DYNAWEAVE: LEARNING TO ADAPT THROUGH DYNAMIC SCENARIO SYNTHESIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents increasingly use tools to complete tasks in stateful environments. However, many training environments remain largely static apart from changes caused by the agent's own actions, whereas external events introduced by other users, services, or sudden contingencies in real environments can invalidate ongoing plans. To address this gap, we introduce DynaWeave, an automated synthesis framework that transforms existing static tasks into verifiable dynamic scenarios. The framework generates exogenous events relevant to each task, covering changes in task requirements, environment state, and external information. It uses directed acyclic graphs (DAGs) to represent dependencies between events and an execution engine to activate them during agent interaction. A dynamic verifier revises only the evaluation checks affected by events that actually occur and retains all other checks. Using DynaWeave, we collect 2,095 verified interaction trajectories across 36 environments and use them for supervised fine-tuning of Qwen3.5 models. Experiments on Gaia2, CostBench, and ToolMaze show that DynaWeave achieves consistent improvements in overall performance across all three evaluated model sizes, increasing the task completion rate on CostBench by 67.34 percentage points and even exceeding the performance of Gemini-3.1-Pro-Preview on ToolMaze. These results support dynamic scenario synthesis as a practical approach to training agents to adapt to changing environments. Our code is available at `https://anonymous.4open.science/r/Anonymous-DynaWeave-81B9`.

1 INTRODUCTION

Large language models (LLM) are increasingly capable of using tools to complete tasks in external environments beyond the conversation. For example, an agent may search a database, compare available options, and then book a room or issue a refund through a tool. Existing benchmarks provide executable environments such as websites, applications, and tool sandboxes in which agents perform tasks, and automatically assess whether the results satisfy the user's request. (Zhou et al., 2024; Xie et al., 2024; Trivedi et al., 2024; Lu et al., 2025; Yao et al., 2024; Barres et al., 2025). Recent advances in environment synthesis further automate the construction of applications, tasks, and verifiers, expanding the supply of verifiable interactions for agent training and evaluation (Dong et al., 2026; Song et al., 2026; Wang et al., 2026; Froger et al., 2025).

Despite this recent progress, most existing environments rely on an implicit assumption: once a task begins, the environment remains largely unchanged except through the actions of the evaluated agent. In practice, however, real-world environments offer no such guarantee of stability: another user may claim a resource that the agent had previously found available; a service may suddenly become unavailable; or an unexpected contingency may disrupt the original plan. We refer to such changes that affect a task during its execution but are not caused by the evaluated agent as *exogenous events*. These changes can easily invalidate an otherwise correct plan, making success contingent on the agent's ability to dynamically revise its decisions rather than merely execute them.

Recent benchmarks have demonstrated the importance of evaluating agents under dynamic scenarios. Unlike previous static benchmarks, a dynamic scenario is characterized by the possibility of exogenous events during task execution. Gaia2 (Froger et al., 2026) provides asynchronous environments that evolve independently of the agent; CostBench (Liu et al., 2026) changes tool costs and

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.

availability during execution, requiring agents to revise their plans; ToolMaze (Zhu et al., 2026) introduces explicit or implicit tool failures that require recovery and replanning; and ClawMark (Meng et al., 2026) incorporates external updates into multi-turn, multi-day workflows. TThese benchmarks highlight the value of evaluating adaptation, but constructing dynamic scenarios and their corresponding verifiers can require substantial manual effort. Producing such scenarios at the scale needed for agent training therefore remains a challenge.

In this work, we propose DynaWeave, a framework that automatically transforms existing static tasks into verifiable dynamic scenarios. Given an executable environment, a task, and its original verifier, DynaWeave identifies possible changes relevant to the task and generates corresponding exogenous events. It filters out events that are invalid, implausible, or have no meaningful effect on the task. The resulting scenarios are configured before evaluation, and events are activated during agent interaction when their conditions are met. The original verifier is adapted to account for events that actually occur while preserving unaffected task requirements. This design combines automatic event generation, reproducible execution, and verification, allowing existing task collections such as EnvScaler (Song et al., 2026) and Agent-World (Dong et al., 2026) to be extended without manually designing dynamic scenarios for each domain.

To evaluate the value of these scenarios for agent training, we use DynaWeave to collect 2,095 verified interaction trajectories and subsequently perform supervised fine-tuning on Qwen3.5 models with 2B, 4B, and 9B parameters. Across all three model sizes, this training consistently improves the overall scores on Gaia2 and ToolMaze, as well as the task completion rate on CostBench. In particular, the Qwen3.5-2B model improves its task completion rate on CostBench from 3.00% to 70.34%, representing a substantial gain of 67.34 percentage points. Furthermore, on ToolMaze, the 9B model achieves a composite score of 32.28 after training, even exceeding the baseline performance of Gemini-3.1-Pro-Preview (32.19). These results strongly indicate that the benefits of training on the generated trajectories generalize well and extend beyond the training environments, effectively improving the models' adaptability to unexpected changes during task execution.

