SAP: State-Guided Data Synthesis with Argument Provenance for Multi-Turn Tool Use

Zichen Tian[1], Jinpeng Chen[1], Cheng Gong[2], Suiyun Zhang[2], Rui Liu[2]*
[1]Independent Researcher [2]Huawei Research
zichentian1024@outlook.com, jinpeng.chen@my.cityu.edu.hk, liu.rui2@huawei.com

Abstract

High-quality multi-turn tool-use data is essential for training agentic models, yet existing data synthesis methods often underrepresent the argument-level dependencies that are critical to long-horizon tool use. As a result, even when a model selects the correct tool, task execution may still fail because the model fills tool arguments with fabricated, stale, or weakly grounded values. To address this problem, we propose **State-Guided Data Synthesis with Argument Provenance (SAP)**. SAP combines state guidance, tool-argument provenance constraints, and turn-level validation to efficiently construct tool-use trajectories with long-range dependencies and high accuracy. Using data generated by SAP, we build SAP-4B, which is highly competitive even when compared with much larger models across multiple benchmarks. Source code, synthesized data, and trained weights are available at https://github.com/Zichen1024/SAP.

1 Introduction

Tool use is the primary interface through which large language models (LLMs) interact with the real world (Patil et al., 2025; Yao et al., 2024; Barres et al., 2025). As LLMs are deployed in long-horizon agentic systems, multi-turn tool use becomes a key axis of agent capability; yet the complexity of state evolution, tool feedback, and cross-turn dependencies makes high-quality multi-turn trajectories scarce, a major bottleneck for training agentic models (Rabinovich and Anaby-Tavor, 2025; Zhang et al., 2025; Prabhakar et al., 2025).

In practice, many multi-turn failures arise not from choosing the wrong tool, but from incorrectly specified tool arguments (Rabinovich and Anaby-Tavor, 2025). Yet existing data-synthesis work mostly focuses on tool-selection dependencies: MAGNET (Yin et al., 2025) and FunReason-MT (Xu et al., 2025a), for example, use pre-built tool dependency graphs to ensure tool-sequence validity, while cross-turn dependencies among tool arguments are largely overlooked. Such dependencies are ubiquitous; arguments typically come from the initial environment state, prior tool-call returns, or earlier user messages, and when underrepresented in training data, the resulting models fabricate upstream references, reuse stale user inputs, or hallucinate substitutes when an upstream call fails. To quantify this gap, we measure the length of cross-turn argument-value propagation chains and the proportion of arguments involved in such dependencies, and find that existing open-source training datasets (Liu et al., 2024; Prabhakar et al., 2025) are consistently shallow on both,[1] leaving models prone to errors in complex multi-turn tasks (Rabinovich and Anaby-Tavor, 2025).

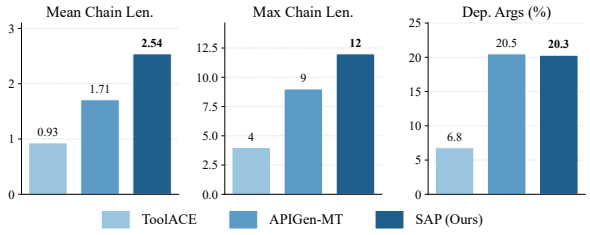

Figure 1: Argument-dependency statistics on representative open-source multi-turn tool-use training sets and ours. *Mean Chain Len.* and *Max Chain Len.* denote the mean and maximum lengths of argument-value propagation chains across turns; *Dep. Args (%)* denotes the proportion of arguments involved in cross-turn dependencies.

Beyond provenance, existing methods also face a hard trade-off between trajectory accuracy and generation cost. Two paradigms currently dominate. **Intent-First** methods (Qin et al., 2024; Prabhakar et al., 2025; Xu et al., 2025b; Zeng et al., 2026; Chen et al., 2026; Xu et al., 2026b) draft user intents first and expand dialogues and tool

*Corresponding author.

[1]Sample sizes for the statistics in Figure 1: ToolACE 11k, APIGen-MT 5k, SAP (Ours) 9k.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 21400–21422
October 24-29, 2026 ©2026 Association for Computational Linguistics

calls post hoc; their validation typically combines rule-based blueprint checks (schema, types, executability) with LLM committee review. Owing to the limited reliability of the underlying LLMs, this validation cannot fully guarantee accuracy, and a single failed check often discards the entire trajectory even when it contains locally correct subsequences. **Trace-First** methods (Yin et al., 2025; Xu et al., 2025a; Hao et al., 2026) sample executable trajectories over a tool-dependency graph and derive user-side queries from those traces; this gives strong executability guarantees but couples the graph to the source tool ecosystem, making migration to new domains costly.

To address these issues, we propose **State-Guided Data Synthesis with Argument Provenance (SAP)**. Given a toolset and its documentation, SAP dynamically constructs a Finite State Machine (FSM) whose edges already carry per-argument provenance tags; it then samples state-transition trajectories from the FSM and fills tool arguments turn by turn. Arguments that depend on prior context are validated against the executed history, and each tool call is executed immediately after generation. If a call fails, only the offending call is retried rather than discarding the trajectory. Once the call sequence is fixed and verified, the framework generates the corresponding natural-language user and assistant messages. By combining state-transition guidance, parameter-provenance awareness, and immediate tool-call validation, SAP jointly supports executability, causal consistency of arguments, and cross-domain scalability, achieving both high quality and efficiency.

Using data synthesized by SAP, we train SAP-4B, which achieves competitive results on both BFCL v4 multi-turn (Patil et al., 2025) and τ^2-bench Retail/Airline (Barres et al., 2025). To support future research, the synthesis code, synthesized data, and trained weights are publicly available at `https://github.com/Zichen1024/SAP`. Our main contributions are as follows:

- We introduce argument-provenance diagnostics for multi-turn tool-use data, showing that existing open-source data contain shallow dependency chains and few arguments that depend on previous turns, limiting their support for learning long-range argument tracking.
- We propose SAP, a state-guided synthesis framework that dynamically builds an FSM from tool documentation, treats argument provenance as an explicit constraint, and validates tool calls turn by turn, enabling the efficient construction of multi-turn tool-use data with strong causal dependencies and high correctness.
- We show that SAP-4B achieves competitive results on both BFCL v4 multi-turn and τ^2-bench; ablation studies further confirm the importance of each component.

2 Related Work

Multi-turn tool-use data synthesis. High-quality multi-turn trajectory synthesis is a principal bottleneck for tool-augmented LLMs (Patil et al., 2025; Yao et al., 2024; Barres et al., 2025; Prabhakar et al., 2025; Xu et al., 2025a; Guan et al., 2025; Hao et al., 2026; Li et al., 2025; Xu et al., 2026a; Chai et al., 2026; Su et al., 2026). The **intent-first** paradigm starts from user intent and expands tool invocations post hoc, instantiated as DFS planning over a large API pool (Qin et al., 2024), structured blueprints validated by rule-based checks and LLM committees (Prabhakar et al., 2025), non-autoregressive skeletons with mask-and-fill refinement (Zeng et al., 2026), graph-sampled planned generation for dialogue coherence (Wang et al., 2025b), user-side intent modeling at scale (Cho et al., 2026), scaling tool-use synthesis from real-world MCP environments (Xu et al., 2025b), or distillation of implicit tool-use intents from raw text (Xu et al., 2026b). The **trace-first** paradigm samples executable traces over a tool dependency graph (hand-curated, signature-derived, or execution-evolved) and derives user queries conditioned on those traces (Yin et al., 2025; Xu et al., 2025a; Hao et al., 2026; Tian et al., 2026). A recent line of *provenance-aware planning*, exemplified by ToolWeave (Khandelwal et al., 2026), constructs tools with built-in dependencies and tracks parameter provenance at plan time to reduce argument hallucination.

In contrast, SAP extends the type system to five categories (P_i, P_o, P_c, P_u, plus the runtime-only P_f) and requires every declared source to be resolved against the executor state prior to binding, with no pre-built or evolved dependency graph, enforcing argument-level causal consistency at synthesis time rather than post hoc. Along a similar line, ToolMind (Yang et al., 2025b) applies per-turn filtering after generation to catch errors that propagate across turns, whereas SAP lifts the check from

a post-hoc filter into a synthesis-time constraint. InfTool (Li et al., 2025) co-evolves synthesized data and the trained model via outer-loop GRPO, whereas SAP keeps the executor loop within a single synthesis pass and produces fixed SFT data.

Multi-turn tool-use benchmarks. Several benchmarks target multi-turn tool use. BFCL v4 (Patil et al., 2025) is among the most widely adopted; its multi-turn track is partitioned into base, long-context, miss-param, and miss-func subsets that probe distinct capabilities. The τ-bench family (Yao et al., 2024; Barres et al., 2025) constructs state-aware Retail/Airline dialogues, requiring the agent to maintain a coherent world state under an LLM-simulated user over long horizons. ToolDial (Shim et al., 2025) and Dialog-Tool (Wang et al., 2025a) provide complementary multi-turn / stateful tool-use datasets. Rabinovich and Anaby-Tavor (2025) expose the fragility of function calling under distribution shift, and Zhang et al. (2025) survey multi-turn LLM interactions more broadly.

