DAVINCI-TOOL: SCALING AGENTIC TOOL-USE SUPERVISION WITH VERIFIABLE EXECUTABLE ENVIRONMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reliable agentic tool use requires training data that pair diverse multi-step workflows with executable evidence that the requested outcomes were achieved. Scaling either trajectory coverage or verification alone is insufficient as new environments introduce distinct states, rules, and dependencies that must also be reflected in the success criteria. To address this challenge, we introduce **Davinci-Tool**, a 30B tool-use model, and **Davinci-ToolData**, an execution-grounded dataset designed to scale environment coverage and outcome verification together. Davinci-ToolData preserves restorable states, execution traces, outcomes, and verifiers as a unified substrate for supervision. Two complementary pipelines populate this substrate: Execution-Grounded Task Synthesis (EGTS) derives verified demonstrations from independently authored systems, and Executable Environment Synthesis (EES) synthesizes auditable environments, state-grounded tasks, and executable verifiers across single and coordinated domains. These pipelines yield 30,395 verified trajectories spanning 2,260 invoked tool names. We first train on these demonstrations with supervised fine-tuning, then reuse EES states and verifiers to compare alternative actions and construct outcome-grounded preferences for Direct Preference Optimization. Davinci-Tool improves Qwen3-Coder-30B-A3B-Instruct from 52.4 to 88.2 on τ^2-bench, from 30.4 to 62.8 on BFCL v4 multi-turn, and from 16.9 to 56.2 on MCP-Atlas, outperforming Qwen3-Coder-480B-A35B-Instruct on all three benchmarks with only $1/16$ of its total parameters.

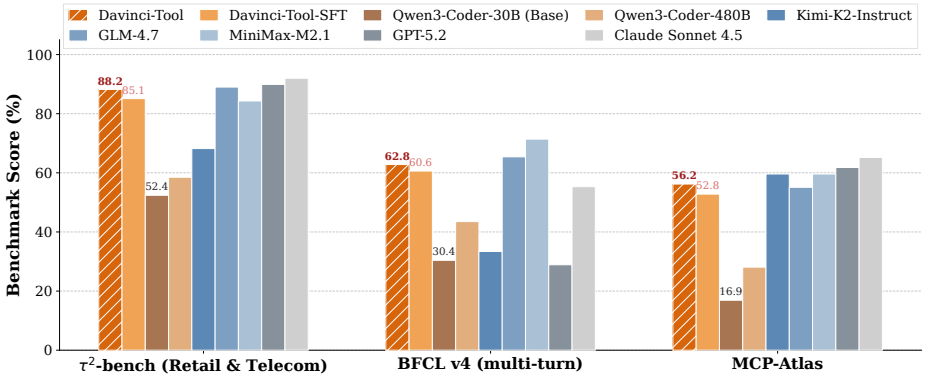


Figure 1: Performance on τ^2-bench, BFCL v4 multi-turn, and MCP-Atlas. Davinci-Tool-30B after DPO is shown with hatching, with its SFT checkpoint included for comparison. Detailed results and score provenance are reported in Table 2.

1 INTRODUCTION

Tool use enables language models to move beyond generating responses and perform real-world tasks, powering applications such as coding agents (Yang et al., 2024) and deep research systems (Zheng

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.

et al., 2025; Li et al., 2026b). Such tasks require stateful multi-step interaction, where models select tools, interpret observations, and adapt actions toward desired outcomes (Zhou et al., 2024; Xie et al., 2024a; Patil et al., 2025; Yao et al., 2024). Training such behavior therefore requires data that connect user intent, multi-step execution, and verifiable environment outcomes.

However, scaling such data presents two coupled challenges: covering diverse tool-use workflows and verifying their outcomes. Prior efforts improve either workflow coverage through expanded tool inventories and synthetic interactions (Qin et al., 2024; Liu et al., 2025; Wang et al., 2026a) or reliability through execution validation and executable verification (Liu et al., 2025; Ouajdi et al., 2026; Song et al., 2026). As environments become more diverse and cross-domain, verification must capture environment-specific states, rules, and dependencies. Therefore, environment coverage and outcome verification need to scale together.

Environment expansion follows two complementary paradigms. Independently authored systems provide realistic tools and operational rules but require adaptation of reset, execution, and outcome-checking interfaces, as exemplified by ToolSandbox and AppWorld (Lu et al., 2025; Trivedi et al., 2024). Synthesized environments provide controllable states and executable criteria for systematic task generation but may inherit assumptions from the environment generator, as explored by EnvScaler and AutoWebWorld (Song et al., 2026; Wu et al., 2026a). These complementary sources motivate a unified execution-grounded framework that supports heterogeneous environments under shared outcome verification.

To address these settings, we introduce **Davinci-Tool**, a model for reliable multi-step tool use in stateful environments, together with **Davinci-ToolData**, an execution-grounded training dataset spanning diverse tool-use environments. Our central design principle is to preserve executable evidence throughout the training pipeline: each example is grounded in a restorable initial state, an action–observation trace, the resulting environment outcome, and an executable verification result. This common interface allows data from heterogeneous environment sources to be accepted under a shared outcome-based criterion and enables the same states and verifiers to support both supervised and preference learning.

To construct **Davinci-ToolData**, we combine independently authored systems with synthesized executable environments. For authored systems, **Execution-Grounded Task Synthesis (EGTS)** explores supported interactions to establish achievable outcomes and their evidence, synthesizes user tasks from these outcomes, and replays them from corresponding initial states to collect verified trajectories. This separates outcome discovery from demonstration construction: exploration determines what can be achieved, while replay collects trajectories conditioned on the synthesized user request. For synthesized environments, **Executable Environment Synthesis (EES)** constructs and audits rule-governed domains before generating state-grounded tasks and executable verifiers. EES produces scenarios within individual domains (**EES-Single**) and composes compatible domains through shared entities, policies, and state dependencies before generating coordinated tasks over the resulting executable world (**EES-Cross**). Together, these strategies provide complementary supervision from single-domain to cross-domain workflows. Davinci-ToolData contains 30,395 verified trajectories covering 2,260 distinct invoked tool names (Table 1).

We first fine-tune Qwen3-Coder-30B-A3B-Instruct (Yang et al., 2025a) on Davinci-ToolData, producing Davinci-Tool-30B-SFT. To complement successful demonstrations with comparisons between alternative decisions, we reuse restorable EES scenarios and their executable verifiers for decision-level preference learning. At selected points in SFT rollouts, we restore the same executable context, sample alternative decisions, and use verified downstream outcomes to construct preference pairs. Only divergent assistant turns are optimized with Direct Preference Optimization (DPO) (Rafailov et al., 2023). Unlike approaches that rely on tree search with intermediate self-evaluation (Xie et al., 2024b; Putta et al., 2024), our construction uses bounded sibling rollouts and executable outcome checks without an additional expert action generator or process evaluator.

Davinci-Tool-30B achieves substantial gains across three held-out tool-use benchmarks. SFT improves the base model from 52.4 to 85.1 on τ^2-bench, from 30.4 to 60.6 on BFCL v4 multi-turn, and from 16.9 to 52.8 on MCP-Atlas, while decision-level DPO further raises these scores to 88.2, 62.8, and 56.2, respectively. Despite using only $1/16$ as many total parameters, Davinci-Tool-30B outperforms Qwen3-Coder-480B-A35B-Instruct (Yang et al., 2025a) on all three benchmarks. Source-composition studies further show that the full EGTS, EES-Single, and EES-Cross mixture provides

the strongest SFT performance among evaluated configurations, while outcome-grounded preference learning provides additional gains after SFT.

Our contributions are:

- We introduce **Davinci-Tool**, a tool-use model trained through an executable-environment framework that connects verified demonstration construction with outcome-grounded, decision-level preference learning.
- We construct **Davinci-ToolData**, an execution-grounded dataset with 30,395 verified tool-use demonstrations and 2,260 invoked tool names. EGTS builds task-conditioned demonstrations through exploration and verified replay, while EES constructs and audits executable environments for single-domain and coordinated cross-domain tasks.
- We demonstrate strong tool-use performance at the 30B scale. Davinci-Tool-30B improves over its backbone by 35.8, 32.4, and 39.3 percentage points on τ^2-bench, BFCL v4 multi-turn, and MCP-Atlas, respectively, outperforming Qwen3-Coder-480B-A35B-Instruct across all three benchmarks. The same Davinci-ToolData supervision also substantially improves the tested smaller models.

2 Related Work

Tool-use data construction. ToolLLM (Qin et al., 2024) and ToolACE (Liu et al., 2025) scale supervision through synthesis, while MCP-Flow (Wang et al., 2026a) and TOUCAN (Xu et al., 2025b) collect trajectories from MCP environments. Execution-first methods such as SyntheticAgent-TraceQA (Ouajdi et al., 2026) and Trajectory2Task (Wang et al., 2026b) derive tasks from observed executions; Trajectory2Task also collects successful task rollouts, while AgentEvolver (Zhai et al., 2025) explores for RL data. EGTS builds on this direction for heterogeneous authored systems: an originating execution provides outcome evidence and replay guidance, and the synthesized request is re-executed from the corresponding initial state to collect a verified task-conditioned demonstration. Unlike AgentTrek's tutorial-guided replay (Xu et al., 2025a), EGTS derives both guidance and verification evidence from the originating execution.

Environment synthesis. EnvScaler (Song et al., 2026) constructs executable environments, task scenarios, and validation functions, while ASTRA (Tian et al., 2026) synthesizes verifiable sub-environments around question–answer decompositions. EES follows this domain-first direction by auditing reusable executable domains before task generation. EES-Single generates state-grounded tasks and verifiers within individual domains, while EES-Cross composes audited domains through shared entities, roles, policies, and state dependencies before coordinated task generation. AppWorld (Trivedi et al., 2024) and Toolathlon (Li et al., 2026a) support multi-application interaction; EES-Cross instead uses executable domain composition to construct verified training supervision.

Preference learning. ETO (Song et al., 2024) learns from success–failure trajectory pairs, while MCTS-DPO (Xie et al., 2024b), Agent Q (Putta et al., 2024), and AlphaLLM-CPL (Wang et al., 2024) derive finer-grained preferences through search-based alternatives and evaluation. Our construction branches from restorable EES states: the SFT policy generates bounded sibling decisions and continuations under the same executable context, and verified outcomes label the divergent turns. This uses bounded siblings rather than recursive tree search and executable verification rather than a separate process evaluator, while reusing the restoration and verification infrastructure from demonstration construction. Further discussion appears in Appendix F.

3 Verified Demonstration Construction

To jointly scale environment coverage and outcome verification, Davinci-ToolData follows two complementary construction routes (Figure 2). Both operate under a shared execution and verification interface but differ in how their environments and supervision are obtained. For independently authored systems, EGTS identifies feasible outcomes through exploration, synthesizes outcome-grounded tasks, and collects verified task-conditioned demonstrations through guided replay. For synthesized environments, EES constructs and audits reusable executable domains, then generates state-grounded tasks and executable verifiers for individual domains (EES-Single) and coordinated cross-domain workflows (EES-Cross) linked by shared entities, policies, and state dependencies.