The main contributions of our work are as follows: (1) We propose DynaWeave, a general framework for automatically transforming existing static, executable tool-use tasks into verifiable dynamic variants with exogenous changes occurring during task execution.(2) We construct 2,095 dynamic scenarios across 36 environments from EnvScaler, τ^2-bench, ToolSandbox, and AgentDojo, and collect verified interaction trajectories for supervised fine-tuning. (3) Experiments on three dynamic benchmarks show consistent improvements across three model sizes, demonstrating the value of DynaWeave for training agents to adapt to changes in dynamic environments.

2 Related Work

2.1 Executable benchmarks and environment synthesis

Executable benchmarks evaluate agents through continuous interactions with working applications and rigorous checks of task completion. WebArena (Zhou et al., 2024) and OSWorld (Xie et al., 2024) support reproducible interaction with complex web and computer systems; AppWorld (Trivedi et al., 2024) and ToolSandbox (Lu et al., 2025) evaluate intermediate environment states or complete interaction trajectories; and τ-bench (Yao et al., 2024) incorporates simulated users and specific domain policies. AndroidWorld (Rawles et al., 2025) expands overall task coverage through parameterized initialization, which effectively varies task instances rather than necessarily introducing unpredictable external changes during execution. τ^2-bench (Barres et al., 2025) permits both the user and the agent to modify a shared environment, emphasizing real-time coordination between participants. Ultimately, these existing settings provide valuable, reusable tools and evaluation procedures on which more dynamic extensions can readily build.

Environment synthesis addresses the cost of constructing such resources. AgentScaler (Fang et al., 2025) and AutoForge (Cai et al., 2025) automate environment construction for agent training. EnvScaler (Song et al., 2026) separates executable environment skeletons from task scenarios and rule-based validators, while Agent World Model (AWM) (Wang et al., 2026) constructs database-backed environments with accessible state for reward computation. Agent-World (Dong et al., 2026) further connects environment and task synthesis to capability gaps identified during training. Their primary focus is expanding the supply and diversity of executable tasks. DynaWeave addresses a comple-

mentary dimension: generating changes within an existing task, determining when and how they become observable, and adapting evaluation to their effects. It reuses the source environment and tool interface, preserving unaffected requirements rather than constructing every dynamic scenario from scratch. This executable foundation also allows event application to be checked separately from agent behavior, so errors in the generated scenario need not be treated as failures of adaptation.

2.2 Dynamic and asynchronous agent benchmarks

Dynamic benchmarks examine how agents respond when earlier information or plans become invalid. Gaia2 (Froger et al., 2026) introduces independently evolving, asynchronous environments, supported by ARE's (Froger et al., 2025) for applications, event execution, and verification. ClawMark (Meng et al., 2026) incorporates external updates into workflows spanning multiple turns and days. CostBench (Liu et al., 2026) studies economical replanning under changes such as tool failures and revised costs, whereas ToolMaze (Zhu et al., 2026) combines tool dependency structures with explicit or implicit failures that may be transient or permanent. ClawArena (Ji et al., 2026) focuses on revising beliefs as evidence changes, and SimuHome (Seo et al., 2026) models temporal changes in a smart-home simulator. Together, these benchmarks expose distinct challenges in adaptation, from recognizing unreliable information to recovering from disrupted execution.

Our focus is complementary to constructing these evaluation settings: automatically deriving plausible events and corresponding verification rules from existing static tasks across domains. An execution engine enables scheduled changes, but event selection and evaluation must still account for the source task's requirements. ProEvolve (Li et al., 2026) and EvoArena (Xu et al., 2026) address change across environment versions and successive tasks, including evolving tools, schemas, data, and agent memory. In contrast, DynaWeave retains the tool interface and environment rules while introducing changes during task execution. This scope supports paired evaluation of static and dynamic variants and the collection of demonstrations of adaptation within the same underlying environments.

3 DynaWeave

DynaWeave transforms existing static tasks into verifiable dynamic scenarios by reusing their executable environments and adapting the original success criteria. Given an environment, its initial state, a task instruction, and the original verifier, it constructs a dynamic scenario Π and a corresponding dynamic verifier V_{Π}. As illustrated in Figure 1, the framework consists of three components. First, *exogenous event construction* identifies plausible event themes relevant to the task, specifies each event's effects, activation conditions, and observability, and organizes compatible events into a directed acyclic graph (DAG). Second, *controlled dynamic execution* activates events based on action completion, virtual time, or random triggers, while respecting event dependencies and environment conditions. Third, *dynamic verifier adaptation* revises checks affected by events that actually occur and preserves requirements that remain applicable. When correctness depends on the timing or order of actions, the verifier also checks the execution trace.

3.1 Exogenous Event Construction

To construct plausible exogenous events for static tasks, we design a three-stage pipeline that first identifies event themes, then generates their specific content, and finally organizes the resulting events into a directed acyclic graph.

Identifying Sources of Change. The first stage identifies where relevant dynamics may arise during task execution. To guide the automatic generation of plausible event themes, we distinguish three sources of dynamics: (i) task requirements, where the user revises the original goal; (ii) environment state, where resource availability or tool functionality changes; and (iii) external information, where a contingency affect the goal or path of the task. For example, a typhoon warning at the intended destination may require the agent to reconsider a flight booking. Given the environment information, initial state, and task description, DynaWeave determines which sources can plausibly affect the task and proposes corresponding event themes. Each theme provides a concise description of a potential change, such as the user cancelling a trip, fewer rooms being available, or a typhoon warning being issued. These themes guide the generation of concrete events in the next stage.