3 Preliminaries

3.1 Task Definition

We formalize multi-turn tool-use data synthesis as a **Partially Observable Markov Decision Process (POMDP)**, defined by the tuple $\mathcal{M} = (S, A, O, T)$, where S is the latent environment state space (the world state of the tool-execution environment, e.g., database contents or session context); $A = A_{\text{tool}} \cup A_{\text{resp}}$ is the action space, with A_{tool} covering tool invocations and A_{resp} covering natural-language responses; O is the observation space, comprising tool returns $r \in \mathcal{R}$ and user messages u; and $T(s_{t+1} \mid s_t, a_t)$ is the stochastic transition function, realized by a grounded executor ε. A trajectory $\tau = [(o_1, a_1), \ldots, (o_T, a_T)]$ records the full sequence of observation-action pairs. Because the agent cannot directly read the latent environment state s, it must ground arguments on c_0 (accessed indirectly through read-only tool returns or values relayed via user messages), prior tool-call returns r, or earlier user messages u. The synthesis goal is to produce a dataset $\mathcal{D} = \{(c_0^{(i)}, U^{(i)}, GT^{(i)})\}_i$, where $U^{(i)} = (u_k^{(i)})_k$ is the per-turn user-message sequence, such that every argument value v carries an explicitly declared, executor-verifiable causal source.

3.2 FSM Definition

While the POMDP above captures the full environment dynamics (tool-execution state, returns, user messages), the FSM abstracts over it at the dialogue-phase level: each FSM state represents a recognizable phase of the interaction rather than a concrete environment configuration. SAP models this high-level dialogue structure as a **Finite State Machine (FSM)**:

$$\mathcal{F} = (\Sigma, \sigma_0, \Sigma_f, \Delta), \tag{1}$$

where Σ is a set of *dialogue-phase* states, each instantiating one type of a predefined, domain-agnostic state-type taxonomy (Appendix A.1), $\sigma_0 \in \Sigma$ the initial state, $\Sigma_f \subseteq \Sigma$ the set of accepting terminal states, and Δ the set of directed transition edges. The taxonomy is fixed across domains, while the concrete states and edges of $\mathcal{F}$ are generated per toolset at synthesis time.

States. Each $\sigma \in \Sigma$ describes a *phase* of the user–agent interaction, rather than a tool-level execution dependency. For instance, Σ includes: σ_{init} (session opened), $\sigma_{\text{logged_in}}$ (user authenticated) and $\sigma_{\text{search_done}}$ (search results retrieved).

Edges. Each edge $\delta = (\sigma, \mathcal{C}_\delta, \sigma', \mathcal{T}_\delta) \in \Delta$ encodes one dialogue turn: $\mathcal{C}_\delta = [t_1, \ldots, t_m]$ is the ordered tool list, and $\mathcal{T}_\delta$ is a Provenance Tag map that assigns to every argument a declared causal source from the type system of §3.3. The first edge δ_0 (from σ_0) may only declare P_i or P_c sources, since no prior turns exist to reference.

Extensibility. Unlike trace-first methods that rely on a hand-curated or signature-derived tool-dependency graph, the FSM here is constructed by $\mathcal{A}_{\text{FSM}}$ at synthesis time by matching each tool in the documentation to the predefined dialogue-phase state types it transitions between, without manually specifying pairwise tool dependencies; a new toolset simply triggers automatic rebuilding from its updated documentation.

3.3 Provenance Tag Type System

For every argument p in a tool call, its causal source satisfies:

$$\begin{aligned}&\text{src}(p) \in \\ &\{P_i,\ P_o(k', j', f'),\ P_c,\ P_u(k''),\ P_f(s_0)\}.\end{aligned} \tag{2}$$

Table 1 summarizes the semantics. The four design-time types form $\mathcal{S}_{\text{decl}} = \{P_i,\ P_o,\ P_c,\ P_u\}$, which

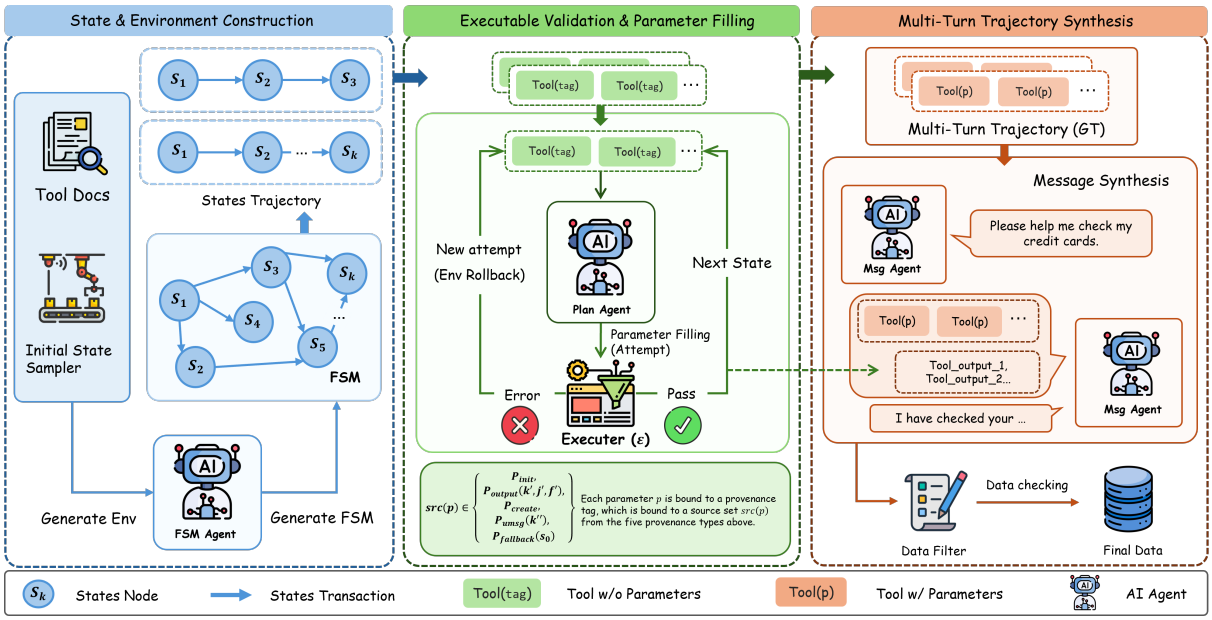

Figure 2: Overview of the SAP pipeline. From left to right, $\mathcal{A}_{\text{FSM}}$ constructs an FSM from tool documentation and sampled initial states, and the pipeline samples a state trajectory from it. $\mathcal{A}_{\text{plan}}$ fills the tagged calls $\text{Tool}(\text{tag})$ with concrete parameters, yielding $\text{Tool}(p)$, and executes them with ε, using environment rollback and local retry on failure. Finally, $\mathcal{A}_{\text{msg}}$ synthesizes the messages for the validated trajectory, which is filtered into the final data. Each argument is assigned a provenance source from $\{P_i, P_o, P_c, P_u, P_f\}$.

$\mathcal{A}_{\text{FSM}}$ declares for each turn; P_f is excluded from $\mathcal{S}_{\text{decl}}$ and is produced only by $\mathcal{A}_{\text{plan}}$ at runtime when a declared source $s_0 \in \{P_i, P_o, P_u\}$ turns out to be unusable. The planner then recovers the argument from another available initial-state value (P_i) or a newly introduced user-side value (P_c) when permitted. The resulting call is executor-validated after binding. In all cases, the original declared source s_0 is preserved in `fallback_from`. The P_f rate probes how faithfully $\mathcal{A}_{\text{plan}}$ honors the declared sources; see App. B.2.

3.4 Argument Dependency Graph

Given a trajectory, we build a directed graph $\mathcal{G} = (V, E)$ over its tool-call arguments. Each node $v \in V$ is one argument-slot binding, a parameter consumed by some tool call at some turn. An edge $(u, v) \in E$ means that v's bound value is reused from an earlier occurrence u of the *same* value instance, where u is the upstream source of that value (which may be a field of a prior tool-call return, or a value the user first supplied in an earlier user message).

For each argument $v \in V$, let $\text{turn}(v)$ be the turn in which v is consumed and $\text{origin}(v)$ be the turn in which the value carried by v first entered the dialogue (via a user message or an earlier tool return). Initial-state values enter the dialogue in the turn that consumes them, so for them $\text{origin}(v) = \text{turn}(v)$. The chain length of v, the mean chain

Sym.	Type	Semantics
P_i	initial_state	A field of c_0; since the assistant cannot read c_0 directly, the value enters turn k either from the user message or from a same-turn read-only call.
P_o	prev_output	Field f' of the j'-th call at a prior executed turn $k' < k$ (all of k', j', f' are bound at runtime by $\mathcal{A}_{\text{plan}}$; $\mathcal{A}_{\text{FSM}}$ only declares the tag).
P_c	self_create	New user-side value introduced at this turn; later surfaced in u_k.
P_u	prev_user_msg	Value semantically introduced in $u_{k''}$ for some $k'' < k$; dialogue generated retroactively.
P_f	fallback	Runtime fallback when a declared source $s_0 \in \{P_i, P_o, P_u\}$ cannot be used: no matching field of c_0, no matching prior return, or no usable anchor value. The planner recovers the argument from another available P_i value or from a new P_c value when permitted; the resulting call is executor-validated after binding, and the original source s_0 is kept in `fallback_from`.

Table 1: Provenance Tag categories. P_i through P_u are declarable by $\mathcal{A}_{\text{FSM}}$ at design time; P_f is produced only by $\mathcal{A}_{\text{plan}}$ at runtime. Parametrized forms ($P_o(k', j', f')$, $P_u(k'')$, $P_f(s_0)$) are used only in the definition of §3.3.

length, and the longest chain over the trajectory are

$$L(v) = \mathrm{turn}(v) - \mathrm{origin}(v),$$
$$\bar{L} = \frac{1}{|V|}\sum_{v\in V} L(v), \quad L^\star = \max_{v\in V} L(v), \tag{3}$$

i.e., $L(v)$ counts how many turn boundaries the value crosses before reaching its consumption point. For example, an argument consumed at turn 3 whose value was first introduced at turn 1 has $L = 2$. We call v a *dependent argument* when $L(v) \geq 1$; in provenance-tag terms (§3.3), arguments with P_i, P_c, or P_f sources always have $L(v) = 0$ (their value is introduced in the consuming turn itself), while P_o and P_u are the two sources of cross-turn dependencies. The fraction of dependent arguments in V is a breadth-style measure of cross-turn grounding, complementary to chain length.