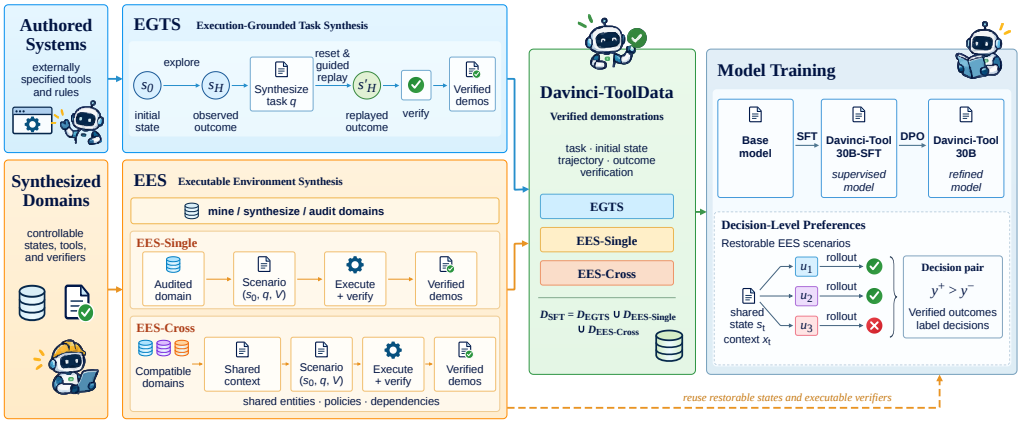


Figure 2: Construction of Davinci-ToolData and training of Davinci-Tool. EGTS converts explored outcomes in authored systems into verified task-conditioned replay demonstrations, while EES constructs audited executable environments for verified single-domain and cross-domain trajectories. The resulting demonstrations support SFT, and restorable EES states and verifiers are subsequently reused for decision-level DPO.

Definition of tool-use trajectory. We represent a resettable tool environment as $W = (\mathcal{S}, \mathcal{T}, \rho, P, \mathcal{E})$, where $\mathcal{S}$ is the state space, $\mathcal{T}$ is the available tool set, $\rho(s_0)$ restores the environment to an initial-state snapshot s_0, P executes tool calls and produces state transitions and observations, and $\mathcal{E}$ is the outcome-evidence space. The restoration operator ρ enables replaying tasks or branching alternative decisions from the same executable state.

Given a user task q and initial state s_0, the model maintains an interaction history $h_t = (a_1, o_1, \ldots, a_{t-1}, o_{t-1})$. At step t, the tool-use policy π_θ selects a tool call a_t and receives observation o_t: $a_t \sim \pi_\theta(\cdot \mid q, h_t, \mathcal{T})$, $(s_t, o_t) = P(s_{t-1}, a_t)$. After H tool interactions, the model produces final response y, yielding $\tau = (a_1, o_1, \ldots, a_H, o_H, y)$ with terminal state s_H. Given outcome evidence $\xi \in \mathcal{E}$, an execution is accepted when $\mathrm{Verify}_W(q, s_0, \tau, s_H, \xi) = 1$. EGTS evaluates replay using exploration-derived evidence, whereas EES instantiates verification through executable task predicates. Accepted trajectories provide SFT supervision, while restorable EES states and verifiers further support decision-level preference learning (Section 4.2). Appendix A details the shared execution and verification interface.

3.1 EGTS: From Explored Outcomes to Verified Demonstrations

EGTS separates *outcome discovery* from *task-conditioned demonstration construction*. An exploratory execution first establishes a feasible outcome in an independently authored environment W. EGTS then synthesizes a user task q from captured outcome evidence, restores the same initial state with $\rho(s_0)$, and collects a new trajectory conditioned on q. Thus, exploration determines *what can be achieved*, while replay demonstrates *how to achieve it for the user request*. This design builds on execution-first task synthesis (Ouajdi et al., 2026; Wang et al., 2026b) while separating discovery traces from SFT supervision.

Outcome exploration and evidence collection. EGTS adapts each authored system to the shared interface, providing callable tools $\mathcal{T}$, restoration ρ, transitions P, and source-specific evidence capture in $\mathcal{E}$. Starting from s_0, coverage-guided exploration produces trajectory τ_{exp} and terminal state s_H. The collector extracts evidence $\xi_{\mathrm{exp}} \in \mathcal{E}$ characterizing the achieved outcome:

$$(s_0, \tau_{\mathrm{exp}}, s_H) \xrightarrow{\text{capture in } W} \xi_{\mathrm{exp}}. \tag{1}$$

For state-modifying environments, ξ_{exp} retains relevant state transitions; for retrieval-oriented environments, it retains grounded observations. Failed executions, insufficient evidence, or outcomes without usable evidence are discarded. Reusable skill guidance and prior execution experience help exploration reach more involved workflows (Yang et al., 2025b; Ramrakhya et al., 2026).

Task synthesis. Only after an outcome is executed and recorded does EGTS generate a user task, $q \sim G_{\mathrm{task}}(s_0, \tau_{\mathrm{exp}}, s_H, \xi_{\mathrm{exp}})$. The task specifies the desired outcome without exposing the

exploratory action sequence. For mutation tasks, it requests state changes supported by the observed transition; for retrieval tasks, it requests information grounded in ξ_{exp}. Candidate tasks are retained only when they are specific, natural, feasible from s_0, and fully supported by evidence.

Trajectory construction and verification. EGTS extracts task-relevant procedural information from $(\tau_{\text{exp}}, \xi_{\text{exp}})$ as replay guide g, invokes $\rho(s_0)$, and executes task q to obtain trajectory τ_{rep} terminating in s'_H. The guide is used only during collection and excluded from serialized SFT examples.

The replay is accepted when $\text{Verify}_W(q, s_0, \tau_{\text{rep}}, s'_H, \xi_{\text{exp}}) = 1$. For mutation tasks, this requires recovering the canonical outcome from exploration, including $\phi_W(s'_H) = \phi_W(s_H)$, where ϕ_W removes source-specific volatile state irrelevant to task semantics. For retrieval tasks, tool observations and final responses must be grounded in ξ_{exp}. Additional checks cover tool-call executability, argument–observation consistency, action ordering, and task satisfaction.

Each accepted SFT example retains user task q, visible tool set $\mathcal{T}$, and task-conditioned replay trajectory τ_{rep}, together with verification provenance. The exploratory trajectory τ_{exp} and replay guide g support construction but are not used as model-visible SFT context. Appendix B.1 provides the full exploration, replay, and verification procedure.

3.2 EES: From Executable Domains to Verified Trajectories

Unlike EGTS, which adapts authored systems and derives tasks from explored outcomes, EES jointly constructs executable environments, state-grounded tasks, and their success criteria. It first synthesizes and audits reusable domains, then instantiates verified scenarios within individual domains (EES-Single) or composed domains (EES-Cross). Since EES controls state initialization, transitions, and verification, each accepted trajectory retains a restorable state and executable predicates that can be reused for preference construction (Section 4.2).

3.2.1 EES-Single: Audited Single-Domain Scenarios

Domain synthesis and audit. An EES domain is $d_i = (W_i, \Gamma_i)$, where $W_i = (\mathcal{S}_i, \mathcal{T}_i, \rho_i, P_i, \mathcal{E}_i)$ follows the shared tool-environment interface and Γ_i specifies entity schemas and operational rules. Staged synthesis produces schemas, tools, and transition logic. An executor–auditor loop exercises $\mathcal{T}_i$ over initialized states and checks arguments, observations, state transitions, and compliance with Γ_i; detected inconsistencies trigger revision before task generation (Song et al., 2026).

Scenario and verifier construction. For an audited domain, EES generates a scenario $(s_0, q, \mathcal{V})$, where $s_0 \in \mathcal{S}_i$, q is grounded in s_0, and $\mathcal{V} = \{v_1, \ldots, v_J\}$ contains executable predicates for atomic requirements. A mutation target already satisfied in s_0 is regenerated so that success requires an interaction-induced state change. Executing q from $\rho_i(s_0)$ produces trajectory τ and terminal state s_H. Its predicate evaluations form scenario evidence $\xi_{\mathcal{V}} \in \mathcal{E}_i$, and the trajectory is accepted when

$$\text{Verify}_{W_i}(q, s_0, \tau, s_H, \xi_{\mathcal{V}}) = \prod_{j=1}^{J} \mathbb{I}[v_j(\tau, s_H) = 1] = 1. \tag{2}$$

Domain auditing ensures that the environment follows its contract, while scenario predicates verify completion of the requested task.

3.2.2 EES-Cross: Composing Domains into Coordinated Workflows

EES-Cross composes audited domains before task generation to create executable shared worlds where tasks depend on explicit entity, state, or policy relationships rather than unrelated tool collections.

Domain selection and composition. Embedding retrieval proposes related domains, an affinity rubric evaluates functional compatibility and potential dependencies, and bounded random walks select connected groups. Namespace maps separate tool names, entity types, identifiers, and state attributes before composition. The resulting $W_{\text{cross}} = (\mathcal{S}_{\text{cross}}, \mathcal{T}_{\text{cross}}, \rho_{\text{cross}}, P_{\text{cross}}, \mathcal{E}_{\text{cross}})$ preserves each domain's local transitions under a unified runtime; static checks validate initialization, schemas, and namespace consistency.

Shared-world construction. EES-Cross defines shared entities, user roles, linking fields, cross-domain policies, and coherence constraints over W_{cross}. These relations are checked against domain contracts before $\rho_{\text{cross}}(s_0)$ initializes linked records, making the operational dependencies explicit.

Task construction and verification. EES-Cross generates task q and predicates $\mathcal{V}_{\text{cross}}$ from the shared context and s_0. Requirements may transfer an observation or identifier across domains, enforce operation ordering, or constrain state changes through cross-domain policies. Candidate trajectories first pass runtime checks and are then accepted through the shared interface:

$$\text{Verify}_{W_{\text{cross}}}(q, s_0, \tau, s_H, \xi_{\mathcal{V}_{\text{cross}}}) = 1, \tag{3}$$

where $\xi_{\mathcal{V}_{\text{cross}}} \in \mathcal{E}_{\text{cross}}$ records cross-domain predicate outcomes. A retained example uses content from a "Cryo-EM Vendor Call" note to update a mobile reminder; the agent verifies reminder ownership before the update and separately reports the user's notification preference across the `personal_` and `mobile_` interfaces (Appendix C.1.6). Appendix B.2 details EES construction and auditing.

3.3 Environment Coverage and Corpus Statistics

Environment coverage. EGTS covers 8 authored environment families spanning services, APIs, workspaces, web interaction, and simulation. EES comprises 191 audited domains across 19 categories, of which 188 appear in 246 retained cross-domain compositions. Appendix C.1 provides the full source and domain taxonomy.

Table 1: Verified SFT corpus. Tokens and calls are means; tools are distinct invoked serialized names.