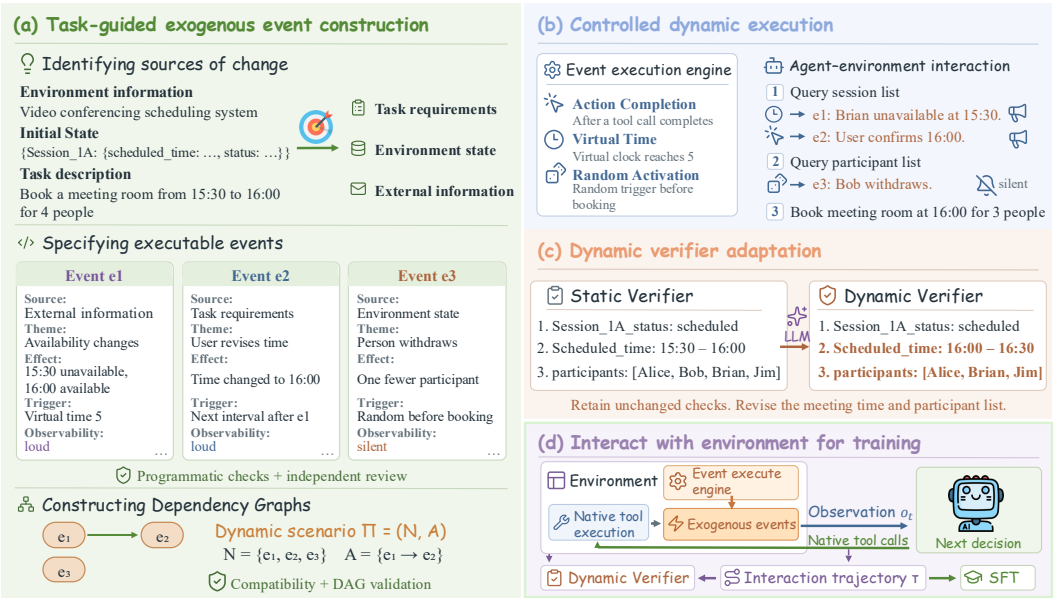


Figure 1: Overview of DynaWeave. (a) The framework generates task-grounded exogenous events from an existing static task, validates their specifications, and organizes compatible events into a dependency graph. (b) An execution engine activates events during agent interaction through action completion, virtual time, or random activation. (c) The dynamic verifier adapts checks affected by realized events while retaining unaffected requirements. (d) Agents interact through the original tools, receiving tool results and permitted event notifications; verified teacher trajectories provide demonstrations for supervised fine-tuning.

Specifying Executable Events. Building on the selected themes, DynaWeave expands each concise description into a concrete event with three components. First, it specifies the source of the change, the environment entities or resources affected, and the resulting effects. Second, it defines the conditions under which the event is triggered, such as after a particular agent action or when a virtual clock reaches a scheduled time. Detailed trigger mechanisms and their implementation are discussed in Section 3.2. Third, it specifies how the event becomes observable to the agent. We distinguish three modes: *loud* events explicitly notify the agent of the change; *silent* events provide no notification and must be discovered through available tools; and *partial* observations reveal only part of the change, requiring further investigation to establish its implications. Generated event candidates undergo programmatic checks and independent review to assess whether their effects are executable, their causes are plausible, and their consequences are relevant to the task. Accepted events are then passed to the composition stage.

Constructing Dependency Graphs. In the final stage, DynaWeave combines multiple events into a coherent dynamic scenario. Some events depend on earlier changes; for example, a service can recover only after it has failed. An LLM selects compatible events and proposes dependencies based on their causes, effects, and activation conditions. The resulting scenario is represented as a directed acyclic graph (DAG), $\Pi = (\mathcal{N}, \mathcal{A})$, where each node in $\mathcal{N}$ represents an event. An edge $(e_i, e_j) \in \mathcal{A}$ means that event e_i must occur before event e_j can be triggered. The graph does not impose a fixed order on independent events. Its acyclic structure prevents circular dependencies in which events wait for one another. Each proposed graph is validated for structural correctness and consistency among event dependencies, activation conditions, and effects.

3.2 Controlled Dynamic Execution

Once the event DAG is constructed, DynaWeave uses a programmatic execution engine to monitor agent interactions and apply events when their trigger conditions are satisfied. We consider three trigger mechanisms: completion of a specified agent operation, passage of virtual time, and random activation to model the complex mechanisms of changes in real environment.

Action Completion. An event can be triggered after the agent completes a specified tool operation. The execution engine matches the completed call against the operation recorded in the event specification, including relevant arguments or occurrence counts when needed. This places changes at meaningful points in the task, such as immediately after the agent checks a resource's availability. The event is applied after the tool call finishes, so the operation itself remains intact while its result may become outdated before the agent's next decision. For example, after the agent confirms that a hotel room is available, an external booking may reserve that room, requiring the agent to reconsider its choice before completing the reservation.