4 Method

4.1 Overview

SAP casts multi-turn tool-use data synthesis as a closed loop in which three specialized LLM agents cooperate with one executor (Figure 2). $\mathcal{A}_{\text{FSM}}$ synthesizes a provenance-annotated FSM tool-call skeleton from the raw tool documentation. $\mathcal{A}_{\text{plan}}$ then fills, for each call, the executor-bound arguments $\boldsymbol{\theta}^{\text{exec}}$ and per-argument provenance metadata $\boldsymbol{\theta}^{\text{prov}}$, dispatches the call to ε, and groups same-turn calls into parallel step groups. After the call sequence is verified, $\mathcal{A}_{\text{msg}}$ synthesizes the trajectory's textual messages. The pipeline enforces a **Provenance Invariant**: before any argument value v is committed, its causal source $\mathrm{src}(v)$ must be explicitly declared, and for the three upstream-grounded types (P_i, P_o, P_u) the referenced upstream must exist or be pre-registered at synthesis time. The two ungrounded types (P_c, P_f) are still logged in $\boldsymbol{\theta}^{\text{prov}}$ so every value remains evaluable. This shifts cross-turn argument dependencies from a passively emerging side-effect into an explicitly planned property validated turn by turn.

4.2 $\mathcal{A}_{\text{FSM}}$: FSM Skeleton Synthesis

$\mathcal{A}_{\text{FSM}}$ synthesizes the FSM $\mathcal{F}$ of §3.2 from the summarized tool documentation $\widetilde{\mathcal{T}}$ for the tool set $\mathcal{T}$ and the initial-state summary $\widetilde{c}_0$. For each edge δ, it commits an ordered tool list $\mathcal{C}_\delta = [t_1, \ldots, t_m]$ for the corresponding turn and a Provenance Tag map $\mathcal{T}_\delta$ assigning every required argument a tag from $\mathcal{S}_{\text{decl}}$ (Table 1). A lightweight verifier $\mathcal{V}_{\text{spec}}$ runs structural checks on the output and triggers regeneration (Appendix A.2).

4.3 $\mathcal{A}_{\text{plan}} + \varepsilon$: Tool-Call Planning and Execution

$\mathcal{A}_{\text{plan}}$ takes the skeleton $\mathcal{F}$ and a sampled path $\pi = (\delta_0, \ldots, \delta_{N-1})$, obtained by a weighted random walk over $\mathcal{F}$ from σ_0 to some $\sigma_f \in \Sigma_f$ (transition weights and long-tail emphasis detailed in Appendices A.1, A.2), and converts it into an executable call sequence carried out by ε. For each turn, $\mathcal{A}_{\text{plan}}$ inspects the executed history $\mathcal{H}_{<k}$ and the declared tags, binds a value for each call, and groups intra-turn calls into parallel step groups (no intra-group P_o edges). ε then executes group by group; on failure, only the offending call is regenerated and retried (Appendices A.3 and A.4).

Two tag types require runtime binding. For P_o, $\mathcal{A}_{\text{FSM}}$ only declares that *some* prior return is reused; $\mathcal{A}_{\text{plan}}$ binds the specific field at runtime against the actual return of ε. If no declared upstream satisfies the binding, it records P_f and recovers the argument from another available P_i value or from a new P_c value when permitted; a declared P_i source that cannot be located in c_0 falls back the same way. The executor validates the resulting call after the recovery value is bound (see App. A.3). This both surfaces the mismatch as a diagnostic signal in $\boldsymbol{\theta}^{\text{prov}}$ (§6) and prevents a single binding failure from invalidating an entire trajectory. For P_u, the value must appear in an earlier user message that $\mathcal{A}_{\text{msg}}$ has yet to generate; we break this cycle by fixing the value at planning time (via a temporary P_c remap) and registering it together with a scheduled anchor turn $k^\star$ in a virtual history Δ_v, which $\mathcal{A}_{\text{msg}}$ later renders into $u_{k^\star}$ (Appendix A.5).

4.4 $\mathcal{A}_{\text{msg}}$: Post-hoc Synthesis of Dialogue Text Messages

Once all $(t_i, \boldsymbol{\theta}_i^{\text{exec}}, r_i)$, $\boldsymbol{\theta}_i^{\text{prov}}$ and Δ_v are frozen, $\mathcal{A}_{\text{msg}}$ synthesizes the user message u_k for every turn and an assistant natural-language response when $\mathcal{C}_k = \varnothing$. Because messages are written only after the ground truth is fixed, every message is grounded against actual returns. The user message u_k is driven by a must-mention list derived from $\boldsymbol{\theta}_i^{\text{prov}}$ and $\Delta_v[k]$, with two regimes: *newly introduced* values (P_c, P_f, the anchor side of a future P_u consumer, and P_i values that must be passed verbatim to a tool) appear as explicit literals in u_k; *referenced* values (P_o and the consumer side of P_u) appear via vague referents whose unambiguity is

verified by the validator. Full marker semantics and prompt-level constraints are in Appendix A.6.

4.5 Challenge Scenario Injection and Multi-Layer Validation

For each validated base trajectory, two rewrite operators synthesize hard variants targeting common deployment failure modes. Ψ_{param} (MissParam) samples a few P_c arguments from an originating turn k^*, replaces their literals in u_{k^*} with vague references, and clears the turn-k^* ground truth so that the assistant emits a clarification; an inserted reveal turn then re-supplies the missing values and re-binds the call. Ψ_{func} (MissFunc) temporarily removes a tool that the base actually invokes and re-exposes it at the next turn via a user-side handoff message. Both operators are guarded by an executor replay that requires the rewritten outcome to match the original base (Appendix A.7).

Before being persisted, every trajectory passes through two complementary checks: a code-side deterministic check (legal provenance references, closed Δ_v, minimum length), and an agent-side LLM judge checking intent alignment and rewrite-scenario structural compliance. Rejection by either check triggers immediate discard.

5 Experiments

5.1 Setup

Backbone and baselines. We SFT on Qwen3-4B-Instruct-2507 (Yang et al., 2025a). The directly comparable open-source BFCL v4 multi-turn baselines at the 4B / 8B scale include AWM (Wang et al., 2026), CM2 (Zhang et al., 2026a), ToolACE-2 and BitAgent (Zhang et al., 2026b), and TOUCAN (Xu et al., 2025b); we additionally report BFCL v3 multi-turn numbers for MAGNET (Yin et al., 2025), ToolWeave (Khandelwal et al., 2026), and MUA-RL (Zhao et al., 2025) as reference (listed in a separate row group in Table 2; not directly comparable due to benchmark version drift).

Synthesis pipeline. The three agents are instantiated as $\mathcal{A}_{\text{FSM}}$ = Gemini-3.1 Pro (Google DeepMind, 2026), $\mathcal{A}_{\text{plan}}$ = Gemini-3 Pro (Google DeepMind, 2025), and $\mathcal{A}_{\text{msg}}$ = Qwen3-235B-A22B-Instruct-2507 (Yang et al., 2025a), chosen to balance FSM-source compliance and dialogue-style diversity (see App. B.2). Gemini-3.1 Pro attains the lowest fallback rate as the planner (2.02% vs. 2.76%, App. B.2), but its endpoint was less stable under high request concurrency; since a single FSM skeleton is amortized over several sampled trajectories while $\mathcal{A}_{\text{plan}}$ accounts for the bulk of the pipeline's request volume, we assign Gemini-3.1 Pro to the low-throughput FSM stage and Gemini-3 Pro to the high-throughput planning stage. The executor ε wraps the original BFCL v4 multi-turn and τ^2-bench backends without modification, so every synthesized trajectory is grounded against the same simulator used at evaluation. The initial configurations consumed at synthesis time are self-generated (following the released schemas) rather than taken from the benchmark test data, which is used only for evaluation. A single pipeline configuration is used across all source domains: only the tool documentation and the executor backend are swapped per domain, while $\mathcal{A}_{\text{FSM}}, \mathcal{A}_{\text{plan}}, \mathcal{A}_{\text{msg}}$, the FSM state-type taxonomy, and the Provenance Tag system remain identical. The released SFT mix contains $\approx$ 9k trajectories.

Training. We train with the verl framework (Sheng et al., 2025) on a single node equipped with eight 80GB GPUs. All experiments use full-parameter SFT with AdamW: learning rate 1×10^{-6}, global batch size 128, and 10 epochs.

Benchmarks and metrics. BFCL v4 multi-turn (Base / MissFunc / MissParam / LongCtx) is scored with the official BFCL harness under the official environments and test data; we report the harness's multi-turn accuracy, which combines state-based checks on the executed environment with response-based checks on the emitted call sequence. τ^2-bench Retail / Airline (Barres et al., 2025) is evaluated under the official harness with the think tool disabled; we report pass^1 only. The argument-value chain-length diagnostics $\bar{L}$ and $L^\star$ (§3) are reported as a synthesis-side measurement in Fig. 1.