Source	Traj.	Tokens	Calls	Tools
EGTS	13,824	10,063.9	6.04	350
EES-Single	8,026	10,372.7	5.87	1,610
EES-Cross	8,545	20,444.7	7.01	1,683
Full SFT corpus	30,395	13,063.8	6.26	2,260

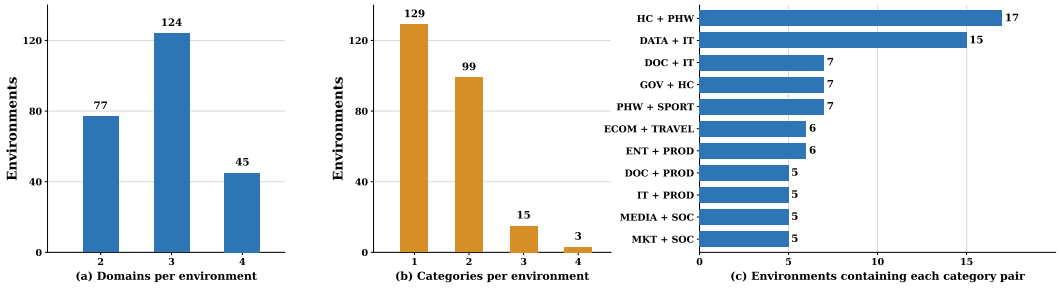


Figure 3: Structure of the 246 retained EES-Cross compositions: (a) constituent domains per environment; (b) represented application categories; and (c) category pairs appearing in at least 5 compositions. Pair counts may overlap when an environment spans more than two categories. Category codes are defined in Table 5.

Composition structure. Figure 3 shows 77 two-domain, 124 three-domain, and 45 four-domain environments. Among them, 117 span multiple application categories and 129 connect distinct systems within one category, demonstrating diversity in both domain count and category composition.

Corpus diversity and interaction scale. Table 1 shows that Davinci-ToolData contains 30,395 verified trajectories with 28,985 distinct normalized requests and 436 declared-tool inventories. Across these inventories, 4,672 tool names are exposed and 2,260 are invoked, producing 190,410 tool calls; source-level tool counts overlap and are not additive. The median serialized length is 9,284 tokens, with 957 trajectories (3.15%) exceeding 32,768 tokens before training-time handling. EES-Cross has the highest mean length and tool-call count, reflecting additional cross-domain coordination. Appendix C.2 provides source-level distributions.

Beyond corpus scale, representation alignment shows source-specific execution-pattern neighborhoods (Appendix C.3); paired base/SFT analysis characterizes benchmark-specific execution changes associated with task recovery (Appendix C.4); and length-controlled analysis shows that EES-Cross introduces domain transitions, entity transfer, and read–write dependencies beyond interaction length (Appendix C.1.4).

4 OUTCOME-GROUNDED MODEL TRAINING

Davinci-Tool is trained in two stages. We first perform supervised fine-tuning (SFT) on the verified demonstrations from Section 3, then reuse restorable EES scenarios and executable verifiers to compare alternative decisions from the same interaction context. Verified downstream outcomes define decision-level preferences for DPO.

4.1 SUPERVISED FINE-TUNING

We fine-tune Qwen3-Coder-30B-A3B-Instruct (Yang et al., 2025a) on Davinci-ToolData's 30,395 verified demonstrations. Each example contains the user task, visible tools, assistant actions, tool observations, and final response. SFT optimizes assistant tokens conditioned on the preceding interaction, while tool outputs remain in context but are excluded from the loss. The resulting policy π_{SFT} is used for preference construction. Training details are provided in Appendix E.

4.2 OUTCOME-GROUNDED PREFERENCE CONSTRUCTION

A verified demonstration provides one successful execution but does not compare alternative decisions from the same interaction state. We construct such comparisons by branching from π_{SFT} rollouts in EES-Single and EES-Cross scenarios. Restorable states allow alternatives to start from the same executable state, while verifiers provide a common outcome criterion for completed branches.

Shared decision contexts. For each selected scenario, π_{SFT} generates a trunk trajectory. At a retained decision point t, we record

$$x_t = (q, m_{<t}, \mathcal{T}_t),$$

where x_t is the model-visible context containing task q, message prefix $m_{<t}$, and visible tools $\mathcal{T}_t$, while s_t is the corresponding restorable environment state. Up to B trunk decision points are retained as forks, with executable state s_t kept separate from model-visible context x_t.

Sibling generation and outcome verification. At each fork, the trunk decision is retained and $K-1$ alternatives are sampled from the same policy:

$$u_t^{(1)} = u_t^{\mathrm{trunk}}, \qquad u_t^{(k)} \sim \pi_{\mathrm{SFT}}(\cdot \mid x_t), \quad k = 2, \ldots, K.$$

Before each candidate, ρ restores the environment to s_t and the same context x_t is reused. Branches continue with π_{SFT} under the same decoding configuration until termination, producing sibling trajectories $\tau_t^{(k)}$ and terminal states $s_H^{(k)}$. Forks occur only along the trunk and are not recursively expanded. Each sibling receives the verified outcome

$$r_t^{(k)} = \mathrm{Verify}_W(q, s_0, \tau_t^{(k)}, s_H^{(k)}, \xi_{\mathcal{V}}) \in \{0, 1\}.$$

Because all candidates start from the same fork state and model-visible context, and are evaluated under the same greedy continuation policy and verifier, differences in $r_t^{(k)}$ capture differences in their verified downstream outcomes.

Preference-pair construction. When two distinct candidates obtain different verified outcomes, $r_t^+ > r_t^-$, we retain

$$(x_t, y^+, y^-), \qquad y^+ = u_t^+, \quad y^- = u_t^-.$$

The divergent assistant turns serve as preferred and rejected responses, while their completed continuations determine the labels and are retained with terminal states and verification records as construction provenance. The resulting pairs form $\mathcal{D}_{\mathrm{pref}}$. This procedure yields 1,150 pairs from EES-Single and EES-Cross scenarios after SFT, separate from the 30,395 SFT demonstrations but reusing the same restorable states and verifiers. Appendix D provides collection details and statistics.

4.3 DECISION-LEVEL PREFERENCE OPTIMIZATION

We initialize π_θ from π_{SFT} and use a frozen copy as the reference policy π_{ref}. Standard DPO (Rafailov et al., 2023) is applied to $\mathcal{D}_{\mathrm{pref}}$, with only the divergent preferred and rejected turns entering the objective; downstream continuations are used solely to determine their verified labels. The resulting checkpoint is Davinci-Tool-30B. Appendix E provides optimization details.

5 EXPERIMENTS

We evaluate verified-demonstration training, decision-level preference learning, transfer across model scales, comparison with public tool-use supervision, and contributions of the three data-construction routes. We further analyze supervision structure and resulting execution behavior.

5.1 EXPERIMENTAL SETUP

Benchmarks. We evaluate three held-out tool-use suites using their official protocols. τ^2-bench (Barres et al., 2025) evaluates stateful interaction in Retail and Telecom, and we report both domains and their macro average. BFCL v4 multi-turn (Patil et al., 2025) evaluates multi-turn function calling across four categories. MCP-Atlas (Bandi et al., 2026) evaluates heterogeneous MCP tool use; we use the 89 tasks without external API keys. Evaluation queries and states are excluded from data construction, and all scores are from our evaluations.

Models and training. Our primary model is Qwen3-Coder-30B-A3B-Instruct (Yang et al., 2025a); DeepSeek-V4-Pro (Xu et al., 2026) is used for trajectory generation. Davinci-Tool-30B-SFT is trained on the 30,395-trajectory Davinci-ToolData corpus, and Davinci-Tool-30B is obtained from this checkpoint using the decision-level preferences in Section 4.2.

Comparison settings. We compare with Qwen3-Coder-480B-A35B-Instruct (Yang et al., 2025a), Kimi-K2-Instruct (Team et al., 2025), MiniMax-M2.1 (MiniMax, 2025), GLM-4.7 (Z.ai, 2025), GPT-5.2 (OpenAI, 2025), and Claude-Sonnet-4.5 (Anthropic, 2025). We also fine-tune Qwen3-8B and Qwen3-14B (Yang et al., 2025a) on Davinci-ToolData, and compare against the public Toucan-1.5M release (Xu et al., 2025b) under the same Qwen3-Coder-30B backbone and SFT hyperparameters. MCP-Atlas is omitted for Qwen3-8B/14B because interactions exceed their default 40,960-token context window.

5.2 MAIN RESULTS

Table 2: Main results and model transfer. Reference models are strong open- and closed-source models included for external comparison. Underlining marks the best reference-model score in each metric, while bold marks the best score within each of the Qwen3-8B, Qwen3-14B, and primary 30B training groups. Light and dark shading denote Davinci-ToolData SFT and SFT+DPO, respectively.

Model	Training	τ^2-bench			BFCL v4 multi-turn					MCP-Atlas
		Retail	Telecom	Avg.	Base	Long-Context	Miss-Func	Miss-Param	Avg.	Avg.
Reference models										
Qwen3-Coder-480B-A35B-Instruct	—	77.5	39.5	58.5	55.5	54.5	32.0	32.0	43.5	28.1
Kimi-K2-Instruct	—	70.6	65.8	68.2	42.0	38.5	23.0	30.0	33.4	59.6
MiniMax-M2.1	—	81.6	87.0	84.3	76.5	69.0	71.0	69.0	71.4	59.6
GLM-4.7	—	79.8	98.2	89.0	69.5	67.0	66.0	59.0	65.4	55.1
GPT-5.2	—	81.6	98.2	89.9	37.0	27.0	22.5	29.0	28.9	61.8
Claude-Sonnet-4.5	—	86.0	98.0	92.0	55.5	54.5	60.0	51.0	55.3	65.2
Transfer to smaller models										
Qwen3-8B	Base	47.4	17.5	32.5	22.5	16.0	6.5	16.5	15.4	—
	Davinci-ToolData SFT	**64.9**	**84.2**	**74.6**	**50.5**	**23.5**	**29.0**	**32.5**	**33.9**	—
Qwen3-14B	Base	50.0	16.7	33.4	35.0	23.0	5.5	21.0	21.1	—
	Davinci-ToolData SFT	**67.5**	**86.0**	**76.8**	**50.0**	**34.0**	**42.5**	**39.0**	**41.4**	—
Primary backbone										
Qwen3-Coder-30B-A3B-Instruct	Base	68.7	36.0	52.4	43.0	36.0	17.0	25.5	30.4	16.9
	Toucan-1.5M SFT	65.8	89.5	77.7	52.5	46.0	35.0	34.0	41.9	22.5
	Davinci-ToolData SFT	73.7	**96.5**	85.1	71.5	61.0	**63.0**	47.0	60.6	52.8
Davinci-Tool-30B	Davinci-ToolData SFT + DPO	**79.8**	**96.5**	**88.2**	**72.5**	**62.0**	61.5	**55.0**	**62.8**	**56.2**

Training-stage gains. Verified-demonstration SFT establishes most of the improvement on the primary 30B backbone, raising τ^2-bench, BFCL v4, and MCP-Atlas from $52.4/30.4/16.9$ to $85.1/60.6/52.8$. Gains occur in both τ^2 domains and all four BFCL categories. Decision-level DPO adds 3.1, 2.2, and 3.4 points on the three aggregates. The changes are category-dependent: τ^2 Retail improves by 6.1 points and BFCL Missing-Parameter by 8.0, while Missing-Function decreases by 1.5. Thus, SFT provides the primary capability gains, while the aggregate DPO gains from 1,150 pairs support reusing the same restorable states and executable verifiers for post-SFT decision refinement.

Comparison with reference models. Davinci-Tool-30B exceeds Qwen3-Coder-480B-A35B-Instruct on all three aggregates despite using $1/16$ as many total parameters. On BFCL v4, its 62.8 score

exceeds Qwen3-Coder-480B, Kimi-K2-Instruct, GPT-5.2, and Claude-Sonnet-4.5, while remaining below MiniMax-M2.1 and GLM-4.7. On MCP-Atlas, its 56.2 exceeds Qwen3-Coder-480B and GLM-4.7 but remains below the other evaluated reference models.