Virtual Time. A virtual clock provides a strictly reproducible measure of elapsed time within the simulated environment. Starting from an initial state of $c_0 = 0$, the clock advances after each executed tool operation according to $c_{k+1} = c_k + d(a_k)$, where $d(a_k)$ is the simulated duration of action a_k. We use the durations pre-defined by the source environment when available; otherwise, we assign fixed durations to tool operations and keep them strictly consistent across all agent evaluations. To precisely determine an event's scheduled time c_e, we accumulate these simulated action durations along the task's reference static trajectory and select a strategic time when the event's prerequisite conditions hold and its effects can still influence subsequent decisions. During dynamic execution, the event is automatically activated at the first interaction boundary where $c_k \geq c_e$ and its eligibility conditions are satisfied.

Random Activation. At each interaction interval, an event that satisfies its eligibility conditions can be activated with a specified probability p_e. We fix the random seed before execution to make the random activation process reproducible. An event may remain inactive when the interaction ends, and the verifier accounts only for changes that actually occurred.

Under all three mechanisms, each event is applied at most once. When an event occurs, the engine applies its specified effects, provides only the observations allowed by its observability mode, and records the changes separately from those caused by the agent's tool calls. These records support the dynamic verifier described in Section 3.3.

3.3 Dynamic Verifier Adaptation

Given the original verifier V_0 and the generated event specifications, DynaWeave uses an LLM to identify checks whose validity depends on the information, state, or task requirements affected by each event. Unaffected checks are retained, while those whose expected outcomes have changed are revised accordingly. A task requirement is removed only when the user explicitly cancels it. Changes that affect how a task can be completed do not, by themselves, alter the requested outcome. For example, consider a meeting for which the user changes only the requested start time from 15:30 to 16:00. The original verifier would reject a booking at the updated time, whereas removing the time check could accept bookings at an incorrect time. The dynamic verifier instead revises the expected time while retaining other requirements. By contrast, a room becoming unavailable does not, by itself, authorize a change in meeting time. This distinction separates changes in the task from changes in the plan used to complete it.

The proposed modifications are expressed as structured rules and validated before execution for consistency with the task requirements and event specifications. Incompatible modifications are rejected at this stage. During execution, each modification takes effect only when its associated event occurs; inactive events leave the applicable checks unchanged. When multiple events occur, their modifications are applied in the realized execution order, consistent with the DAG. The verifier therefore evaluates the requirements applicable to the observed execution rather than assuming that all potential changes have occurred. When a requirement depends on the timing or order of actions, DynaWeave supplements outcome checks with checks over the execution trace. These checks assess whether the agent's actions satisfy the requirements in effect at the relevant point in execution, while allowing different valid action sequences rather than prescribing a fixed solution.

Before accepting a generated verifier, DynaWeave checks that it agrees with the original verifier when events are disabled. Strict validation additionally tests two solutions: a solution to the original static task that the event is expected to invalidate, and an adapted solution that satisfies the revised requirements. The verifier should reject the first and accept the second. This comparison provides evidence that the event meaningfully affects the task and that the adapted verifier distinguishes an invalid response from a valid adaptation.

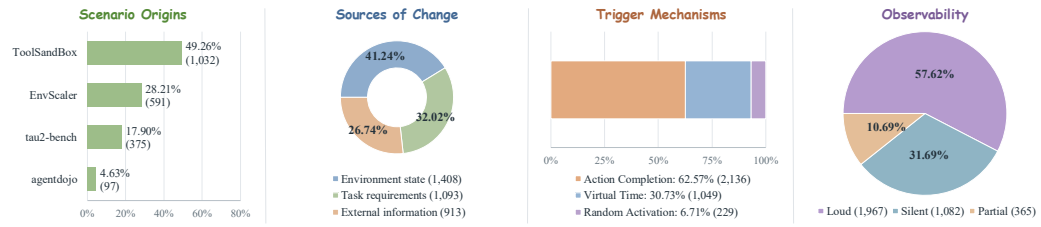


Figure 2: Statistics of the dynamic scenario corpus generated by DynaWeave.

3.4 Interaction with Dynamic Scenarios

DynaWeave retains the tools, argument schemas, and interaction protocol of the source benchmark. Dynamic scenarios change how the environment evolves and what the agent observes, without requiring a new interaction interface.

We model the interaction as a partially observable Markov decision process (POMDP). The action space remains unchanged, while the hidden state is extended to $x_t = (S_t, g_t, z_t)$. Here, S_t contains the environment state and external information, g_t denotes the current task requirements, and z_t records the virtual clock, the events that have occurred, and the action counts used to determine when subsequent events are triggered.

At decision step t, let $h_t = (u, o_0, a_0, \ldots, o_{t-1}, a_{t-1})$ denote the history consisting of the initial task instruction u and all preceding observations and actions. The current observation o_t contains newly received tool results, user messages, and any event notifications permitted by the specified observability modes. The agent selects its next action based on both h_t and o_t:

$$a_t \sim \pi_\theta(\cdot \mid h_t, o_t), \qquad (x_{t+1}, o_{t+1}) \sim K_\Pi(\cdot \mid x_t, a_t), \tag{1}$$

where π_θ is the agent's policy and K_Π jointly specifies the next state and observation, accounting for tool execution and exogenous events. After a tool call, an event may change S_t or, when authorized, revise g_t before the agent's next decision. The agent must therefore use subsequent observations to determine whether its earlier information and plan remain valid. At the end of the interaction, V_Π evaluates whether the agent has satisfied the task requirements, taking into account the events that actually occurred.