5.2 Main Results

Over the Qwen3-4B-Instruct backbone, SAP-4B SFT improves BFCL v4 multi-turn Avg by +8.3 (22.1 $\rightarrow$ 30.4) and τ^2-bench Avg by +2.9 (32.2 $\rightarrow$ 35.1) without any RL. On BFCL v4 the gains concentrate on Base (+11.5), LongCtx (+10.5), and MissParam (+9.0), while MissFunc shows a smaller lift (+2.0). The τ^2-bench improvement (Retail +1.7 / Airline +4.0) is more modest, consistent with the benchmark's long-horizon state-tracking demands; SAP-4B's competitive τ^2 result under a 4B SFT-only setup indicates that argument-level provenance provides a meaningful signal even at this small scale. As a complementary out-of-

Model	Size	BFCL v4 Multi-Turn					τ^2-bench (pass^1)		
		Avg	Base	MFn	MPm	LCtx	Retail	Airline	Avg
Closed-source / large-scale reference (BFCL v4)									
GPT-5.2-High	–	48.5	–	–	–	–	81.6	62.5	72.1
Claude Sonnet 4.5	–	61.4	69.0	65.0	52.5	59.0	86.2	70.1	78.2
Gemini-3 Pro	–	60.8	64.5	60.0	54.5	64.0	85.3	72.7	79.0
DeepSeek-V3.2-Exp	685B	37.4	41.5	39.5	33.5	35.0	–	–	–
Qwen3-235B-A22B	235B	45.4	–	–	–	–	71.9	45.6	58.8
BFCL v3 MT (reference)									
ToolWeave-8B	8B	21.1	28.0	18.0	18.5	20.0	–	–	–
MAGNET-7B-SFT	7B	26.5	35.5	24.0	27.5	19.0	–	–	–
MAGNET-7B-mDPO	7B	27.8	39.0	24.0	26.0	22.0	–	–	–
MAGNET-14B-SFT	14B	33.4	47.0	32.0	32.0	22.5	–	–	–
MAGNET-14B-mDPO	14B	37.9	52.0	36.0	35.5	28.0	–	–	–
MUA-RL-8B	8B	14.6	21.0	11.5	15.0	11.0	49.8	19.0	34.4
MUA-RL-14B	14B	25.3	40.5	14.0	25.0	21.5	66.0	38.0	52.0
MUA-RL-32B	32B	28.4	42.0	20.0	30.0	21.5	67.3	45.4	56.4
BFCL v4 MT (directly comparable)									
CM2-8B-RL	8B	36.5	44.5	32.0	35.0	34.5	36.4	27.0	31.7
CM2-8B-SFT	8B	26.8	30.0	27.5	24.5	25.0	19.5	23.5	21.5
ToolACE-2-8B	8B	37.0	47.0	31.0	28.0	42.0	8.2	26.7	17.5
BitAgent-8B	8B	37.8	46.5	37.5	24.0	43.0	6.1	37.3	21.7
TOUCAN-7B	7B	24.0	30.0	21.0	24.0	21.0	22.8	20.0	21.4
AWM-4B	4B	30.3	37.5	35.5	28.5	19.5	30.3	19.0	24.7
AWM-8B	8B	40.1	49.5	48.0	36.0	27.0	41.2	33.5	37.4
Qwen3-4B-Instruct-2507	4B	22.1	26.5	21.0	15.5	25.5	40.4	24.0	32.2
SAP-4B SFT (ours)	4B	**30.4**	**38.0**	**23.0**	**24.5**	**36.0**	**42.1**	**28.0**	**35.1**
Δ *(Ours − backbone)*		+8.3	+11.5	+2.0	+9.0	+10.5	+1.7	+4.0	+2.9

Table 2: Main results on BFCL v4 Multi-Turn and τ^2-bench. MFn, MPm, and LCtx denote MissFunc, MissParam, and LongCtx, respectively; Retail and Airline are the two τ^2-bench domains. Row groups separate BFCL v4 (directly comparable) from BFCL v3 (reference). **Sources:** GPT-5.2 (OpenAI, 2025), Claude Sonnet 4.5 (Anthropic, 2025), Gemini-3 Pro (Google DeepMind, 2025), DeepSeek-V3.2-Exp (DeepSeek-AI et al., 2025), Qwen3-235B-A22B and backbone (Yang et al., 2025a); ToolWeave (Khandelwal et al., 2026), MAGNET (Yin et al., 2025), MUA-RL (Zhao et al., 2025); ToolACE-2 / BitAgent results from (Zhang et al., 2026b), CM2 (Zhang et al., 2026a), TOUCAN (Xu et al., 2025b), AWM (Wang et al., 2026).

distribution probe, we evaluate SAP-4B on the BFCL v4 single-turn track (Appendix B.4): multi-turn-only training does not regress single-turn performance and yields a small consistent lift on both Non-live and Live splits.

Among methods evaluated on the same BFCL v4 MT release, SAP-4B is on par with same-parameter-scale baselines on BFCL v4 MT Avg (30.4 vs. AWM-4B 30.3) while substantially outperforming them on τ^2-bench (35.1 vs. AWM-4B 24.7), where long-horizon state-tracking is the bottleneck. SAP-4B still trails 8B baselines on BFCL v4, but on τ^2 it matches or exceeds most of them (CM2-8B-RL 31.7, ToolACE-2-8B 17.5, BitAgent-8B 21.7) despite using only SFT on a 4B backbone with 9k trajectories. We attribute the BFCL v4 gap primarily to scale rather than data quality.

The closed-source frontier (Claude Sonnet 4.5, Gemini-3 Pro) reaches BFCL v4 MT Avg in the 60s and τ^2 Avg in the 70s, indicating that the open-source 4B regime still has substantial headroom on BFCL v4 multi-turn. Notably, GPT-5.2-High shows a wider gap to the Claude/Gemini frontier on BFCL v4 MT (48.5 vs. $\sim$60) than on τ^2 (72.1 vs. $\sim$78), suggesting the two benchmarks exercise partly orthogonal capabilities.

5.3 Ablations

Note on $\mathcal{A}_{\text{FSM}}$ and pipeline-level ablations. We do not include an $\mathcal{A}_{\text{FSM}}$ ablation row in Table 3: removing the FSM skeleton (i.e., reverting to free-form turn-by-turn generation by $\mathcal{A}_{\text{plan}}$) cuts the

Variant	Avg	Base	MFn	MPm	LCtx
SAP-4B (Full)	**30.4**	**38.0**	**23.0**	**24.5**	**36.0**
A1: w/o Prov. Tag	24.0	29.5	19.5	21.5	25.5
A2: w/o Ψ	28.8	37.0	21.0	22.5	34.5

Table 3: Ablation on BFCL v4 Multi-Turn at matched data scale. A1 removes per-argument Provenance Tag declarations; A2 removes both rewrite operators $\Psi_{\text{param}}, \Psi_{\text{func}}$.

pipeline's data pass-rate from 89% to 50% (see Appendix B.3), and the surviving trajectories are too few and too biased to support a fair SFT comparison. Similarly, disabling per-call retry (the "noRetry" variant) drops the pass-rate to 54%. Removing Provenance Tags while keeping the FSM ("FSM+noTag") yields a high pass-rate (92%) but produces zero P_o arguments, confirming that the tag declarations are the primary mechanism driving cross-turn dependency generation. We treat $\mathcal{A}_{\text{FSM}}$ as a non-optional structural component and report only the two downstream ablations below.

A1: Provenance Tag declarations. Removing the per-argument Provenance Tag declarations that $\mathcal{A}_{\text{FSM}}$ commits in $\mathcal{T}_\delta$ (§4.2) and letting $\mathcal{A}_{\text{plan}}$ choose argument sources freely causes a substantial drop across all four subsets (Avg $30.4 \rightarrow 24.0$, -6.4). The hit is sharpest on LongCtx (-10.5) and Base (-8.5), with MissFunc (-3.5) and MissParam (-3.0) showing moderate declines, confirming that the tag declarations act as a structural prior that constrains $\mathcal{A}_{\text{plan}}$ to ground each argument in a verifiable upstream, most useful precisely where long-horizon argument tracking matters. App. B.2 reports a complementary backbone-level study where the P_f rate correlates with these declaration-compliance failures.

A2: Rewrite operators $\Psi_{\text{param}}, \Psi_{\text{func}}$. Removing the rewrite operators Ψ_{param} and Ψ_{func} (§4.5) at matched data scale lowers Avg to 28.8 (-1.6). The effect is modest but consistent across MissFunc (-2.0), MissParam (-2.0), Base (-1.0), and LongCtx (-1.5), matching the expectation that the operators specifically target deployment failure modes while leaving standard multi-turn performance largely intact. The smaller magnitude relative to A1 also indicates that the bulk of the gain comes from the structural Provenance Tag constraint, not from the hard-scenario rewrites alone.

5.4 Case Study

Figure 3 provides qualitative evidence for SAP's provenance constraints and execution-in-the-loop validation. In (a), full SAP resolves the `cp` source `routes.js` as P_o from the earlier `ls` return; validation confirms the copy, so the backup can be read on the next turn. Without turn-level validation, a one-character error (`route.js`) causes the copy and the subsequent `cat` call to fail. In (b), when the initial search provides no usable recipient, SAP records P_f, recovers `USR005` from the initial state, and validates the send successfully. Disabling fallback leaves `USR009` ungrounded and nonexistent, causing both message delivery and subsequent verification to fail.

6 Discussion

SAP does not fully prevent argument hallucinations at newly created values (P_c) or at fallback recoveries that must introduce a new value. When a declared upstream is unavailable, P_f instead records the recovery event and its original source, whether the recovered binding selects another available value or introduces a new one; the resulting call is then checked by the executor. This localizes unresolved argument quality to a single call and makes every recovery auditable in $\boldsymbol{\theta}^{\text{prov}}$. Hallucinations cannot propagate along P_o (blocked by real execution) or P_u (blocked by the §4.4 consistency check). This contrasts with committee-style validators that reduce hallucination probability without bounding its propagation scope.