Backbone transfer and data comparison. Davinci-ToolData improves the tested smaller models: Qwen3-8B rises from 32.5 to 74.6 on τ^2-bench and from 15.4 to 33.9 on BFCL v4, while Qwen3-14B rises from 33.4 to 76.8 and from 21.1 to 41.4, respectively. Both improve in every reported τ^2 and BFCL category, indicating that the supervision is not specific to the primary 30B model. Under the same Qwen3-Coder-30B backbone and SFT hyperparameters, Davinci-ToolData exceeds Toucan-1.5M by 7.4, 18.7, and 30.3 points on τ^2-bench, BFCL v4, and MCP-Atlas, respectively, despite using a substantially smaller training corpus. This matched comparison shows that corpus size alone does not explain the gains.

Execution-level analysis. Appendix C.4 compares paired base/SFT trajectories on recovered tasks and reveals benchmark-specific behavioral changes. On τ^2-bench, recovered tasks show substantially fewer explicit tool errors; on BFCL v4, trajectories cover more of the reference call chain and exhibit stronger output-to-argument dependencies; on MCP-Atlas, empty retrievals and malformed terminal outputs disappear despite increased tool interaction. These patterns show that the benchmark gains are associated with improved execution reliability, multi-step completeness, and result utilization rather than uniformly shorter or simpler trajectories.

5.3 Source-Composition Analysis

Table 3 compares six SFT mixtures that differ in both source composition and corpus size, so we use them to characterize source complementarity rather than size-controlled marginal effects. EGTS gives the strongest single-source BFCL v4 result (57.4), while EES-Cross is strongest on τ^2-bench (83.4) and MCP-Atlas (47.2). These distinct profiles support complementary roles for authored-system replay and synthesized-domain supervision.

Table 3: SFT data-source mixtures. Bold marks the strongest evaluated mixture on each benchmark.

SFT data			Benchmarks		
EGTS	EES-Single	EES-Cross	τ^2-bench	BFCL v4	MCP-Atlas
✓			78.9	57.4	28.1
	✓		82.0	54.5	29.2
		✓	83.4	46.6	47.2
	✓	✓	82.9	55.6	50.6
✓	✓		79.4	57.6	31.5
✓	✓	✓	**85.1**	**60.6**	**52.8**

Source combinations are not uniformly additive: some pairwise mixtures underperform a constituent single source. Nevertheless, adding EES-Cross to EGTS+EES-Single improves τ^2-bench, BFCL v4, and MCP-Atlas by 5.7, 3.0, and 21.3 points, respectively, while adding EGTS to EES-Single+EES-Cross yields gains of 2.2, 5.0, and 2.2 points. The full EGTS+EES-Single+EES-Cross mixture is strongest among all six configurations, exceeding the best single-source result by 1.7, 3.2, and 5.6 points on the three benchmarks, respectively.

Appendix C.3 shows benchmark-dependent source alignment: EGTS is most aligned with BFCL v4 and MCP-Atlas, while EES-Single is most aligned with τ^2-bench, supporting non-redundant source roles alongside Table 3. Appendix C.1.4 further shows that, after matching tool-call count and interaction length, EES-Cross retains domain transitions, cross-domain entity reuse, and read–write dependencies, indicating coordinated cross-domain structure beyond simply longer trajectories.

6 Conclusion

We presented Davinci-Tool and Davinci-ToolData, an execution-grounded framework for scaling verified supervision for agentic tool use. EGTS converts explored outcomes in independently authored systems into verified task-conditioned demonstrations, while EES constructs audited executable domains for single- and cross-domain supervision. Across three held-out benchmarks, verified-demonstration SFT provides the primary gains and transfers to the tested smaller models. Restorable EES states and executable verifiers are further reused for outcome-grounded decision refinement through DPO. Together, these results show how reusable execution and verification infrastructure can support tool-use training from demonstration construction to decision refinement.

AI USE STATEMENT

Generative AI tools were used as research assistants in this work. In particular, ChatGPT (OpenAI, 2026) and other generative AI systems were used to assist with manuscript drafting and language editing, synthetic-data generation, data analysis and interpretation, code debugging and optimization, and the implementation of experimental and system components. As described in the paper, generative models were also used within the data-construction pipeline, where generated artifacts were subject to the execution-based verification and auditing procedures specified in Sections 3–4 and the appendices. The core research ideas, methodological contributions, and pipeline and algorithm designs were developed by the authors. Generative AI was used to assist in implementing, analyzing, and refining these designs rather than to determine the central research contributions. The authors reviewed AI-assisted text, code, analyses, and generated artifacts, independently checked the resulting experimental claims and references, and take full responsibility for the final content of this work, including text, claims, code, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies agentic tool use in controlled executable environments and does not involve human-subject experiments. Some synthesized environments model potentially sensitive application domains, such as healthcare and finance; the reported results should therefore not be interpreted as validating deployment in safety-critical or production settings. We will follow applicable licensing, privacy, and redistribution requirements when releasing research artifacts. Real-world deployment of tool-using agents should employ appropriate authorization, access control, sandboxing, monitoring, and human oversight.

REPRODUCIBILITY STATEMENT

We provide detailed information for reproducing our data construction, training, and evaluation procedures throughout the paper and appendices. Appendix A specifies the shared execution, restoration, evidence, and verification interfaces; Appendix B details the EGTS and EES data-construction procedures; Appendix C reports environment coverage, corpus statistics, and analysis protocols; Appendix D describes the construction and statistics of the decision-level preference data; and Appendix E provides the SFT and DPO objectives, training configurations, context-length settings, and optimization details. Section 5 specifies the evaluation benchmarks, model settings, and comparison protocols. To support reproducibility and follow-up research, we will publicly release the Davinci-Tool model weights, the Davinci-ToolData training dataset, the training and evaluation code, and the executable data-generation engine, including the environment construction, execution, restoration, and verification components where redistribution permits.

REFERENCES

Ibrahim Abdelaziz, Asim Munawar, Kinjal Basu, Maxwell Crouse, Chulaka Gunasekara, Suneet Katrekar, and Pavan Kapanipathi. Synthesize and reward–reinforcement learning for multi-step tool use in live environments. *arXiv preprint arXiv:2606.03892*, 2026.

Anthropic. Introducing Claude Sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5, 2025.

Chaithanya Bandi, Razvan-Gabriel Dumitru, Ben Hertzberg, Divyansh Agarwal, Geobio Boo, Tejas Polakam, Sami Hassaan, Jeff Da, HiJae Kim, Vipul Gupta, et al. Mcp-atlas: A large-scale benchmark for tool-use competency with real mcp servers. *arXiv preprint arXiv:2602.00933*, 2026.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35: 16344–16359, 2022.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11143–11156, 2024.

Peter Jansen, Marc-Alexandre Côté, Tushar Khot, Erin Bransom, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, Oyvind Tafjord, and Peter Clark. Discoveryworld: A virtual environment for developing and evaluating automated scientific discovery agents. *Advances in neural information processing systems*, 37:10088–10116, 2024.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, et al. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. In *International Conference on Learning Representations*, volume 2026, pp. 53359–53394, 2026a.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 3102–3116, 2023.

Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yongkang Wu, Ji-Rong Wen, Yutao Zhu, and Zhicheng Dou. Webthinker: Empowering large reasoning models with deep research capability. *Advances in Neural Information Processing Systems*, 38:120091–120131, 2026b.

Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, et al. Toolace: Winning the points of llm function calling. In *International conference on learning representations*, volume 2025, pp. 41359–41381, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, 2025.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

MiniMax. MiniMax M2.1: Significantly enhanced multi-language programming, built for real-world complex tasks. https://www.minimax.io/news/minimax-m21, 2025.

OpenAI. Introducing GPT-5.2. https://openai.com/index/introducing-gpt-5-2/, 2025.

OpenAI. ChatGPT. https://chatgpt.com/, 2026. Generative AI assistant; accessed September 26, 2026.

Hafsa Ouajdi, Francesco Giannuzzo, Alaa Boukhary, Paolo Papotti, Gerard Conangla, and Adam Elwood. Execution-first synthetic tool-use trace generation for llm agents. *arXiv preprint arXiv:2607.29175*, 2026.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. In *International Conference on Learning Representations*, volume 2024, pp. 31932–31951, 2024.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199*, 2024.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: international conference for high performance computing, networking, storage and analysis*, pp. 1–16. IEEE, 2020.

Ram Ramrakhya, Andrew Szot, Omar Attia, Bogdan Mazoure, Anh Nguyen, Yuhao Yang, Zhe Gan, Harsh Agrawal, and Alexander Toshev. Scaling synthetic task generation for agents via exploration. In *International Conference on Learning Representations*, volume 2026, pp. 123630–123660, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, pp. 3135–3144. PMLR, 2017.

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. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8326–8357, 2026.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7584–7600, 2024.

Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Xiaoyu Tian, Haotian Wang, Shuaiting Chen, Hao Zhou, Kaichi Yu, Yudian Zhang, Jade Ouyang, Junxi Yin, Jiong Chen, Baoyan Guo, et al. Astra: Automated synthesis of agentic trajectories and reinforcement arenas. *arXiv preprint arXiv:2601.21558*, 2026.

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. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022.

Wenhao Wang, Peizhi Niu, Zhao Xu, Zhaoyu Chen, Jian Du, Yaxin Du, Xianghe Pang, Keduan Huang, Yanfeng Wang, Qiang Yan, et al. Mcp-flow: Facilitating llm agents to master real-world, diverse and scaling mcp tools. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5087–5116, 2026a.

Xiyao Wang, Linfeng Song, Ye Tian, Dian Yu, Baolin Peng, Haitao Mi, Furong Huang, and Dong Yu. Towards self-improvement of llms via mcts: Leveraging stepwise knowledge with curriculum preference learning. *arXiv preprint arXiv:2410.06508*, 2024.

Ziyi Wang, Yuxuan Lu, Yimeng Zhang, Pei Chen, Ziwei Dong, Jing Huang, Jiri Gesi, Xianfeng Tang, Chen Luo, Qun Liu, et al. Trajectory2task: Training robust tool-calling agents with synthesized yet verifiable data for complex user intents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 44021–44044, 2026b.

Yifan Wu, Yiran Peng, Yiyu Chen, Jianhao Ruan, Zijie Zhuang, Cheng Yang, Jiayi Zhang, Man Chen, Yenchi Tseng, Zhaoyang Yu, et al. Autowebworld: Synthesizing infinite verifiable web environments via finite state machines. *arXiv preprint arXiv:2602.14296*, 2026a.

Zijian Wu, Xiangyan Liu, Lingjun Chen, Fanqing Meng, Lingxiao Du, Yiran Zhao, Fanshi Zhang, Yaoqi Ye, Jiawei Wang, Zirui Wang, et al. Mcpmark: A benchmark for stress-testing realistic and comprehensive mcp use. In *International Conference on Learning Representations*, volume 2026, pp. 79852–79895, 2026b.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024a.

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. Monte carlo tree search boosts reasoning via iterative preference learning. *arXiv preprint arXiv:2405.00451*, 2024b.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *International Conference on Learning Representations*, volume 2025, pp. 79822–79843, 2025a.