For supervised fine-tuning, we collect trajectories from a teacher model. These trajectories record the observations available to the teacher, including event notifications, user messages, or changes in tool results, together with its subsequent actions. They provide demonstrations of how the teacher responds to changes and adjusts its actions during task execution.

4 Experiments

4.1 Experiment Setup

Training Details. We apply DynaWeave to EnvScaler (Song et al., 2026) ToolSandbox (Lu et al., 2025), AgentDojo (Debenedetti et al., 2024), and τ^2-bench (Barres et al., 2025), extending their executable environments with dynamic scenarios. The resulting corpus contains 2,095 dynamic scenarios and 3,414 events across 36 environments, covering healthcare, scheduling, mobile assistance, workplace applications, travel, retail, banking, and telecommunications. Figure 2 summarizes the corpus statistics. Using Qwen3.5-35B-A3B as the teacher model, we collect 2,095 verified interaction trajectories for supervised fine-tuning (SFT) of Qwen3.5 models. We fine-tune the models in non-thinking mode using LLaMA-Factory and LoRA with a rank of 16. All models are trained for two epochs with a learning rate of 1×10^{-4}, an effective batch size of 64, and a maximum sequence length of 16,384 tokens.

Evaluation Benchmarks. We evaluate DynaWeave on three dynamic benchmarks to assess generalization beyond the training environments. Gaia2 (Froger et al., 2026) provides asynchronous application environments that evolve independently of the agent. It evaluates execution, search, ambiguity resolution, adaptation, and reasoning under time constraints. CostBench (Liu et al., 2026)

evaluates planning in a travel environment where tools have different costs. Its dynamic conditions change tool costs, user preferences, execution lengths, or tool availability, requiring agents to revise their plans while controlling cost. ToolMaze (Zhu et al., 2026) evaluates recovery from explicit tool errors and from corrupted outputs without explicit error messages. Both types of failure can be transient or permanent.

4.2 Main Results

Table 1 presents the performance of DynaWeave on three dynamic benchmarks. Across all three model sizes, DynaWeave improves overall scores on Gaia2 and ToolMaze, as well as task completion on CostBench. These gains encompass more reliable tool use, more efficient task completion, and better aggregate recovery from tool failures, demonstrating that DynaWeave enhances agents' ability to adapt to dynamic scenarios and generalize beyond the training environments.

DynaWeave transfers across distinct forms of environmental change. The three benchmarks test different dynamic scenarios, from asynchronous application updates to changing planning constraints and tool failures. DynaWeave yields gains across these settings: for 4B, Gaia2 overall improves from 3.3 to 5.0, CostBench completion rises from 72.41% to 90.81%, and ToolMaze success under perturbations increases from 21.25% to 25.38%. Both the 2B and 9B models likewise improve across all three benchmarks. DynaWeave-9B even surpasses GPT-4o and Gemini-3.1-Pro-Preview on ToolMaze. These consistent improvements across out-of-domain environments show that DynaWeave learn to adjust the actions and plans in response to dynamic changes, rather than merely reproduce tool-use patterns from the training environments.

The gains extend to both execution reliability and efficiency. DynaWeave improves task completion without sacrificing execution efficiency. On CostBench, the 9B model raises completion to 92.39% while reducing its cost gap from 0.755 to 0.140 and invalid tool use from 7.08% to zero. ToolMaze likewise shows higher mean recovery rates and lower normalized recovery costs at every model size (The detailed results of ToolMaze are deferred to Appendix A), alongside improvements in Gaia2 Search and Execution. These joint gains are consistent with agents obtaining updated information when needed and revising their actions, rather than continuing a plan based on outdated assumptions. A targeted query after a change can incur an immediate cost but prevent repeated failed calls or actions that no longer advance the task. Responding appropriately to the new information can therefore improve both the likelihood of completion and the efficiency of reaching it.

DynaWeave improves models across different sizes. DynaWeave consistently improves models of all evaluated sizes. Larger models retain stronger overall performance after training across the three benchmarks, suggesting that their underlying reasoning and tool-use capabilities may help them interpret changes and select appropriate responses. Targeted training also narrows the gap between model sizes: the trained 4B model outperforms the original 9B model on both Gaia2 overall and CostBench task completion. These results show that DynaWeave can strengthen smaller models while providing further gains for larger ones.

Despite these gains, models trained with DynaWeave still struggle when tool results are incorrect but contain no explicit error message, or when success requires responding to changes at the right time, as reflected in ToolMaze's corrupted-output conditions and Gaia2's Adaptability and Time categories. These cases require agents to recognize when information is unreliable or outdated and decide when to check it again. The limitations may stem from the absence of these types of changes in the dynamic scenarios used for training.

4.3 Data Efficiency and Performance Saturation

We examine how the number of training trajectories affects Qwen3.5-4B performance, keeping the source tasks, environments, and number of training epochs fixed. Figure 3 shows that adding trajectories improves task completion on CostBench and both task success under perturbations and the composite score on ToolMaze. A substantial share of these gains is achieved with 1,048 trajectories, approximately half of the full corpus. At this size, CostBench task completion reaches 87.08% and the ToolMaze composite score reaches 28.91, compared with 90.81% and 29.99, respectively, when all 2,095 trajectories are used. The gains become smaller as the training set grows.