We train on ≈9k trajectories with SFT only, no RL, a deliberately compact setup imposed by the per-trajectory live-executor cost (§7) and our compute budget. Under this small-model, small-data, SFT-only setting, SAP-4B still matches or exceeds several 8B RL-trained methods on τ^2-bench, particularly where long-horizon state-tracking aligns with our provenance design. That this holds *without* scale or RL confounders gives a clean lower-bound estimate of the method's intrinsic data-quality contribution.

Most open-source 4B baselines (e.g., AWM-4B) are think-mode RL-trained models with long deliberation chains before each tool call. SAP-4B trains from a non-think backbone (Qwen3-4B-Instruct-2507) with SFT only, so its per-turn inference cost is substantially lower at comparable or better τ^2-bench accuracy.

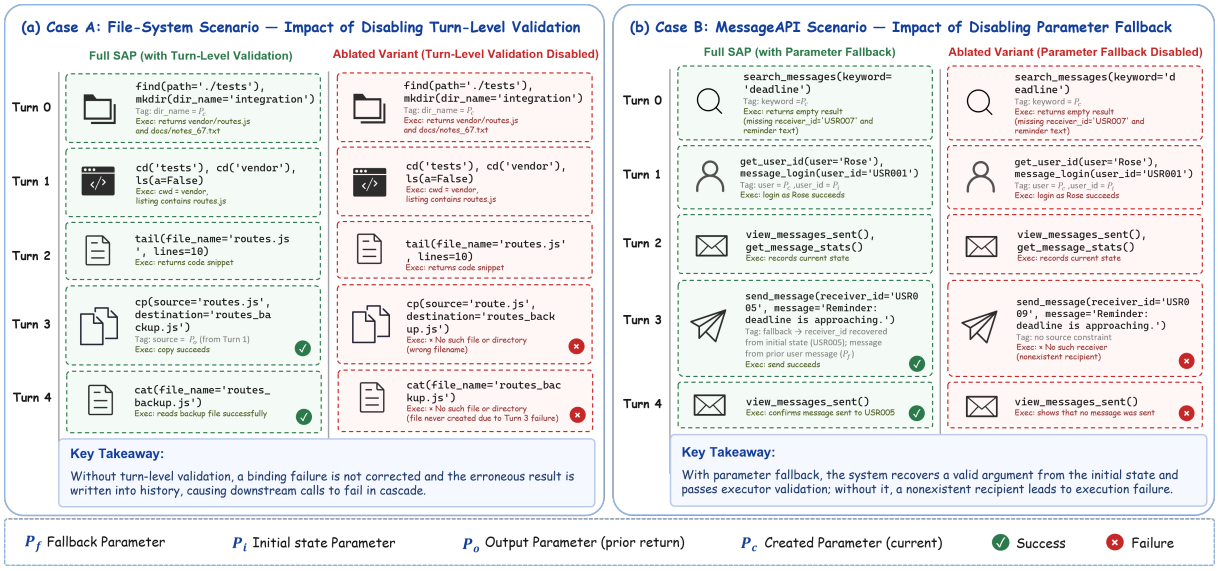

Figure 3: Case study of SAP's error localization and hybrid validation. (a) Turn-level execution validation prevents an incorrect file binding from cascading to a later call. (b) Parameter fallback recovers a valid recipient and is checked by the executor, whereas disabling it causes message delivery to fail.

7 Conclusion

We presented SAP, a state-guided pipeline for multi-turn tool-use trajectory synthesis that promotes argument provenance from a post-hoc annotation to a synthesis-time active constraint. In addition, SAP is domain-agnostic: rebuilding the FSM from updated tool documentation and swapping the executor backend is sufficient to support new toolsets, with no manual rule engineering. The mechanism rests on two components: (i) the Provenance Tag type system, which explicitly declares the causal source of every argument; and (ii) FSM-driven structure-first generation, which dynamically builds a provenance-annotated skeleton from tool documentation and enforces per-turn executability via execution-in-the-loop validation. Experiments and ablations on BFCL v4 multi-turn and τ^2-bench validate the design.

The framework opens two directions. *(i)* SAP *as a compiler for agentic RL.* Provenance Tag serves as a process-reward interface: successful P_o retrievals are positive signals, and argument-binding mismatches localize provenance errors, supporting an SFT cold-start $\rightarrow$ RL with process rewards curriculum without extra annotation, since the signals are written into $\boldsymbol{\theta}^{\text{prov}}$ at synthesis time. *(ii) Scaling and pipeline extensions.* Data scaling, larger backbones, RL, *execution-conditioned FSMs* that revise the skeleton when turns fail, and *depth-guided synthesis* that uses the dependency-graph diagnostic as an optimization objective.

Limitations

We train SAP only on a 4B backbone; the 8B-and-larger numbers in Table 2 are quoted from the original papers, as most concurrent data-synthesis works (FunReason-MT, MAGNET, ToolWeave) release no full pipeline. Methodologically, $\mathcal{A}_{\text{FSM}}$ drafts the skeleton in one pass, so dependencies that surface only after partial execution (e.g., return-code-conditioned branches) are downgraded to P_f at runtime instead of a grounded P_o edge; Provenance Tag covers tool-call arguments only, not entity anaphora in free text; and all runs use English documentation in single-agent settings.

Following standard practice, the data-synthesis pipeline reuses the BFCL and τ^2-bench schemas and executor backends as its environment, though initial states, user messages, and call sequences are sampled independently rather than copied from the test split. Two costs remain: live-executor validation is expensive in wall-clock and API terms (App. B.1), and we do not report human-agreement statistics for the single LLM judge of §4.5.

AI Use Statement

In line with the ACL Policy on AI Writing Assistance, large language models were used as writing aids for grammar polishing, paragraph restructuring, and literature search, and for code completion when implementing the synthesis pipeline. All experimental design, methodological contributions, hypothesis formulations, error analyses, and final

claims are the authors' work; no content was generated end-to-end by AI without subsequent verification and editing.

References

Anthropic. 2025. Introducing Claude Sonnet 4.5. Anthropic news release.

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

Yuxiang Chai, Han Xiao, Xinyu Fu, Jinpeng Chen, Rui Liu, and Hongsheng Li. 2026. UI-KOBE: Knowledge-oriented behavior exploration for lightweight graph-guided gui agents. *Preprint*, arXiv:2605.29534.

Jinpeng Chen, Cheng Gong, Hanbo Li, Ziru Liu, Zichen Tian, Xinyu Fu, Shi Wu, Chenyang Zhang, Wu Zhang, Suiyun Zhang, Dandan Tu, and Rui Liu. 2026. CoVe: Training interactive tool-use agents via constraint-guided verification. *Preprint*, arXiv:2603.01940.

Jungho Cho, Minbyul Jeong, and Sungrae Park. 2026. User-oriented multi-turn dialogue generation with tool use at scale. *Preprint*, arXiv:2601.08225.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, and 245 others. 2025. DeepSeek-V3.2: Pushing the frontier of open large language models. *Preprint*, arXiv:2512.02556.

Google DeepMind. 2025. Gemini 3 Pro. Google DeepMind model page.

Google DeepMind. 2026. Gemini 3.1 Pro model card. Google DeepMind model card.

Ziyi Guan, Jason Chun Lok Li, Zhijian Hou, Pingping Zhang, Donglai Xu, Yuzhi Zhao, Mengyang Wu, Jinpeng Chen, Thanh-Toan Nguyen, Pengfei Xian, Wenao Ma, Shengchao Qin, Graziano Chesi, and Ngai Wong. 2025. KG-RAG: Enhancing gui agent decision-making via knowledge graph-driven retrieval-augmented generation. *Preprint*, arXiv:2509.00366.

Bingguang Hao, Zengzhuang Xu, Yuntao Wen, Xinyi Xu, Yang Liu, Tong Zhao, Maolin Wang, Long Chen, Dong Wang, Yicheng Chen, Cunyin Peng, Xiangyu Zhao, Chenyi Zhuang, and Ji Zhang. 2026. From failure to mastery: Generating hard samples for tool-use agents. *Preprint*, arXiv:2601.01498.

Dinesh Khandelwal, Gnana Prakash Punnavajhala, G P Shrivatsa Bhargav, Gaurav Pandey, Sachindra Joshi, Hima Karanam, and Dinesh Raghu. 2026. ToolWeave: Structured synthesis of complex multi-turn tool-calling dialogues. *Preprint*, arXiv:2605.12521.

Yuwen Li, Wei Zhang, Zelong Huang, Mason Yang, Jiajun Wu, Shawn Guo, Huahao Hu, Lingyi Sun, Jian Yang, Mingjie Tang, and Byran Dai. 2025. Close the loop: Synthesizing infinite tool-use data via multi-agent role-playing. *Preprint*, arXiv:2512.23611.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, and 8 others. 2024. ToolACE: Winning the points of LLM function calling. *Preprint*, arXiv:2409.00920.

OpenAI. 2025. OpenAI Update to GPT-5 System Card: GPT-5.2. OpenAI system card update.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The Berkeley Function Calling Leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 48371–48392. PMLR.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan Wang, Silvio Savarese, and Caiming Xiong. 2025. APIGen-MT: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *Preprint*, arXiv:2504.03601.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*. Spotlight; arXiv:2307.16789.

Ella Rabinovich and Ateret Anaby-Tavor. 2025. On the robustness of agentic function calling. In *Proceedings of the 5th Workshop on Trustworthy NLP (TrustNLP 2025)*, pages 298–304. Association for Computational Linguistics.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. HybridFlow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys)*.