Zhangchen Xu, Adriana Meza Soria, Shawn Tan, Anurag Roy, Ashish Sunil Agrawal, Radha Poovendran, and Rameswar Panda. Toucan: Synthesizing 1.5 m tool-agentic data from real-world mcp environments. *arXiv preprint arXiv:2510.01179*, 2025b.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Yongjin Yang, Sinjae Kang, Juyong Lee, Dongjun Lee, Se-Young Yun, and Kimin Lee. Automated skill discovery for language agents through exploration and iterative feedback. *arXiv preprint arXiv:2506.04287*, 2025b.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Z.ai. GLM-4.7. https://z.ai/blog/glm-4.7, 2025.

Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, et al. Agentevolver: Towards efficient self-evolving agent system. *arXiv preprint arXiv:2511.10395*, 2025.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 414–431, 2025.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

A Shared Execution and Verification Interface

This appendix specifies the shared execution, restoration, and verification interface used by all trajectory-construction routes.

A.1 Environment Interface and Evidence Records

A resettable tool environment is represented as $W = (\mathcal{S}, \mathcal{T}, \rho, P, \mathcal{E})$, where $\mathcal{S}$ is the state space, $\mathcal{T}$ the tool set, ρ the restoration operation, P the tool-conditioned transition function, and $\mathcal{E}$ the observable evidence space. For a task q, a trajectory $\tau = (a_1, o_1, \ldots, a_H, o_H)$ maps an initial state s_0 to a terminal state s_H. Its evidence record

$$R = (q, s_0, \tau, s_H, \xi, \nu)$$

retains outcome evidence $\xi \in \mathcal{E}$ together with verification and provenance metadata ν.

Evidence capture follows the environment type. Mutation-oriented environments retain state snapshots, retrieval-oriented environments retain grounded answer evidence, and synthesized EES scenarios evaluate executable task predicates. The shared interface supports repeated execution, state restoration, and outcome-evidence capture as required by each construction route. EES scenarios additionally retain scenario-specific executable verifiers that are reused for preference construction in Appendix D.

A.2 Environment Wrapping and Evidence Capture

Executable environments are adapted to a common interaction interface exposing callable tools, a reset procedure, outcome-evidence capture, and collection metadata. Depending on the underlying system, restoration may use a container lifecycle, filesystem image, database snapshot, service endpoint, or another source-specific mechanism.

Before each collection episode, the harness records the initial state s_0. During execution, it records ordered actions, observations, and execution status:

$$\tau = \big[(a_1, o_1, z_1), \ldots, (a_H, o_H, z_H)\big],$$

where z_t indicates whether the corresponding tool invocation completed successfully. After execution, the harness records the resulting state s_H and observable outcome evidence ξ.

For state-modifying environments, source-specific snapshot adapters map raw states to canonical representations. Volatile fields such as timestamps, transient identifiers, and execution metadata are removed when they do not affect task semantics, enabling consistent comparison across repeated executions. Retrieval-oriented environments instead retain grounded information returned by successful read operations as answer evidence. EGTS uses these records for task synthesis and replay verification, while EES combines similar state and evidence records with executable task predicates.

B Trajectory Construction Details

This section provides implementation details for EGTS, EES-Single, and EES-Cross under the shared interface.

B.1 Execution-Grounded Task Synthesis

B.1.1 Progressive and Experience-Guided Exploration

Exploration objectives. Exploration is organized around local executable objectives derived from the currently exposed tools and observable environment structure. These objectives guide the collector toward feasible interaction patterns, while coverage statistics prioritize under-explored tools and behaviors to broaden the outcomes available for task synthesis.

Progressive action-space expansion. EGTS progressively expands the action space available to the explorer. Exploration begins with orientation tools that reveal environment structure, proceeds to task-bearing anchor tools, and then expands to the broader tool inventory. As coverage accumulates, under-used tools receive higher sampling priority to encourage new interaction patterns.

Two additional sources of guidance support exploration. Skill guides provide reusable procedural knowledge for common workflows, while an experience store retrieves relevant execution traces from previous episodes. Together, they transfer useful execution knowledge across exploration objectives and support more involved interactions (Yang et al., 2025b; Ramrakhya et al., 2026).

Collection behavior adapts to interaction difficulty. Simpler stages use strict execution, where an unsuccessful call terminates the episode, while harder stages permit bounded retry and rollback. If execution ends with an unsuccessful tail, the collector retains the longest executable prefix ending at a successful call and removes unmatched or failure-only messages.

B.1.2 Task Candidate Construction

For mutation-oriented exploratory trajectories, task synthesis conditions on the retained calls, affected entities, and observed transition from s_0 to s_H. For retrieval-oriented trajectories, it conditions on successful reads and their grounded answer evidence. The synthesizer generates outcome-oriented tasks supported by the recorded execution (Wang et al., 2026b; Ouajdi et al., 2026).

Multiple task candidates may be generated from one exploratory interaction. Candidate selection considers specificity, naturalness, and consistency with the recorded evidence, retaining requests whose target outcomes can be traced to the corresponding calls and observations.

B.1.3 Execution-Derived Replay Guidance

The replay harness associates the observed outcome with the calls that contributed to it and extracts task-relevant execution information from the exploratory episode. This yields a replay guide

$$g(\tau_{\text{exp}}, \xi_{\text{exp}})$$

used during replay:

$$s_0 \xrightarrow[\pi_{\text{rep}}]{q,\, g(\tau_{\text{exp}}, \xi_{\text{exp}})} \tau_{\text{rep}}, s'_H.$$

The guide transfers grounded procedural information from exploration to task-conditioned replay (Xu et al., 2025a). Unlike an exploratory trace, the replay trajectory is generated after the user request has been fixed and is therefore organized around fulfilling that request.

Replay is evaluated as part of demonstration construction. Its acceptance records whether the synthesized request recovers the target outcome under the replay harness.

The EGTS SFT partition contains 13,824 accepted task-conditioned replay demonstrations. Each serialized example contains `messages` and `tools` and uses the standard task-solving system prompt. Replay guidance is privileged collection context and is not serialized as a separate training field; the collection prompt is replaced during SFT serialization. The resulting training example retains the task-conditioned interaction rather than the exploration-oriented collection context.

B.1.4 Replay Verification and Trajectory Audit

For mutation tasks, replay succeeds when the canonical terminal state matches the outcome recorded during exploration:

$$\phi_W(s'_H) = \phi_W(s_H).$$

Retrieval tasks are evaluated against the grounded answer evidence collected during exploration.

After outcome verification, additional checks determine whether the replayed trajectory is suitable for training. Structural validation enforces valid message–tool-call pairing, executable tool invocation, final-response presence, and failure constraints. A hard judge checks consistency among the task, tool names, arguments, observations, action ordering, and final evidence, while a soft judge evaluates trajectory coherence and response quality.

Accepted samples retain the task, reset provenance, execution snapshots, replay trace, task-candidate records, harness metadata, and verification results, linking demonstrations to acceptance decisions.

B.2 Executable Environment Synthesis

This section expands the EES procedures introduced in Sections 3.2.1 and 3.2.2, covering single-domain synthesis and auditing, state-grounded scenario construction, executable domain composition, and cross-domain trajectory verification.

B.2.1 Single-Domain Synthesis and Audit

Staged contract synthesis. For each target domain, EES first mines representative backend concepts, entity structures, operational constraints, and interaction patterns. After deduplication, staged synthesis converts this information into typed schemas, executable scaffolds, operation inventories, and concrete tool implementations, producing a runnable domain with explicit state, transition, and rule structure (Song et al., 2026).

The resulting domain contract specifies entity types, state structure, exposed operations, transition behavior, and domain rules. This contract provides shared semantics for tool execution, scenario construction, and verification.

Executor–auditor protocol. Each synthesized domain is exercised before scenario construction. The executor invokes its operations over initialized states, covering both state-reading and state-mutating behavior. The auditor checks tool arguments, returned values, exceptions, resulting state transitions, and rule compliance against the domain contract.

Detected inconsistencies trigger revisions to the corresponding schema, implementation, or domain specification before another audit round. Audited domains are used for scenario construction.

B.2.2 State-Grounded Task and Verifier Construction

Scenario initialization and task construction. For each audited domain, EES instantiates an initial state s_0 consistent with its entity schemas and domain rules. Task generation conditions on this state, the exposed operations, and applicable constraints to construct achievable interaction requirements. These requirements are decomposed into atomic conditions represented by executable predicates.

Each resulting scenario has the form

$$(s_0, q, \mathcal{V}),$$

where q is the state-grounded user task and

$$\mathcal{V} = \{v_1, \ldots, v_J\}$$

contains executable predicates encoding its atomic requirements.

For state-modifying tasks, state-based target predicates are first evaluated on the initial state s_0. A target already satisfied in s_0 is regenerated or revised so that completing the task requires an interaction-induced state change. Candidate trajectories are then collected by executing q from s_0, producing an execution trace τ and terminal state s_H. The task requirements are evaluated on the

resulting execution and state:

$$\text{Verify}(q, \tau, s_H) = \prod_{j=1}^{J} \mathbb{I}\left[v_j(\tau, s_H) = 1\right].$$

The initial state, task requirements, execution trace, terminal state, predicate outcomes, and verification record form the executable evidence retained with each verified EES-Single trajectory.

B.2.3 Cross-Domain Composition

Affinity estimation and subgraph sampling. For each audited domain, embedding retrieval proposes semantically related candidates. An affinity rubric then evaluates their functional compatibility and possible relationships between their entity and state spaces. These relations define an affinity graph; bounded random walks sample connected domain subsets for composition (Li et al., 2026a).

Namespace injection and executable merging. For each selected domain d_i, namespace injection π_i transforms tool names, entity types, identifiers, and state attributes before composition. The merged environment state is

$$s^{\text{cross}} = \big(\pi_1(s_1), \ldots, \pi_m(s_m)\big),$$

and the exposed tool set is formed from the corresponding namespaced interfaces.

Static validation checks initialization, state structure, tool-schema well-formedness, and namespace consistency before the merged runtime proceeds to shared world-context construction.

Shared world-context construction and audit. EES-Cross constructs a shared world context over the merged domains. The context specifies user roles, shared entities, linking fields, cross-domain policies, and coherence constraints that establish relationships among the participating domain states.

Schema and consistency checks validate referenced entities, attributes, and interfaces against the merged runtime. A consistency judge further evaluates the generated roles, policies, linking relations, and shared entities. Accepted world contexts are then used for cross-domain task construction.

B.2.4 Cross-Domain Task Construction and Verification

Merged environments are initialized with records whose identifiers and linking fields establish dependencies across participating domains. Cross-domain tasks are conditioned on the shared world context, user role, initialized state, and applicable policies, with required operations and success criteria spanning multiple systems.

Candidate trajectories first pass runtime and tool-execution checks. Task-level verification then evaluates the observed calls, state changes, terminal outcome, and applicable policy constraints against the scenario requirements. Trajectories satisfying these requirements are retained.

Accepted trajectories retain the initial and terminal states, shared world context, cross-domain execution trace, outcome evidence, verification records, transferred-entity types, and dependency structure together with their construction and verification provenance.

C Environment and Data Analysis

C.1 Environment Sources and Coverage

This section characterizes the authored environment families used by EGTS and the domain taxonomy and composition patterns used by EES. Trajectory-level statistics for the resulting SFT corpus are reported in Section 3.3 and Appendix C.2.