Table 1: Main results on Gaia2, CostBench, and ToolMaze. For CostBench, Cost Gap measures excess tool execution cost relative to the optimal reference; TCR (Task Completion Ratio) is the percentage of correctly completed tasks; and ITUR (Invalid Tool-Use Ratio) is the percentage of invalid tool calls. ToolMaze Avg denotes its official composite score. Detailed ToolMaze results and the definition of Avg are provided in Appendix A.

	Gaia2						CostBench			ToolMaze
Model	Execution	Search	Ambiguity	Adaptability	Time	Overall	Cost Gap ↓	TCR	ITUR ↓	Avg
Advanced models										
GPT-5.5	79.4	96.2	46.9	55.0	4.4	56.4	0.000	95.01	0.00	35.94
Gemini-3.1-Pro-Preview	78.6	92.8	40.6	45.9	2.1	52.0	0.010	95.28	1.42	32.19
GLM-5.1	71.2	83.8	39.4	56.9	1.2	50.5	0.030	93.18	0.29	32.28
GPT-4o	8.3	17.5	4.4	6.2	5.8	7.4	1.179	90.29	19.96	31.29
Qwen3.5-2B	0.0	2.1	0.0	0.0	0.0	0.4	1.998	3.00	19.81	18.52
+DynaWeave	**1.0**	**5.2**	0.0	0.0	0.0	**1.3**	**1.242**	**70.34**	**0.16**	**23.49**
Qwen3.5-4B	6.3	10.4	0.0	0.0	0.0	3.3	0.650	72.41	13.98	26.26
+DynaWeave	**8.3**	**14.6**	**2.1**	0.0	0.0	**5.0**	**0.331**	**90.81**	**2.47**	**29.99**
Qwen3.5-9B	6.3	16.7	0.0	0.0	0.0	4.6	0.755	86.76	7.08	30.59
+DynaWeave	**10.4**	**26.0**	**3.1**	**2.1**	**1.0**	**8.5**	**0.140**	**92.39**	**0.00**	**32.28**

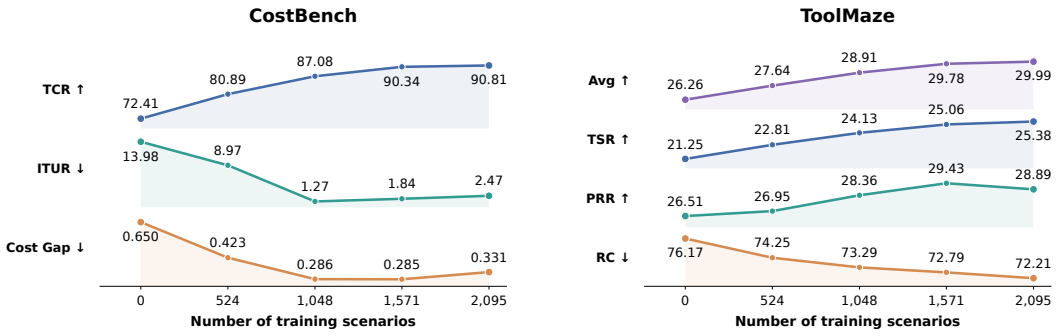


Figure 3: Effect of training scenario quantity on Qwen3.5-4B performance on CostBench (left) and ToolMaze (right). Zero scenarios denotes the baseline Qwen3.5-4B.

The effects on other metrics are less uniform. ToolMaze's normalized recovery cost decreases across all evaluated training set sizes. However, CostBench's invalid tool use ratio and cost gap reach their lowest values before the full corpus is used, while ToolMaze's PRR peaks at 1,571 trajectories and then declines slightly. These results show that higher task completion does not necessarily coincide with improvements in every measure of reliability and efficiency. One possible explanation is that SFT trains the model to imitate demonstrated actions without directly optimizing execution cost. Overall, training on more trajectories generated with DynaWeave improves task performance within the evaluated range, but the gains diminish and additional data does not improve all metrics consistently.

4.4 Contributions of Different Event Sources

To examine the contributions of different sources of change, we compare training on a mixture of all three sources with training on subsets that each exclude one source: *Task Requirement*, *Environment State*, or *External Information*. For this comparison, we use Qwen3.5-4B and 1,048 trajectories for each setting. As shown in Table 2, the mixed subset achieves the highest task completion rate on CostBench and the highest composite score on ToolMaze. It also yields the highest ToolMaze task success under perturbations and the lowest recovery cost. CostBench task completion drops most when Task Requirement events are excluded, whereas ToolMaze success under perturbations and recovery efficiency decline most when Environment State events are excluded. These patterns suggest complementary benefits from learning to revise task goals and adjust execution to changes in the environment. In particular, demonstrations involving changes in task progress may help agents identify which actions remain necessary and avoid redundant execution during recovery.

Table 2: Effect of event-source composition on Qwen3.5-4B performance. Data denotes the number of training trajectories; Mixed includes all three event sources. For ToolMaze, NP (No Perturbation) reports task success without injected failures; TSR (Task Success Rate) reports average task success under tool failures or corrupted outputs; and RC (Recovery Cost) reports average normalized recovery cost under the same conditions. Bold marks the best value in each metric column.