Jeonghoon Shim, Gyuhyeon Seo, Cheongsu Lim, and Yohan Jo. 2025. ToolDial: Multi-turn dialogue generation method for tool-augmented language models.

In *Proceedings of the Thirteenth International Conference on Learning Representations (ICLR)*.

Junhao Su, Yuanliang Wan, Junwei Yang, Hengyu Shi, Tianyang Han, Yurui Qiu, and Junfeng Luo. 2026. Failure makes the agent stronger: Enhancing accuracy through structured reflection for reliable tool interactions. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 12712–12734, San Diego, California, United States. Association for Computational Linguistics.

Xiaoyu Tian, Haotian Wang, Shuaiting Chen, Hao Zhou, Kaichi Yu, Yudian Zhang, Jade Ouyang, Junxi Yin, Jiong Chen, Baoyan Guo, Lei Zhang, Junjie Tao, Yuansheng Song, Ming Cui, and Chengwei Liu. 2026. ASTRA: Automated synthesis of agentic trajectories and reinforcement arenas. *Preprint*, arXiv:2601.21558.

Hongru Wang, Wenyu Huang, Yufei Wang, Yuanhao Xi, Jianqiao Lu, Huan Zhang, Nan Hu, Zeming Liu, Jeff Z. Pan, and Kam-Fai Wong. 2025a. Rethinking stateful tool use in multi-turn dialogues: Benchmarks and challenges. In *Findings of the Association for Computational Linguistics: ACL 2025*.

Zezhong Wang, Xingshan Zeng, Weiwen Liu, Liangyou Li, Yasheng Wang, Lifeng Shang, Xin Jiang, Qun Liu, and Kam-Fai Wong. 2025b. ToolFlow: Boosting LLM tool-calling through natural and coherent dialogue synthesis. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. 2026. Agent World Model: Infinity synthetic environments for agentic reinforcement learning. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR.

Siyuan Xu, Shiyang Li, Xin Liu, Tianyi Liu, Yixiao Li, Zhan Shi, Zixuan Zhang, Zilong Wang, Qingyu Yin, Jianshu Chen, Tuo Zhao, and Bing Yin. 2026a. Controllable and verifiable tool-use data synthesis for agentic reinforcement learning. *Preprint*, arXiv:2604.09813.

Zengzhuang Xu, Bingguang Hao, Zechuan Wang, Yuntao Wen, Xinyi Xu, Yang Liu, Long Chen, Dong Wang, Maolin Wang, Tong Zhao, Yicheng Chen, Cunyin Peng, Jinjie Gu, Leilei Gan, Xiangyu Zhao, Chenyi Zhuang, and Shi Gu. 2025a. FunReason-MT technical report: Advanced data synthesis solution for real-world multi-turn tool-use. *Preprint*, arXiv:2510.24645.

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

Zhihao Xu, Rumei Li, Jiahuan Li, Rongxiang Weng, Jingang Wang, Xunliang Cai, and Xiting Wang. 2026b. Unlocking implicit experience: Synthesizing tool-use trajectories from text. *Preprint*, arXiv:2601.10355. Method name: GEM.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Chen Yang, Ran Le, Yun Xing, Zhenwei An, Zongchao Chen, Wayne Xin Zhao, Yang Song, and Tao Zhang. 2025b. ToolMind technical report: A large-scale, reasoning-enhanced tool-use dataset. *Preprint*, arXiv:2511.15718.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for Tool-Agent-User interaction in real-world domains. *Preprint*, arXiv:2406.12045.

Fan Yin, Zifeng Wang, I-Hung Hsu, Jun Yan, Ke Jiang, Yanfei Chen, Jindong Gu, Long T. Le, Kai-Wei Chang, Chen-Yu Lee, Hamid Palangi, and Tomas Pfister. 2025. Magnet: Multi-turn tool-use data synthesis and distillation via graph translation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32600–32616. Association for Computational Linguistics.

Xingshan Zeng, Weiwen Liu, Lingzhi Wang, Liangyou Li, Fei Mi, Yasheng Wang, Lifeng Shang, Xin Jiang, and Qun Liu. 2026. ToolACE-MT: Non-autoregressive generation for agentic multi-turn interaction. In *Proceedings of the 14th International Conference on Learning Representations (ICLR)*. ArXiv:2508.12685.

Chen Zhang, Xinyi Dai, Yaxiong Wu, Qu Yang, Yasheng Wang, Ruiming Tang, and Yong Liu. 2025. Beyond single-turn: A survey on multi-turn interactions with large language models. *Preprint*, arXiv:2501.09959.

Zhen Zhang, Kaiqiang Song, Xun Wang, Yebowen Hu, Weixiang Yan, Chenyang Zhao, Henry Peng Zou, Haoyun Deng, Sathish Reddy Indurthi, Shujian Liu, Simin Ma, Xiaoyang Wang, Xin Eric Wang, and Song Wang. 2026a. CM2: Reinforcement learning with checklist rewards for multi-turn and multi-step agentic tool use. *Preprint*, arXiv:2602.12268.

Zhiwei Zhang, Fei Zhao, Rui Wang, Zezhong Wang, Bin Liang, Jiakang Wang, Yao Hu, Shaosheng Cao, and Kam-Fai Wong. 2026b. Robust tool use via Fission-GRPO: Learning to recover from execution errors. *Preprint*, arXiv:2601.15625.

Weikang Zhao, Xili Wang, Chengdi Ma, Lingbin Kong, Zhaohua Yang, Mingxiang Tuo, Xiaowei Shi, Yitao

Zhai, and Xunliang Cai. 2025. MUA-RL: Multi-turn user-interacting agent reinforcement learning for agentic tool use. *Preprint*, arXiv:2508.18669.

Appendix

A SAP Pipeline Details

This appendix expands the design choices behind the three agents and the rewrite operators of §4.

A.1 Detailed FSM Specification

The FSM of §3.2 is realized by mapping each domain's tool documentation to a shared, domain-agnostic *state-type taxonomy* that lists which dialogue phases may appear in any tool-use scenario. New domains reuse the same taxonomy and downstream protocol; supplying updated tool documentation is sufficient to regenerate the corresponding states and transitions, without manually rewriting rule code.

State-type taxonomy (Σ). We use 13 abstract types reused across all domains: `INITIAL` (session start); `AUTH_REQUIRED` / `AUTH_COMPLETED` (authentication phases); `INFO_GATHERING` (waiting for user-supplied arguments); `SEARCHING` (read-only lookup in progress); `FOUND` / `UNAVAILABLE` (lookup outcomes); `ACTION_REQUIRED` (irreversible action pending user confirmation); `ACTION_COMPLETED` (action persisted by the executor); `ERROR` (recoverable executor error); `REJECTION` (out-of-scope or invalid request); `COMPLETED` (terminal success); and `NORMAL` (catch-all for domain-specific intermediate phases). Each $\sigma \in \Sigma$ instantiates exactly one of these types.

Transition record ($\delta \in \Delta$). Each transition carries six fields beyond the source/target states:

- `action`: the tool (or composite tool list $\mathcal{C}_\delta$) whose invocation realizes the transition.
- `condition`: a natural-language guard ($\mathcal{A}_{\text{plan}}$ uses this in-context).
- `probability`: base sampling probability used by the weighted random walk over $\mathcal{F}$.
- `weight`: long-tail emphasis weight (see §A.2).
- `is_critical`: marks rare but high-value paths (e.g., authentication failure, payment-error branches) that the long-tail emphasis subset $\mathcal{E}$ preferentially covers.
- `provenance_tag`: realizes $\mathcal{T}_\delta$ at the engineering level, encoded as a nested map `{tool:{arg: tag}}` where `tag` $\in$ `{initial_state, prev_output, self_create, prev_user_msg}` (the four design-time entries of $\mathcal{S}_{\text{decl}}$; P_f is reserved for runtime and never appears here).

Cross-domain reuse. Our implementation provides concrete FSMs for the τ^2-bench Retail and Airline domains and for the BFCL v4 multi-turn domains (e.g., Twitter, file system), all instantiating the taxonomy above. The Retail FSM, for example, defines $|\Sigma| = 25$ states (including domain-specific instances such as `auth_email_prompt`, `info_gathering_order_id`, `search_order`, `order_found`, `action_confirm_cancel`, `cancel_success`) and $|\Delta| = 35$ transitions; the Airline and BFCL multi-turn FSMs reach comparable scale. A representative transition record:

```
{
  "from_state": "info_gathering_order_id",
  "to_state":   "search_order",
  "action":     "search_order",
  "condition":  "user supplies order ID",
  "probability": 1.0, "weight": 1.0,
  "is_critical": false,
  "provenance_tag": {
    "search_order": {
      "order_id": "prev_user_msg"
    }
  }
}
```

Adding a new domain therefore requires only supplying its tool documentation for this mapping; no change to $\mathcal{S}_{\text{decl}}$, the executor protocol, or downstream agents is needed.

A.2 $\mathcal{V}_{\text{spec}}$: Structural Constraints and Long-Tail Coverage

The verifier $\mathcal{V}_{\text{spec}}$ rejects an FSM $\mathcal{F}$ produced by $\mathcal{A}_{\text{FSM}}$ on three routine sanity checks (topological legality of Σ, Δ; existence of at least one length-N path from σ_0; argument completeness of $\mathcal{T}_\delta$ for every required argument) and triggers regeneration. A more substantive constraint forbids P_o and P_u on the first edge (outgoing from σ_0). This is the most frequent failure mode we observed in $\mathcal{A}_{\text{FSM}}$'s raw output: because the FSM is drafted in one pass, the very first turn is often declared to reuse a prior return or an earlier user message even though no prior turn exists. Rejecting such skeletons at the verifier removes this "first-turn reference to non-existent context" error before any planning cost is incurred.