C.1.1 EGTS Environment Sources

EGTS operates over independently authored environments that support multi-step tool interaction, repeatable initialization, observable state or answer evidence, and stable execution. Table 4 summarizes the represented environment families and their interaction scope.

Table 4: Environment families represented in the EGTS library. Each source is adapted to the common execution and evidence-capture interface.

Environment family	Representative interactions	Representative adapted sources
Personal and business services	Contacts, messages, reminders, calendars, banking, travel, health, and other structured service operations	ToolSandbox (Lu et al., 2025), API-Bank (Li et al., 2023)
API discovery and orchestration	API search, schema inspection, argument grounding, selection, and multi-API composition	StableToolBench (Guo et al., 2024)
Developer and data workspaces	Filesystem operations, repositories, relational data, and browser-backed developer workflows	MCPMark (Wu et al., 2026b)
Terminal and professional production	Shell and text processing, system operations, reports, office files, PDFs, and data deliverables	Terminal-Bench (Merrill et al., 2026)
Web interaction and commerce	Page navigation, form interaction, product search, constraint matching, and shopping workflows	MiniWoB (Shi et al., 2017), WebShop (Yao et al., 2022)
Scientific simulation	Procedural experiments, hypothesis testing, embodied discovery, and stateful scientific interaction	ScienceWorld (Wang et al., 2022), DiscoveryWorld (Jansen et al., 2024)
Integrated multi-tool workflows	Workspace, browser, research, document, communication, commerce, and infrastructure tools within a shared environment	Toolathlon (Li et al., 2026a)
Application ecosystems	Stateful workflows spanning multiple simulated applications and user records	AppWorld (Trivedi et al., 2024)

The collector synthesizes new tasks from explored outcomes while retaining environment identifiers, source versions, reset provenance, and task-level metadata. Evaluation queries and states from BFCL v4, τ^2-bench, and MCP-Atlas are excluded from trajectory construction.

Scope of the source inventory. Table 4 describes the authored-system library, whereas Appendix C.2 characterizes the 13,824 accepted EGTS demonstrations used for SFT. Environment-family counts and trajectory counts therefore describe different levels of the construction pipeline.

C.1.2 EES Domain Taxonomy and Usage

The EES catalog contains 191 executable domain definitions organized into 19 application categories, ranging from healthcare and software infrastructure to commerce, education, and document management. A domain is a stateful executable component; a category groups domains by application area. The retained EES-Cross inventory contains 246 compositions using 188 of the 191 domains, with 706 constituent-domain slots in total.

Table 5 reports both the number of domain definitions in each category and their frequency in the retained cross-domain compositions. A domain slot is one occurrence of a constituent domain within one composed environment.

Category coverage. Healthcare and Clinical accounts for 115 of the 706 constituent-domain slots (16.3%), followed by IT Infrastructure and Software Engineering with 80 slots (11.3%) and Social and Communication with 56 (7.9%). The remaining slots are distributed across all 16 other application categories, showing that the retained compositions span the full EES taxonomy rather than concentrating on a small number of domains.

C.1.3 EES Composition Patterns and Category Affinities

The retained EES-Cross inventory contains 77 two-domain, 124 three-domain, and 45 four-domain compositions, totaling 246 compositions and 706 constituent-domain slots. The mean composition size is 2.87 domains. The category-count distribution also totals 246 compositions: 129 contain domains from one application category, while 117 span multiple categories. Domain count and category count capture different aspects of composition because distinct executable systems can

Table 5: EES domain categories and composition usage. D_c is the number of domain definitions in category c, and S_c counts their occurrences across the 246 retained EES-Cross compositions. Rows follow decreasing S_c. Category codes are used in Figure 3.

Code	Application category	D_c	S_c
HC	Healthcare & Clinical	30	115
IT	IT Infrastructure & Software Engineering	20	80
SOC	Social & Communication	14	56
ECOM	E-commerce, Retail & Marketplace	11	47
ENT	Enterprise, HR & Business Ops	12	47
PHW	Personal Health & Wellness	13	47
PROD	Productivity & Personal Info Mgmt	8	45
FIN	Finance, Banking & Billing	13	40
MEDIA	Media & Entertainment	10	39
TRAVEL	Travel, Hospitality & Local Services	8	30
SPORT	Sports & Fitness	11	27
DATA	Data, Analytics & ML	6	24
IOT	IoT, Industrial & Environmental Monitoring	8	22
MKT	Marketing & Advertising	4	20
DOC	Documents & Knowledge Mgmt	4	16
INS	Insurance & Claims	5	16
EDU	Education	4	13
GOV	Government & Public Sector	5	13
MOB	Transportation, Mobility & Geo	5	9
	Total	191	706

belong to the same application category while retaining separate states and tool interfaces. Figure 3 in Section 3.3 visualizes these distributions and the most frequent cross-category affinities.

Healthcare and Clinical with Personal Health and Wellness is the most frequent category pair, appearing in 17 compositions, followed by Data, Analytics and ML with IT Infrastructure and Software Engineering in 15. Other recurring combinations include E-commerce with Travel and Enterprise operations with Productivity, each appearing in six compositions. These patterns illustrate both within-category system composition and workflows connecting distinct application areas.

C.1.4 Structural Analysis of Cross-Domain Supervision

Beyond composition statistics, we further analyze whether EES-Cross introduces execution structures that differ from single-domain trajectories. The goal is to distinguish cross-domain supervision from simply longer interaction sequences. We compare EES-Single and EES-Cross trajectories under length-controlled matching and measure structural properties related to cross-system coordination.

For each trajectory, we represent the executed tool sequence as an ordered chain of tool invocations. We identify domain transitions when two consecutive tool calls originate from different executable domains. Given a trajectory $\tau = (a_1, \ldots, a_H)$ with domain assignment $d(a_t)$ for each tool call, the domain-switch ratio is defined as

$$R_{\text{switch}}(\tau) = \frac{1}{H-1} \sum_{t=1}^{H-1} \mathbb{I}[d(a_t) \neq d(a_{t+1})].$$

To measure information transfer across domains, we additionally identify entities produced by one domain and consumed by another domain. For an entity e, let $p(e)$ and $c(e)$ denote the producing and consuming tool calls, respectively. We count cross-domain entity reuse when $d(p(e)) \neq d(c(e))$. We further identify cross-domain read–write patterns where a value retrieved from one domain contributes to a later state-modifying operation in another domain.

Because EES-Cross trajectories may naturally contain more involved interactions, we perform a length-controlled comparison. For each EES-Cross trajectory, we select EES-Single trajectories with matched tool-call counts and similar serialized interaction lengths, and compute the structural statistics on the matched subsets. Table 6 reports the resulting comparison.

Table 6: Structural comparison between EES-Single and EES-Cross trajectories under length-controlled matching. Cross-domain metrics measure execution coordination beyond interaction length.

Metric	EES-Single	EES-Cross
Domain transitions (%)	0.00	14.50
Cross-domain entity reuse (%)	0.00	5.23
Trajectories with cross-domain transfer (%)	0.00	6.39
Cross-domain read→write (%)	0.00	0.98
Tool calls (matched)	100.0	100.0
Interaction length (relative)	100.0	99.1

The comparison shows that EES-Cross introduces execution patterns absent from EES-Single, including domain transitions, cross-domain entity transfer, and read–write dependencies across systems. These differences remain after controlling for tool-call count and interaction length, indicating that the additional supervision arises from coordinated execution structures rather than simply longer trajectories. The analysis does not imply that every dimension of structural depth increases; instead, EES-Cross changes the dependency topology of tool-use workflows by introducing interactions among otherwise independent executable domains.

C.1.5 Recorded Composition: Inventory, Promotions, and Orders

The composition manifest identifies `cross_env_0020` as a merge of `env_9_sft` (product inventory), `env_76_sft` (product-and-promotion management), and `env_172_rl` (order management). Table 7 reports selected operations and rules from their domain definitions.

Table 7: Selected interfaces in the recorded composition `cross_env_0020`. Names are shown before namespace injection.

Domain	Selected operations	Declared requirements
Inventory	`get_product_by_id`; `update_product_stock`	Non-negative prices; predefined product categories.
Promotions	`get_product_by_sku`; `get_applicable_promotions`; `evaluate_customer_eligibility`	Active time window, product–promotion link, and customer eligibility determine applicability.
Orders	`get_product_by_id`; `add_product_to_cart`	Stock must cover the added quantity; each cart belongs to one user; repeated products use an aggregated quantity.

The example illustrates two requirements handled separately by EES-Cross. The duplicated `get_product_by_id` interface motivates namespace separation, while the use of product IDs and SKUs across systems requires semantic links between their underlying records. The shared-context and scenario-construction stages establish these relations before cross-domain task generation.

C.1.6 Serialized Cross-Domain Demonstration

One EES-Cross training example asks the agent to consult the "Cryo-EM Vendor Call" note, verify ownership of an existing reminder, update its message and time, and report the user's notification preference. The request names the target note and reminder, while execution retrieves their contents and resolves the corresponding user identity before performing the updates.

The example makes the cross-system dependency concrete. Information retrieved from the personal-note interface contributes to a reminder update, while identity resolution provides the user identifier needed to verify reminder ownership and retrieve notification settings.

C.2 Detailed Corpus Statistics

Davinci-ToolData contains 30,395 SFT trajectories: 13,824 from EGTS, 8,026 from EES-Single, and 8,545 from EES-Cross. Each stored conversation is counted as one task trajectory. Distinct requests

Table 8: Model-visible tool evidence from one serialized two-domain EES-Cross demonstration. Construction-level states and verification records are retained separately from the conversation.

Step	Operation	Relevant observation or update
1	`mobile_get_user_by_name`	Resolves Maya Chen to `usr-001`; notification preference is push.
2	`personal_get_note_by_id`	Note content: compare service quotes and ask BioScope about January slots.
3	`mobile_get_reminder_by_id`	`rem-001` belongs to `usr-001`, with the existing message and time.
4	`mobile_update_reminder_message`	Sets "Call BioScope about January cryo-EM slots"; tool reports success.
5	`mobile_update_reminder_time`	Sets 2025-01-17 at 10:00, UTC−05:00; tool reports success.
6	`mobile_get_user_notification_preference`	Queries `usr-001`; result is push.

are identified from the complete sequence of user messages after Unicode NFKC and whitespace normalization; executable scenarios additionally depend on their initial states. The three source partitions concatenate to the full corpus with no duplicate canonical JSON records. As sets of declared tool names, inventories characterize serialized configurations rather than trajectory counts.

Table 9: Coverage of the SFT corpus. "Requests" counts distinct normalized user-message sequences; "Inventories" counts distinct declared-tool configurations. Declared and invoked tools are counted by their serialized names.

Source	Trajectories	Requests	Inventories	Declared tools	Invoked tools
EGTS	13,824	13,283	117	387	350
EES-Single	8,026	7,550	234	4,042	1,610
EES-Cross	8,545	8,319	276	4,316	1,683
Full SFT corpus	30,395	28,985	436	4,672	2,260

The 4,672 declared tool names describe the union of serialized tool schemas exposed across the corpus, whereas 2,260 distinct serialized tool names are actually invoked in the retained trajectories. The latter is the tool-coverage statistic reported in the main text.