		CostBench			ToolMaze			
Training subset	Data	Gap ↓	TCR ↑	ITUR ↓	NP ↑	TSR ↑	RC ↓	Avg ↑
Mixed	1,048	0.286	**87.08**	1.27	61.75	**24.13**	**73.29**	**28.91**
w/o Task Requirement	1,048	**0.278**	50.39	1.41	61.25	22.31	75.83	26.86
w/o Environment State	1,048	0.287	82.68	1.24	61.00	19.06	77.87	24.85
w/o External Information	1,048	0.285	80.58	**0.31**	**64.50**	21.19	76.54	26.45
Mixed	524	0.423	80.89	8.97	59.75	22.81	74.25	27.64

Compared with the mixed subset, excluding External Information events improves ToolMaze task success without injected failures, but lowers average success and increases recovery cost under tool failures or corrupted outputs. This pattern suggests that learning to respond to new information may help agents adjust their actions when execution is disrupted. On CostBench, some subsets that omit an event source achieve a lower cost gap or a lower invalid tool use ratio than the mixed subset, but all have lower task completion rates. These results highlight the need to assess efficiency alongside task completion.

We further compare these subsets with a smaller mixed subset containing 524 trajectories, approximately 25% of the full training corpus. On ToolMaze, this smaller subset achieves a composite score of 27.64, exceeding all three subsets that contain 1,048 trajectories but omit one source. It also achieves higher average task success under perturbations and lower recovery cost than these larger subsets, despite lower success without injected failures. This comparison highlights the importance of data composition as well as quantity. However, differences in task and environment coverage, together with the different numbers of optimization steps across dataset sizes, prevent us from attributing this advantage solely to the inclusion of all three event sources. Overall, the results support retaining all three sources to improve task completion and recovery, rather than increasing data quantity without considering the diversity of changes it covers.

5 Conclusion

We presented DynaWeave, an automated framework for transforming existing static tasks into verifiable dynamic scenarios. The framework generates exogenous events involving task requirements, environment state, and external information, controls their execution, and adapts the original verifier to the changes that actually occur. It preserves the original tools and unaffected task requirements, allowing existing environments to support training under changing conditions. Supervised fine-tuning on 2,095 verified trajectories collected across 36 environments improves the overall performance of Qwen3.5 models of all three evaluated sizes on Gaia2, CostBench, and ToolMaze, supporting generalization beyond the training environments. Our analyses further show diminishing gains from increasing data quantity and highlight the value of including all three event sources for task completion and recovery. However, higher task completion does not always coincide with better execution efficiency, and responding to changes at the appropriate time remains challenging. Future work should investigate how scenario construction and training can better address these limitations. Overall, DynaWeave provides a practical way to use existing task collections to train agents not only to execute plans, but also to revise their actions as conditions change.

AI use statement

We used large language models to generate synthetic data, as described in the paper, and to improve the language and clarity of the manuscript. The authors take full responsibility for the content of the paper, including the generated data, experimental results, and conclusions.

REFERENCES

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.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents, 2024. URL https://arxiv.org/abs/2406.13352.

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.

Runnan Fang, Shihao Cai, Baixuan Li, Jialong Wu, Guangyu Li, Wenbiao Yin, Xinyu Wang, Xiaobin Wang, Liangcai Su, Zhen Zhang, Shibin Wu, Zhengwei Tao, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. Towards general agentic intelligence via environment scaling, 2025. URL https://arxiv.org/abs/2509.13311.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine Benhalloum, Grégoire Mialon, and Thomas Scialom. Are: Scaling up agent environments and evaluations, 2025. URL https://arxiv.org/abs/2509.17158.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine Benhalloum, Grégoire Mialon, and Thomas Scialom. Gaia2: Benchmarking llm agents on dynamic and asynchronous environments, 2026. URL https://arxiv.org/abs/2602.11964.

Haonian Ji, Kaiwen Xiong, Siwei Han, Peng Xia, Shi Qiu, Yiyang Zhou, Jiaqi Liu, Jinlong Li, Bingzhou Li, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Clawarena: Benchmarking ai agents in evolving information environments, 2026. URL https://arxiv.org/abs/2604.04202.

Guangrui Li, Yaochen Xie, Yi Liu, Ziwei Dong, Xingyuan Pan, Tianqi Zheng, Jason Choi, Michael J. Morais, Binit Jha, Shaunak Mishra, Bingrou Zhou, Chen Luo, Monica Xiao Cheng, and Dawn Song. The world won't stay still: Programmable evolution for agent benchmarks, 2026. URL https://arxiv.org/abs/2603.05910.

Jiayu Liu, Cheng Qian, Zhaochen Su, Qing Zong, Shijue Huang, Bingxiang He, and Yi R. Fung. Costbench: Evaluating multi-turn cost-optimal planning and adaptation in dynamic environments for llm tool-use agents, 2026. URL https://arxiv.org/abs/2511.02734.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities, 2025. URL https://arxiv.org/abs/2408.04682.