Long-tail coverage via inverse-frequency emphasis. Pure unconstrained sampling lets head tools

dominate $\mathcal{F}$ and starves rare tools of training signal. We maintain a flat call-frequency histogram $H : \mathcal{T} \to \mathbb{N}$ across all synthesized trajectories and, for each new FSM, sample an emphasis subset $\mathcal{E} \subseteq \mathcal{T}$ under inverse-frequency weighting; $\mathcal{A}_{\text{FSM}}$ must commit at least one turn whose call list intersects $\mathcal{E}$. H encodes only tool frequencies, not inter-tool dependencies, so the resulting bias does not leak any graph structure back into $\mathcal{F}$.

A.3 $\mathcal{A}_{\text{plan}} + \varepsilon$: Filling Protocol

$\mathcal{A}_{\text{plan}}$ fills each call t_i in turn δ_k in a refill-on-failure loop. Two design choices are worth highlighting.

Two parallel outputs. The planner emits two parallel objects per call:

$$(\boldsymbol{\theta}_i^{\text{exec}}, \boldsymbol{\theta}_i^{\text{prov}}) \sim \mathcal{A}_{\text{plan}}(t_i, \mathcal{T}_\delta[t_i], \mathcal{H}_{<k}, c_0, \ell_{\text{last}}) \,, \tag{4}$$

where $\boldsymbol{\theta}_i^{\text{exec}}$ holds the executor-bound argument values and $\boldsymbol{\theta}_i^{\text{prov}}$ holds the per-argument Provenance Tag and (if applicable) the upstream reference. Decoupling the two tracks lets the executor proceed with $\boldsymbol{\theta}_i^{\text{exec}}$ while every fallback escalation or upstream rebinding is logged into $\boldsymbol{\theta}_i^{\text{prov}}$ without disturbing dispatch.

Error feedback as in-context signal. On a failed call, the executor error ℓ_{last} is fed back into the next prompt rather than discarded. This converts the executor into an in-context critic for the planner LLM, which empirically removes most repeated schema/type errors within a couple of refills. If a call still fails after the cap, the trajectory is truncated at the current turn, and trajectories left too short are dropped to avoid degenerate samples.

A.4 $\mathcal{A}_{\text{plan}}$: Intra-Turn Parallelization Grouping

Many tool-set backends admit multiple parallel calls within a single turn (e.g., independent lookups). After all calls of a turn are bound (i.e., arguments filled and validated), $\mathcal{A}_{\text{plan}}$ partitions them into ordered groups $(G_1, \ldots, G_p)$ such that no P_o edge lies within a group (at execution time, intra-group returns become visible only after the entire group completes), with groups ordered by minimum call index.

A.5 $\mathcal{A}_{\text{plan}}$: Resolving the P_u Dependency Cycle

The P_u type is the only source whose dialogue rendering lies outside $\mathcal{A}_{\text{plan}}$'s purview, since user messages are synthesized later by $\mathcal{A}_{\text{msg}}$. We resolve the cycle through a remap-bind-restore protocol mediated by a virtual history Δ_v.

Virtual history Δ_v. Δ_v is a per-turn list of *pending introductions*: $\Delta_v[k] = \{(p_1, v_1), (p_2, v_2), \ldots\}$, meaning "$u_k$ must explicitly introduce values $v_1, v_2, \ldots$ for downstream consumption." $\mathcal{A}_{\text{plan}}$ populates Δ_v during planning; $\mathcal{A}_{\text{msg}}$ later consumes it in turn order.

Remap-bind-restore. For each argument p with $\text{src}(p) = P_u$ at turn k, the planner first samples an anchor turn $k^\star < k$, avoiding turns that already anchor an unrelated value of the same surface form (otherwise the later disambiguator could not tell two homonymous anchors apart). It then *temporarily* sets $\text{src}(p) \leftarrow P_c$ and invokes the filling protocol of Appendix A.3 to obtain a concrete value v that passes executor validation. Finally, it restores the tag to P_u (recording $k^\star$ as the anchor turn) and appends (p, v) to $\Delta_v[k^\star]$.

Closure check. After all turns are planned, a closure check verifies that every P_u argument with anchor $k^\star$ has a matching pair in $\Delta_v[k^\star]$ and that $\mathcal{A}_{\text{msg}}$ later emits v as an explicit literal in $u_{k^\star}$ under normalized (case- and whitespace-insensitive) matching. Closure failures roll back the trajectory. Conceptually, the trick is that P_c depends on no upstream, so reassigning the tag yields a value that is both executor-valid and free to appear anywhere earlier in the dialogue; Δ_v is the bridge that turns this freedom into a contract honored by $\mathcal{A}_{\text{msg}}$.

A.6 $\mathcal{A}_{\text{msg}}$: Must-Mention Dispatch and Per-Call Synthesis

$\mathcal{A}_{\text{msg}}$ synthesizes u_k under a per-turn must-mention list $\mathcal{M}_k$ derived from $\boldsymbol{\theta}^{\text{prov}}$ and $\Delta_v[k]$. Four markers cover all surface behaviors of values in u_k (Table 4). P_i values that the assistant can discover via auxiliary read-only calls (e.g., listing the current directory) are dropped from $\mathcal{M}_k$; only those that must be passed verbatim into a tool argument (IDs, credentials, token-like fields) are marked `[FROM_CONFIG]`.

Per-call decomposition. A naive prompt would hand $\mathcal{A}_{\text{msg}}$ the entire $\mathcal{M}_k$ and ask for u_k in one shot; in practice this causes the LLM to overload the front of u_k with literals and skip later constraints. We instead slice $\mathcal{M}_k$ per call,

$$\mathcal{M}_k = \bigsqcup_{i=1}^{n} \mathcal{M}_k^{(i)}, \tag{5}$$

Marker	Trigger (when processing turn k)	Expression constraint in u_k
[NEW]	$\mathrm{src}(p) = P_c$, *or* $(p, v) \in \Delta_v[k]$ (anchor for downstream P_u)	Explicit literal value: first occurrence in the dialogue.
[FROM_HISTORY]	$\mathrm{src}(p) \in \{P_o,\ P_u\}$ (upstream at turn $< k$)	Vague reference ("that file", "the ID you found earlier"); literal must NOT be repeated.
[FROM_CONFIG]	$\mathrm{src}(p) = P_i$, value must be passed verbatim to a tool argument	Explicit literal value (the assistant cannot read c_0): IDs, credentials, token-like fields.
[FALLBACK_from_s_0]	$\mathrm{src}(p) = P_f$ (original source s_0 in metadata)	Treated as [NEW]: explicit literal value.

Table 4: Must-mention markers dispatched by the orchestrator to $\mathcal{A}_{\text{msg}}$ for user-message synthesis.

where $\mathcal{M}_k^{(i)}$ contains only the slots that c_i actually consumes, generate an intent fragment ϕ_i for each c_i under its slice, and concatenate the fragments with sequencing connectives:

$$u_k = \mathrm{Compose}(\phi_1, \ldots, \phi_n), \tag{6}$$

$$\phi_i = \mathcal{A}_{\text{msg}}(c_i \mid \mathcal{H}_{<k}, \mathcal{M}_k^{(i)}). \tag{7}$$

The per-call decomposition yields by construction the property that every ground-truth call has a corresponding semantic expression in u_k, which removes the need for an additional "call coverage" post-check.

Four prompt-level constraints. On top of $\mathcal{M}_k^{(i)}$, the fragment prompt enforces four semantic constraints that empirically suppress the most common degenerations: (i) u_k is the sole signal driving the assistant on tool-call turns, so it cannot rely on assistant clarification; (ii) every entry in $\mathcal{M}_k^{(i)}$ must be matched by an expression in ϕ_i; (iii) the semantic direction of ϕ_i must agree with c_i's actual operation (when c_i is `unfollow`, ϕ_i must not express "follow"); (iv) when u_k belongs to a different intent family from prior turns (e.g., search→delete), the shift must be expressed explicitly. These collectively prevent silently mis-aligned user messages from passing the downstream LLM judge.

Assistant text responses. On turns with $\mathcal{C}_k = \varnothing$, $\mathcal{A}_{\text{msg}}$ uses a dedicated prompt (e.g., `ASSIST_CLARIFY_PROMPT` for MissParam) and is required to (i) address the immediately preceding user intent (e.g., request the missing argument), (ii) introduce no new tool call or capability promise, and (iii) not disclose information that the real tool returns have not exposed. The prior context visible to $\mathcal{A}_{\text{msg}}$ is the real executed history (calls, arguments, returns, descriptions), so both user and assistant text are grounded against actual returns rather than speculative ones.

A.7 Challenge-Scenario Rewrite Operators

Both rewrite operators take a validated base trajectory and produce a hard variant; the salient trick they share is an executor-replay check: the rewritten trajectory is re-executed through ε and accepted only when its execution outcome matches the original base. Without this check, vague-reference rewrites or toolkit removals can silently produce unsolvable trajectories.

Ψ_{param} (MissParam). The operator samples one or more arguments with $\mathrm{src}(p) = P_c$ from the base, replaces the corresponding literals in u_{k^*} with vague references ("`report.pdf`" → "that file"), and empties turn k^*'s ground-truth call list. The assistant therefore emits a clarification request synthesized by $\mathcal{A}_{\text{msg}}$ in place of a tool call. A new reveal turn is inserted at position $k^* + 1$ in which $\mathcal{A}_{\text{msg}}$ generates a natural user message surfacing the missing values, and the original call is rebound at this turn. Multiple missing arguments may be revealed in a single turn or spread across several reveal turns; the operator is agnostic to the dataset's exact clarification format (free text, a `request_for_info` slot, etc.).