Length and interaction definitions. Token counts use the SFT checkpoint's tokenizer and chat template, including the system prompt, tool definitions, and complete conversation. JSON-encoded arguments and schemas are parsed before template rendering. A tool call is one explicit assistant tool invocation, so parallel calls count separately. An assistant turn is one assistant message, including messages without tool calls.

Table 10: Length and interaction statistics of the SFT corpus. Token lengths describe complete serialized examples before training-time length handling.

	Tokens			Tool calls			Assistant turns		
Source	Mean	Min.	Max.	Mean	Min.	Max.	Mean	Min.	Max.
EGTS	10,063.9	377	768,427	6.04	0	60	9.80	1	45
EES-Single	10,372.7	2,030	369,175	5.87	0	76	10.42	1	33
EES-Cross	20,444.7	1,746	1,259,991	7.01	0	99	7.57	1	34
Full SFT corpus	13,063.8	377	1,259,991	6.26	0	99	9.34	1	45

Length distribution. The full SFT corpus contains 190,410 explicit tool calls and averages 4.05 distinct invoked tools per trajectory. Its median serialized length is 9,284 tokens and its 95th percentile is 20,237 tokens. A total of 957 trajectories (3.15%) exceed 32,768 serialized tokens before training-time length handling. EES-Cross has a median length of 10,709 tokens and a mean of 20,444.7,

reflecting a longer upper tail than the other two sources. During SFT, examples exceeding the 32,768-token maximum sequence length are truncated by the default training pipeline. The statistics in Table 10 describe the complete serialized examples before this truncation. The SFT training configuration is reported in Appendix E.1.

C.3 Representation Alignment with Evaluation Trajectories

To examine how the three construction routes relate to the execution patterns required by held-out evaluations, we compare the complete verified-trajectory files used for this analysis with benchmark-side execution traces. Each synthetic and benchmark trajectory is normalized into three views: the task intent, the available tool capabilities, and the executed action chain. Instance-specific identifiers, entity names, paths, and values are replaced with typed placeholders. We use the executed-action view for the primary comparison and retain the intent and tool views as auxiliary checks. On the benchmark side, we use all official BFCL v4 multi-turn reference chains and all fully successful trajectories in the supplied τ^2-bench and MCP-Atlas evaluation records.

Let D_s denote the trajectories from source s and let $e(x)$ be the Qwen3-Embedding-8B representation of normalized trajectory x. For benchmark trajectory b, we measure source alignment by the mean cosine similarity of its K nearest source trajectories:

$$A_s(b) = \frac{1}{K} \sum_{x \in \mathrm{NN}_K(b; D_s)} \frac{e(b)^\top e(x)}{\|e(b)\|_2 \|e(x)\|_2}, \qquad K = 5.$$

All sources use the same normalizer, encoder, and pooling procedure. To prevent a larger source from receiving an artificial nearest-neighbor advantage, we deduplicate exact representations and sample equally sized candidate pools from EGTS, EES-Single, and EES-Cross over ten repeats. Table 11 reports the resulting mean Top-5 alignment. This metric characterizes relative proximity in the representation space rather than absolute task coverage or execution correctness.

Table 11: Representation alignment between verified synthesis trajectories and benchmark reference or successful execution trajectories. Values are mean Top-5 cosine similarities under equal candidate budgets; bold marks the strongest source for each benchmark.

Source	BFCL v4	τ^2-bench	MCP-Atlas
EGTS	**0.591**	0.665	**0.680**
EES-Single	0.547	**0.695**	0.567
EES-Cross	0.575	0.670	0.589

The source profiles are benchmark dependent. EGTS is most closely aligned with BFCL v4 and the supplied MCP-Atlas trajectories, whereas EES-Single is most closely aligned with τ^2-bench. EES-Cross is not the strongest source in aggregate, but provides additional high-ranked neighbors on BFCL v4 and τ^2-bench. An equal-budget leave-one-source-out comparison gives the same pattern: removing EGTS reduces alignment on BFCL v4 and MCP-Atlas, removing EES-Single reduces alignment on τ^2-bench, and removing EES-Cross produces a smaller reduction on BFCL v4. These results indicate that the construction routes contribute different execution-pattern neighborhoods. Together with the source-mixture results in Table 3, they support the interpretation that the full mixture combines non-redundant supervision; downstream training benefits are evaluated by the source-mixture experiments rather than by embedding similarity alone.

C.4 Execution-Level Analysis of Benchmark Gains

We further examine whether the benchmark gains after SFT are accompanied by changes in actual tool execution. For each benchmark, we align the base-model and Davinci-Tool-30B-SFT trajectories by task identity and retain the original benchmark success criterion. Let M_b denote the matched task set for benchmark b, and let $y_i^{\mathrm{base}}, y_i^{\mathrm{sft}} \in \{0, 1\}$ denote the outcomes of task i before and after SFT. We summarize task-level transitions using

$$R_{u \to v}(b) = \frac{1}{|M_b|} \sum_{i \in M_b} \mathbb{I}\left[y_i^{\mathrm{base}} = u,\ y_i^{\mathrm{sft}} = v\right],$$

where $u, v \in \{0, 1\}$. Table 12 reports the resulting transition counts. Across 1,116 matched tasks, 441 change from failure to success (F→S), while 78 change from success to failure (S→F).

Table 12: Paired outcome transitions between the base model and Davinci-Tool-30B-SFT.

Benchmark	Matched	F→S	S→F	S→S	F→F
τ^2 Retail	114	23	10	68	13
τ^2 Telecom	113	74	4	35	0
BFCL v4	800	304	58	181	257
MCP-Atlas	89	40	6	9	34

Execution statistics. We analyze the F→S trajectories to characterize how recovered tasks change at the execution level. Because the benchmarks expose different tool interfaces and success conditions, we use benchmark-specific observable statistics rather than forcing a single execution-quality metric across all tasks. For τ^2-bench, we measure the explicit tool-error rate

$$\mathrm{ErrRate}(\tau) = \frac{N_{\mathrm{err}}(\tau)}{N_{\mathrm{call}}(\tau)},$$

where N_{err} and N_{call} are the numbers of explicit tool errors and tool calls in trajectory τ.

For BFCL v4, we additionally measure how completely the executed trajectory covers the normalized reference-call chain. Let C_i^{ref} denote the normalized reference call signatures for task i. We define

$$\mathrm{Cov}(\tau_i) = \frac{N_{\mathrm{match}}(\tau_i, C_i^{\mathrm{ref}})}{|C_i^{\mathrm{ref}}|},$$

where N_{match} counts executed calls matched to the normalized reference signatures. We further construct conservative output-to-argument dependencies when a value returned by an earlier successful call is reused in a later call argument, excluding values already present in the user request, input echoes, and trivial constants. From this graph we report the number of visible dependency edges and its longest visible chain. BFCL behavior statistics use the 274 recovered tasks with executable calls available on both sides; missing execution traces are left unknown rather than treated as zero.

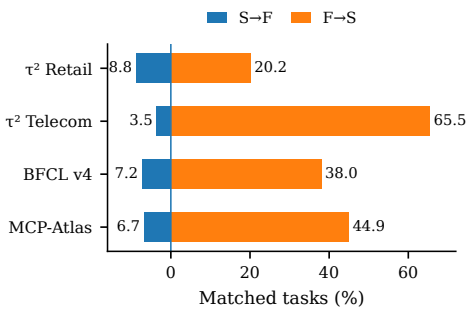
For MCP-Atlas, we use directly observable execution-profile statistics, including empty retrieval responses, protocol-fragment endings, calls per task, and explicit tool-error rates. These measurements characterize the executed trajectory and are not used as alternative benchmark scores.

Observed behavior changes. Figure 4 shows that the performance gains are accompanied by concrete but different changes in execution behavior. On τ^2-bench, recovery is associated with substantially more reliable tool execution; Retail additionally recovers the expected database state on 18 of 23 F→S tasks, while the 18 observed Telecom identity-placeholder calls disappear after SFT. BFCL v4 exhibits a different pattern: recovered trajectories execute more of the reference chain and expose stronger output-to-argument dependencies, even though the average number of calls increases from 7.62 to 9.57. MCP-Atlas likewise does not show a general reduction in execution cost; instead, recovered trajectories avoid observed dead-end retrieval loops and malformed terminal outputs while performing more tool interaction. These results complement the aggregate benchmark improvements by showing that Davinci-ToolData is associated with observable gains in execution reliability, multi-step completeness, and result utilization rather than a uniform reduction in trajectory length or tool use.

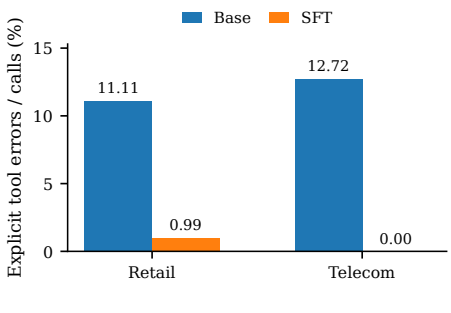
This analysis is descriptive rather than a separate causal estimate: each model contributes one recorded rollout per task, and F→S follows the original benchmark scoring rule, which can include partially correct outcomes.

D Same-Prefix Preference Construction Details

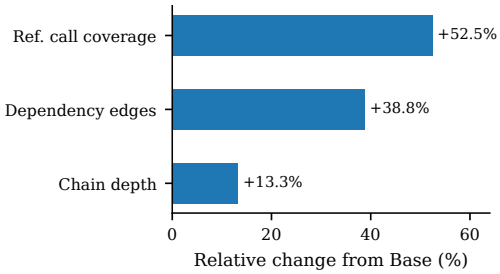
This appendix expands the preference-construction procedure introduced in Section 4.2. Preference pairs are constructed after SFT from restorable EES-Single and EES-Cross scenarios using the SFT checkpoint for candidate generation and continuation rollout. We report the collection settings used for the primary preference corpus; engineering-only execution settings and exploratory collection variants are omitted.



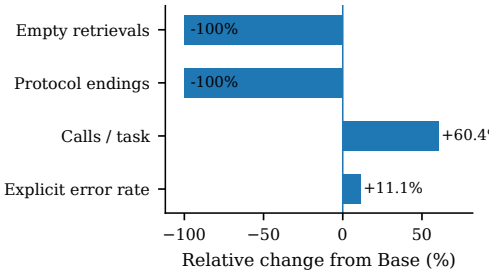
(a) Outcome transitions.



(b) τ^2 reliability.



(c) BFCL completeness.



(d) MCP execution profile.

Figure 4: Paired execution changes associated with benchmark recovery after SFT. (a) shows the fractions of matched tasks changing from failure to success and from success to failure. (b) shows that, among recovered τ^2-bench tasks, explicit tool-error rates decrease from 11.11% to 0.99% on Retail and from 12.72% to 0 on Telecom. (c) shows that recovered BFCL v4 trajectories become more complete: normalized reference-call coverage increases from 56.41% to 86.01%, visible output-to-argument dependency edges from 1.21 to 1.68, and visible chain depth from 1.58 to 1.79. (d) shows a different pattern on MCP-Atlas: observed empty retrievals decrease from 37 to 0 and protocol-fragment endings from 7 to 0, while calls per task increase from 5.55 to 8.90 and the explicit tool-error rate changes from 14.41% to 16.01%. The paired analysis therefore indicates benchmark-specific execution changes rather than a single mechanism based on shorter trajectories or uniformly fewer tool errors.

