng Shi, Jingyi Cao, Qianben Chen, Weichen Sun, Weizhen Li, Hongxuan Lu, Fangchen Dong, Tianrui Qin, King Zhu, Minghao Liu, Jian Yang, Ge Zhang, Jiaheng Liu, Changwang Zhang, Jun Wang, Yuchen Eleanor Jiang, and Wangchunshu Zhou. Taskcraft: Automated generation of agentic tasks, 2025. URL https://arxiv.org/abs/2506.10055.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning, 2021. URL https://arxiv.org/abs/2010.03768.

Dong Wang, Yang Li, Ansong Ni, Ching-Feng Yeh, Youssef Emad, Xinjie Lei, Liam Robbins, Karthik Padthe, Hu Xu, Xian Li, Asli Celikyilmaz, Ramya Raghavendra, Lifei Huang, Carole-Jean Wu, and Shang-Wen Li. Matrix: Peer-to-peer multi-agent synthetic data generation framework, 2026a. URL https://arxiv.org/abs/2511.21686.

Xingyao Wang, Simon Rosenberg, Juan Michelini, Calvin Smith, Hoang H Tran, Engel Nyst, Rohit Malhotra, Xuhui Zhou, Valerie Chen, Robert Brennan, et al. The openhands software agent sdk: A composable and extensible foundation for production agents. *Proceedings of Machine Learning and Systems*, 8:1843–1855, 2026b.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions, 2023. URL https://arxiv.org/abs/2212.10560.

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, 2026c. URL https://arxiv.org/abs/2602.10090.

Jingxu Xie, Dylan Xu, Xuandong Zhao, and Dawn Song. Agentsynth: Scalable task generation for generalist computer-use agents, 2026. URL https://arxiv.org/abs/2506.14205.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. Wizardlm: Empowering large pre-trained language models to follow complex instructions, 2025. URL https://arxiv.org/abs/2304.12244.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents, 2023. URL https://arxiv.org/abs/2207.01206.

A Agent Forest Hardening Algorithm

Algorithm 1 provides the complete procedure used to transform verified seed tasks. Each accepted candidate becomes the parent of the next selected hardening operation, while candidates that remain invalid after repair terminate that transformation path.

Algorithm 1 Agent Forest complexity-growth procedure

Require: Seed tasks $\mathcal{D}$, environment $\mathcal{E}$, tools $\mathcal{T}$, ordered complexity-growth agents $\mathcal{A}$, repair budget B

$\mathcal{D}_{\text{out}} \leftarrow \emptyset$
for each seed task $x \in \mathcal{D}$ **do**
  Verify and ground x in $\mathcal{E}$
  $x_{\text{parent}} \leftarrow x$
  for each selected agent $a \in \mathcal{A}$ **do**
    $x' \leftarrow h_a(x_{\text{parent}}, \mathcal{E}, \mathcal{T})$
    $(v, f) \leftarrow \textsc{Validate}(x', \mathcal{E})$
    $b \leftarrow 0$
    while $v = \textsc{False}$ and $b < B$ **do**
      $x' \leftarrow \textsc{Repair}(x', f)$
      $(v, f) \leftarrow \textsc{Validate}(x', \mathcal{E})$
      $b \leftarrow b + 1$
    end while
    if $v = \textsc{True}$ **then**
      Add x' to $\mathcal{D}_{\text{out}}$
      $x_{\text{parent}} \leftarrow x'$
    else
      break {Terminate the current transformation path}
    end if
  end for
end for
return $\mathcal{D}_{\text{out}}$

B HARDENING AGENT INSTRUCTIONS

We provide the task-hardening instructions used by each of the five specialized agents below.

Tool Expansion Agent

Make the smallest coherent extension that adds one or more useful task-local tool calls. Prefer two or three tools only when they fit naturally, and optionally add a small connected secondary subtask when the parent task supports one. One useful added tool is fully acceptable. Read-only expansion is acceptable. Never invent a deposit, withdrawal, freeze, transfer, cancellation, bonus, or any other write merely to increase tool count or difficulty. Add a write only when the revised user request genuinely requires it.

Branching Agent

Modify the task and its solution so that a real observation induces a meaningful conditional subtask. The observation should change what the agent does next: the selected action, target, arguments, amount, fallback, or deliberate no-op. Prefer a branch that eventually affects a state-changing action or final task outcome when the domain naturally supports one, but do not invent an unjustified write. One well-grounded branch is sufficient; richer multi-condition branching is encouraged only when it remains coherent and executable.

Memory Agent

Modify the task and solution so the agent must retain several task-relevant facts retrieved early, retained across meaningful intervening work, and then visibly reused later. The retained facts should affect a later tool argument, branch choice, write-oriented action, write target, write amount, eligibility decision, verification, or final database state. Prefer multiple facts used together and a later write-oriented effect when naturally supported. Do not create memory difficulty only by adding 'remember this' wording while leaving the solution behavior unchanged.

Partial Observability Agent

Use a strict RECOVER-BEFORE-HIDE workflow. Hide or omit a concrete value only after you have identified the exact task-local recovery tool, actually called that tool in the current generation session, and observed the exact value in a specific response field. The value must be environment state that can be reconstructed from information that remains visible plus tool outputs. Never hide a policy constant, decision threshold, target category/region, requested amount, requested term, destination, exception rule, or other user-defined criterion unless an available tool explicitly returns that exact criterion. Do not replace a removed value with an undefined placeholder such as 'target region A', 'required threshold', or 'designated category'. If the exact value cannot be recovered, do not hide it; choose another concrete environment-backed value. The recovered value must then be used in a later action, branch, argument, write, or verification. Partial observability means hiding recoverable environment state. Do not delete information that defines what the user is asking the agent to do.

Compositionality Agent

Add multiple connected subtasks that create a genuinely longer-horizon dependency chain. Earlier observations or intermediate results must influence later tool calls, branch choices, writes, or verification. Prefer a coherent retrieval $\rightarrow$ retention/decision $\rightarrow$ operation $\rightarrow$ post-operation verification chain over shared entities. Do not count duplicate reads, unused outputs, cosmetic trajectory padding, or final-report-only wording as meaningful composition.