Fanqing Meng, Lingxiao Du, Zijian Wu, Guanzheng Chen, Xiangyan Liu, Jiaqi Liao, Chonghe Jiang, Zhenglin Wan, Jiawei Gu, Pengfei Zhou, Rui Huang, Ziqi Zhao, Shengyuan Ding, Ailing Yu, Bo Peng, Bowei Xia, Hao Sun, Haotian Liang, Ji Xie, Jiajun Chen, Jiajun Song, Liu Yang,

Ming Xu, Qionglin Qiu, Runhao Fu, Shengfang Zhai, Shijian Wang, Tengfei Ma, Tianyi Wu, Weiyang Jin, Yan Wang, Yang Dai, Yao Lai, Youwei Shu, Yue Liu, Yunzhuo Hao, Yuwei Niu, Jinkai Huang, Jiayuan Zhuo, Zhennan Shen, Linyu Wu, Hannah Yao, Charles Chen, Cihang Xie, Yuyin Zhou, Jiaheng Zhang, Zeyu Zheng, Mengkang Hu, and Michael Qizhe Shieh. Clawmark: A living-world benchmark for multi-turn, multi-day, multimodal coworker agents, 2026. URL https://arxiv.org/abs/2604.23781.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents, 2025. URL https://arxiv.org/abs/2405.14573.

Gyuhyeon Seo, Jungwoo Yang, Junseong Pyo, Nalim Kim, Jonggeun Lee, and Yohan Jo. Simuhome: A temporal- and environment-aware benchmark for smart home llm agents, 2026. URL https://arxiv.org/abs/2509.24282.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Envscaler: Scaling tool-interactive environments for llm agent via programmatic synthesis, 2026. URL https://arxiv.org/abs/2601.05808.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents, 2024. URL https://arxiv.org/abs/2407.18901.

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

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024. URL https://arxiv.org/abs/2404.07972.

Jundong Xu, Qingchuan Li, Jiaying Wu, Yihuai Lan, Shuyue Stella Li, Huichi Zhou, Bowen Jiang, Lei Wang, Jun Wang, Anh Tuan Luu, Caiming Xiong, Hae Won Park, Bryan Hooi, and Zhiyuan Hu. Evoarena: Tracking memory evolution for robust llm agents in dynamic environments, 2026. URL https://arxiv.org/abs/2606.13681.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

Dongsheng Zhu, Xuchen Ma, Yucheng Shen, Xiang Li, Yukun Zhao, Shuaiqiang Wang, Lingyong Yan, and Dawei Yin. When tools fail: Benchmarking dynamic replanning and anomaly recovery in llm agents, 2026. URL https://arxiv.org/abs/2606.05806.

A Detailed Evaluation Results

Tables 3 provide the detailed evaluation results on ToolMaze. For ToolMaze, NP denotes execution without injected failures; P1 and P2 introduce explicit transient and permanent failures, respectively, whereas P3 and P4 introduce implicit transient and permanent output corruption.

Table 3: Detailed results on ToolMaze.

	TSR ↑						PRR ↑					RC ↓					Overall
Model	NP	P1	P2	P3	P4	Avg	P1	P2	P3	P4	Avg	P1	P2	P3	P4	Avg	Avg ↑
GPT-5.5	70.00	67.25	26.75	16.75	14.00	31.19	86.09	29.49	18.89	14.75	37.31	35.61	73.02	81.33	83.83	68.45	35.94
Deepseek-V4-Pro	69.75	68.00	34.75	12.00	9.75	31.12	84.70	37.74	12.41	12.00	36.71	29.98	63.55	85.19	87.55	66.57	36.33
GLM-5.1	75.75	60.00	32.50	8.00	8.50	27.25	67.19	35.33	9.04	10.15	30.43	37.86	66.47	88.65	89.11	70.52	32.28
Gemini-3.1-Pro-Preview	71.25	59.75	30.00	6.75	9.00	26.38	73.49	33.52	8.09	10.84	31.48	37.53	67.64	88.64	87.24	70.26	32.19
GPT-4o	59.25	47.00	33.50	9.25	11.50	25.31	70.49	27.72	12.72	12.71	30.91	35.97	78.34	81.77	80.54	69.16	31.29
Qwen3.5-2B	42.50	24.50	14.00	5.75	4.25	12.13	46.70	12.10	12.08	5.00	18.97	67.76	81.27	88.00	89.37	81.60	18.52
+DynaWeave	53.25	49.75	22.50	7.00	9.50	22.19	48.81	18.37	12.85	6.46	21.62	63.66	79.27	86.86	88.46	79.56	23.49
Qwen3.5-4B	57.25	45.50	23.75	9.00	6.75	21.25	64.68	22.49	12.13	6.74	26.51	53.08	74.28	87.71	89.60	76.17	26.26
+DynaWeave	65.00	54.25	27.25	8.50	11.50	25.38	67.49	26.34	10.22	11.52	28.89	43.84	71.57	87.20	86.24	72.21	29.99
Qwen3.5-9B	64.75	56.50	31.50	7.50	10.00	26.38	69.06	30.13	8.76	11.39	29.83	41.49	68.74	90.02	88.20	72.11	30.59
+DynaWeave	66.25	56.50	31.75	14.75	10.25	28.31	67.91	29.75	18.36	10.21	31.56	43.56	67.63	83.96	87.35	70.62	32.28