D.1 Fork-State Collection

For each selected EES-Single or EES-Cross scenario, the SFT policy first generates an interaction rollout, referred to as the trunk. Immediately before each assistant decision, the collector records the restorable environment state together with the interaction prefix and visible tool schemas. We retain at most $B = 10$ recorded decision points along each trunk as candidate fork states. The trunk is generated greedily with temperature 0, providing a deterministic reference path for sibling construction.

A recorded fork is

$$h_t = (q, s_t, m_{<t}, \mathcal{T}_t),$$

where q is the user task, s_t is the executable state immediately before the decision, $m_{<t}$ is the shared model-visible interaction history, and $\mathcal{T}_t$ contains the tools exposed at that point.

D.2 Sibling Generation and Rollout

At each selected fork, the assistant turn from the trunk is retained as one candidate and $K - 1$ alternative decisions are sampled from the same SFT policy. For the preference data used in the main

experiments, we set $K = 3$, yielding one greedy trunk decision and two sampled sibling decisions per fork. Stochasticity is introduced only at the fork decision: sibling decisions use temperature 1.0, while the trunk decision and all subsequent continuations use temperature 0 (greedy decoding). Branch samples use deterministic branch-specific seeds derived from a base seed of 1000.

Before executing each candidate, the collector restores the same environment state s_t and reuses the same task, interaction prefix, and visible tool set. Each candidate decision $u_t^{(k)}$ is executed from the recorded state s_t under the shared model-visible context $x_t = (q, m_{<t}, \mathcal{T}_t)$, producing a candidate-specific next state $s_{t+1}^{(k)}$. The SFT policy then continues greedily from that state until termination, subject to collector limits of `max_steps`$= 80$ and `max_turn_steps`$= 20$, yielding a completed sibling trajectory $\tau_t^{(k)}$.

Because stochasticity is introduced only at the fork decision, sibling continuations compare alternative decisions under the same deterministic continuation policy. The resulting sibling trajectories expose the downstream outcomes of alternative decisions made from the same executable context.

Fixed fork schedule and trunk reuse. Forks are selected only along the trunk; states reached inside sibling continuations do not become new fork points. The completed trunk provides the outcome for its retained decision at every fork. For at most B forks and K candidates per fork, including the trunk decision, the single-continuation schedule therefore requires at most

$$N_{\text{paths}} \leq 1 + B(K-1)$$

completed paths: one shared trunk and $K-1$ sibling continuations per fork. Under the primary setting $B = 10$ and $K = 3$, this gives at most 21 completed paths per scenario. Execution and token cost additionally depend on continuation length. Unlike adaptive tree-search procedures such as MCTS-DPO (Xie et al., 2024b), this collector uses completed outcomes only to compare decisions at the recorded fork states.

D.3 Final Outcome Evaluation

Each completed sibling first passes runtime and tool-execution checks. Its scenario-specific verifier is then evaluated on the completed continuation and terminal state. Let

$$\mathcal{V} = \{v_1, \ldots, v_J\}$$

denote the requirements associated with the scenario. These predicates encode applicable state, execution, and policy conditions. For a valid sibling terminating in $s_H^{(k)}$, the verified outcome is

$$r_t^{(k)} = \prod_{j=1}^{J} \mathbb{I}\left[v_j\left(\tau_t^{(k)}, s_H^{(k)}\right) = 1\right].$$

For each evaluated sibling, the collector retains its complete continuation, terminal state, predicate outcomes, execution metadata, and verified outcome.

D.4 Preference-Pair Extraction

Candidate assistant turns are canonicalized before comparison to remove duplicate decisions caused by transient identifiers. For the primary preference corpus, we require a minimum reward gap of 1.0. Because the executable verifier produces binary outcomes, this retains only comparisons between a fully successful candidate and an unsuccessful candidate. For each fork, we retain at most one such pair satisfying

$$r_t^+ = 1, \qquad r_t^- = 0.$$

The resulting preference example is

$$x_t = (q, m_{<t}, \mathcal{T}_t), \qquad y^+ = u_t^+, \qquad y^- = u_t^-.$$

The complete sibling continuations determine the preference label, while the divergent assistant turns serve as the preferred and rejected responses. Each pair is stored with its sibling continuations, terminal outcomes, and verification records as construction provenance. These pairs form $\mathcal{D}_{\text{pref}}$.

D.5 Preference-Data Distribution

The preference corpus contains 1,150 decision-level pairs with 1,049 distinct serialized interaction prefixes. Preferred and rejected responses together contain 439 distinct serialized tool names in their tool-call blocks. Using the same tokenizer as Section 3.3, prefixes average 3,339.8 tokens (range 40–11,602), while preferred and rejected responses average 68.4 (11–669) and 69.9 (10–647) tokens, respectively. Preferred and rejected responses contain an average of 1.12 and 1.20 explicit tool calls. These statistics reflect the intended supervision granularity: relatively short alternative decisions are compared within substantially longer interaction contexts. The 1,150 preference pairs are constructed after SFT and are separate from the 30,395 demonstration trajectories used for supervised fine-tuning.

E Training Objectives

Davinci-Tool-30B is initialized from Qwen3-Coder-30B-A3B-Instruct (Yang et al., 2025a). All SFT demonstrations use the unified interaction serialization introduced in Section 3.

E.1 Training Configuration

Table 13 summarizes the optimization settings for SFT and for DPO on the 1,150-pair preference corpus. The SFT settings are shared by Davinci-Tool-30B-SFT, the Qwen3-8B and Qwen3-14B models trained on Davinci-ToolData, and the Qwen3-Coder-30B model trained on the public Toucan-1.5M release in Table 2. These runs use the same SFT hyperparameters, including four epochs, a learning rate of 1×10^{-5}, and a maximum training sequence length of 32,768 tokens; serialized examples exceeding this limit are truncated by the default SFT training pipeline. Their backbones and training corpora vary as specified in Section 5. SFT trains assistant tokens using the tokenizer's chat template. DPO uses precomputed reference log probabilities and keeps preference examples unpacked. Both stages use fused AdamW (Loshchilov & Hutter, 2017) and enable gradient checkpointing (Chen et al., 2016), FlashAttention (Dao et al., 2022), BF16, and DeepSpeed ZeRO stage 3 (Rajbhandari et al., 2020), with no held-out validation split.

Table 13: Training hyperparameters. The SFT column applies to the 30B full-corpus model and the additional 8B, 14B, and Toucan-1.5M SFT runs. The DPO column applies to Davinci-Tool-30B. Batch size is per device, and gradient accumulation is measured in optimization steps.

Setting	**SFT (shared settings)**	**DPO**
Optimizer	Fused AdamW	Fused AdamW
Learning rate	1×10^{-5}	5×10^{-6}
Epochs	4	1
Learning-rate schedule	Cosine	Cosine
Warmup fraction	0	0.03
Weight decay	0.01	0.01
Maximum sequence length	32,768	24,576
Batch size per device	1	1
Gradient accumulation	4	4
Sample packing	Enabled	Disabled
Padding to maximum length	Enabled	Disabled
Precision	BF16	BF16
Random seed	42	42
Reference log probabilities	—	Precomputed
Base preference scale	—	1.0

E.2 Context Lengths and Evaluation Coverage

The official Qwen3-8B and Qwen3-14B (Yang et al., 2025a) model cards distinguish a native context length of 32,768 tokens from an extended length of 131,072 tokens with YaRN (Peng et al., 2024). Their released configurations set `max_position_embeddings` to 40,960 with no RoPE scaling configured, whereas Qwen3-Coder-30B-A3B-Instruct is released with a 262,144-token context configuration (Yang et al., 2025a). We retain the default Qwen3-8B and Qwen3-14B positional

configurations during fine-tuning and evaluation, with YaRN disabled. Both the original and SFT checkpoints use a 40,960-token serving limit for the total input and generated output. This evaluation limit is distinct from the 32,768-token maximum training sequence length shared by our SFT runs in Table 13.

MCP-Atlas evaluates multi-step tool use from a user request, with subsequent calls returning observations that contribute to the interaction history (Bandi et al., 2026). The context budget must accommodate the prompt, visible tool schemas, retained assistant and tool messages, and generated output; the required length therefore depends on the realized trajectory. In our evaluation, accumulated interactions exceeded the 40,960-token limit and prevented valid MCP-Atlas scores for Qwen3-8B and Qwen3-14B, both before and after SFT. We report these entries as dashes in Table 2; the SFT runs do not change this serving limit.

E.3 Supervised Fine-Tuning

The SFT training corpus is

$$\mathcal{D}_{\mathrm{SFT}} = \mathcal{D}_{\mathrm{EGTS}} \cup \mathcal{D}_{\text{EES-single}} \cup \mathcal{D}_{\text{EES-cross}},$$

with 13,824 EGTS trajectories, 8,026 EES-Single trajectories, and 8,545 EES-Cross trajectories, for a total of 30,395 demonstrations.

Each instance contains the system context, user task, visible tool schemas, assistant messages, executed tool calls, environment observations, and final response. Let z denote the full serialized interaction and $\mathcal{A}(z)$ its assistant-token positions. SFT minimizes

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\mathbb{E}_{z\sim\mathcal{D}_{\mathrm{SFT}}}\left[\sum_{i\in\mathcal{A}(z)} \log \pi_\theta(z_i \mid z_{<i})\right]. \tag{4}$$

Tool outputs remain in the conditioning context for subsequent assistant tokens but are excluded from the supervised loss.

Section 5.3 compares six evaluated combinations of the three construction routes. The full SFT model uses all three.

E.4 Direct Preference Optimization

After SFT, the resulting checkpoint is used to construct $\mathcal{D}_{\mathrm{pref}}$ from restorable EES-Single and EES-Cross scenarios following Appendix D. The resulting preference corpus contains 1,150 decision-level pairs. A frozen copy of the same SFT checkpoint serves as the reference policy π_{ref} for Direct Preference Optimization (Rafailov et al., 2023).

For preference tuples (x, y^+, y^-), DPO minimizes

$$\mathcal{L}_{\mathrm{DPO}}(\theta) = -\mathbb{E}_{(x,y^+,y^-)\sim\mathcal{D}_{\mathrm{pref}}} \log\sigma\left(\beta\log\frac{\pi_\theta(y^+ \mid x)}{\pi_{\mathrm{ref}}(y^+ \mid x)} - \beta\log\frac{\pi_\theta(y^- \mid x)}{\pi_{\mathrm{ref}}(y^- \mid x)}\right).$$

Only the divergent preferred and rejected assistant turns enter the response loss. Their complete downstream continuations are used to establish the outcome-based preference labels. The resulting checkpoint is reported as Davinci-Tool-30B in Section 5.

F Additional Related Work

Toolformer constructs supervision through tool-call augmentation (Schick et al., 2023), while MCP-Flow and TOUCAN scale trajectory collection across real-world MCP tools (Wang et al., 2026a; Xu et al., 2025b). For executable environment synthesis, AutoWebWorld generates verifiable web environments from finite-state models (Wu et al., 2026a), while PROVE grounds task synthesis and reward signals in live MCP servers (Abdelaziz et al., 2026). These approaches are adjacent to the execution-first trajectory and domain-first environment methods discussed in Section 2. Our construction pipeline connects task-conditioned replay in authored systems, reusable executable-domain composition, and same-context preference construction within a unified training workflow.