Ψ_{func} (MissFunc). The operator removes a tool t^* that the base actually invokes from the visible toolkit at turn k^*, so the assistant cannot proceed. A user-side handoff message is injected whose semantic event is "new tools have been registered into the visible toolkit," re-exposing $\mathcal{T}$; the assistant then invokes t^* at $k^* + 1$. Both the validator and the training objective are driven by this semantic event, not the surface wording, so alternative carriers (a system-side marker, an external tool-registry event) are interchangeable; this keeps Ψ_{func} compatible with future handoff paraphrase strategies.

B Additional Ablation Studies

This appendix complements the main-paper ablations of §5.3 with three pipeline-side studies: a synthesis-cost estimate (§B.1), a cross-backbone fallback-rate analysis (§B.2), and a pipeline-component ablation on data-generation success rate (§B.3).

B.1 Synthesis Cost

We reran the same synthesis pipeline while recording API usage for 500 validated multi-turn trajectories. The measured average cost was $0.49 per trajectory. Extrapolating this cost under the same pipeline configuration and trajectory mix gives an estimated cost of approximately $4,410 for the released set of about 9k trajectories. This estimate covers synthesis-time API usage and excludes SFT training and downstream evaluation. Because comparable trajectory-level billing logs are unavailable for existing methods, we do not make a direct numerical cost comparison; local retry is expected to avoid some recomputation by refilling only failed calls rather than regenerating complete trajectories.

B.2 Plan-Agent Fallback Rate Across Backbones

To assess how faithfully different backbones honor the FSM-declared sources, we run $\mathcal{A}_{\text{plan}}$ on the same 500 FSM skeletons with four backbones and measure the P_f rate, defined as the fraction of filled arguments for which the declared $s_0 \in \{P_i, P_o, P_u\}$ cannot be used directly and the planner records a recovery binding. A recovery uses a P_i or P_c value when permitted; the resulting call is executor-validated after binding, and the original s_0 remains in `fallback_from`. This is an *argument-level* fallback (the P_f Provenance Tag of §3.3). We also report the distribution of declared sources actually consumed (P_i / P_o / P_c). P_u arguments are counted under P_c in this distribution, because the remap-bind-restore protocol of Appendix A.5 binds their values through the P_c path at fill time and only restores the tag afterwards; the three source columns and Fb.% therefore sum to 100 up to rounding. Lower fallback indicates stricter adherence to the Provenance Invariant. This study is a separate synthesis run from the pipeline ablation of Appendix B.3, using its own sampled skeletons and initial configs, so the absolute proportions are not directly comparable across the two tables.

Backbone	Traj	Args	Fb.%	P_i %	P_o %	P_c %
Gemini-3.1 Pro	500	3,962	**2.02**	42.7	25.5	29.8
Gemini-3 Pro	500	4,348	2.76	36.4	24.0	36.9
DeepSeek-V4-Flash	500	3,253	3.08	28.3	9.2	59.4
Qwen3.5-Plus	500	4,365	4.82	27.3	34.4	33.5

Table 5: Cross-backbone P_f study on identical FSM skeletons. Traj is the number of trajectories and Args the total number of filled arguments across them; Fb.% is the runtime fallback rate; the remaining columns show the distribution of declared sources actually consumed by $\mathcal{A}_{\text{plan}}$.

Observations. (i) Gemini-3.1 Pro achieves the lowest fallback (2.02%) and the highest P_i utilization (42.7%), indicating the strongest adherence to declared sources. (ii) Qwen3.5-Plus shows the highest fallback (4.82%) with heterogeneous `fallback_from` labels (`default`, `user_request`, `user_intent`), suggesting weaker compliance with the FSM-imposed source constraint. (iii) DeepSeek-V4-Flash exhibits an atypical distribution: P_o drops to 9.2% while P_c climbs to 59.4%, and the total argument count is the lowest (3,253 vs. 4,348 for Gemini-3 Pro on the same skeletons). The backbone tends to fabricate literals rather than reference prior tool returns, producing shorter trajectories. (iv) Gemini-3 Pro sits in the middle (2.76%) with all fallbacks routed through P_c, giving the most predictable failure mode. Our main pipeline therefore stays within the Gemini family: Gemini-3.1 Pro drives $\mathcal{A}_{\text{FSM}}$, whose cost is amortized because one skeleton yields several sampled trajectories, while $\mathcal{A}_{\text{plan}}$ uses Gemini-3 Pro, whose endpoint remained stable at the request throughput a full synthesis run requires.

B.3 Pipeline-Level Ablation

Table 6 reports the effect of turning off individual pipeline components on data generation success rate and argument-source distribution. All variants use Gemini-3 Pro as the backbone and share the same toolset and initial-config pool. This is an independent synthesis run from the cross-backbone study of Appendix B.2 (different sampled FSM skeletons and initial configs), so the absolute source proportions differ from Table 5; only within-table comparisons across variants are meaningful. We generate 500 trajectories per variant and measure the executor pass-rate (Succ) and the fraction of arguments assigned to each source type (including the argument-level fallback rate P_f), with P_u again

Variant	Succ	P_f %	P_o %	P_c %	P_i %
Full SAP	89%	2.06	29.5	40.9	27.5
FSM+noTag	92%	0.00	0.0	53.8	46.2
noFSM+Tag	50%	15.75	16.2	43.9	24.2
noFSM+noTag	66%	0.00	0.0	54.2	45.8
noRetry	54%	3.45	24.8	42.3	29.4

Table 6: Pipeline-level ablation. FSM = structure-first skeleton; Tag = Provenance Tag declarations; Succ = trajectories that pass executor validation; P_f = argument-level fallback rate; remaining columns show the fraction of arguments assigned to each source type (P_u counted under P_c).

counted under P_c as in Appendix B.2. In the Full, FSM+noTag, and noRetry variants, the FSM samples up to 3 paths per trajectory and each call can be refilled up to 3 times on executor failure. In the noFSM variants, only a single path is generated (no FSM path sampling), but the per-call 3-retry refill budget remains.

Several patterns stand out. (i) Removing the FSM skeleton (noFSM+Tag) nearly halves the success rate (89% $\rightarrow$ 50%) and drastically increases the argument-level fallback rate to 15.75%. This confirms that the FSM's structure-first prior is the primary guarantor of causal chain existence: without the FSM's macro-level path planning, $\mathcal{A}_{\text{plan}}$ degenerates into free-form turn-by-turn generation. Although the historical context is preserved, the generated tool-call sequences misalign with the Tag-declared causal dependencies (i.e., a Tag declares a need for P_o, but the preceding tools fail to produce that specific return). Consequently, a massive number of declared P_o sources cannot be matched in the history, forcing the planner to fall back to P_c. (ii) Removing only the Provenance Tags (FSM+noTag) actually yields a slightly higher pass-rate (92%) and zero fallback, but produces zero P_o arguments; the pipeline reverts to generating only local (P_i and P_c) sources, losing all cross-turn dependency structure. (iii) Disabling per-call retry (noRetry) cuts the success rate to 54%, highlighting the importance of the executor-in-the-loop refill mechanism. (iv) The noFSM+noTag variant confirms that the FSM and Tag components are complementary: without either, the pipeline produces only local arguments at a moderate pass-rate. Together, these results complement the training-level ablations of Table 3 by showing that FSM, Tag, and Retry each contribute a distinct and necessary function at the data-generation stage.

Split	Backbone	SAP-4B	Δ
Non-live	84.36	84.88	+0.52
Live	75.41	75.95	+0.54
Overall avg	79.89	80.42	+0.53

Table 7: BFCL v4 single-turn results (accuracy, %). Backbone is Qwen3-4B-Instruct-2507; SAP-4B is the same backbone after SFT on SAP data. Multi-turn-only training neither degrades single-turn performance nor regresses on the noisier Live split, and yields small consistent gains on both subsets.

B.4 Out-of-Distribution Generalization on BFCL Single-Turn

SAP-4B is trained exclusively on multi-turn trajectories synthesized by the pipeline of §4. To probe whether this multi-turn-focused training transfers to settings that differ in both interaction structure and task distribution, we evaluate SAP-4B on the BFCL v4 single-turn track (Table 7), which is split into two complementary subsets:

- **Non-live** (simple, parallel, multiple, parallel_multiple, irrelevance): static, BFCL-authored benchmark tasks. The distribution is controlled and reproducible; performance here primarily reflects basic function-calling capability (tool selection, argument filling, intra-turn parallelism).

- **Live** (live_simple, live_parallel, live_multiple, live_parallel_multiple, live_relevance, live_irrelevance): real-world user-contributed queries with noisier phrasing and broader intent coverage. Performance here is more sensitive to generalization and robustness.

Two takeaways. First, multi-turn-only training does not regress on single-turn tasks: Non-live accuracy edges up by $+0.52$, indicating that the multi-turn argument-provenance prior does not interfere with single-step tool selection or argument filling. Second, Live accuracy improves by a comparable $+0.54$; since Live queries are out-of-distribution with respect to both the synthesis-time tool schemas and the multi-turn structure, this constitutes a mild but consistent OOD generalization signal. The overall lift ($+0.53$) is modest in absolute magnitude but uniformly positive, suggesting that the structural prior induced by Provenance Tag constraints transfers as auxiliary signal even when the trajectory collapses to a single turn.

	Turns	P_i	P_o	P_c/P_f	**Span**
Trajectory A (local)	2	0	0	3	0
Trajectory B (cross-turn)	3	0	2	1	2

Table 8: Source distribution and max dependency span for the two case-study trajectories.

C Case Study: Local vs. Cross-Turn Argument Grounding

To illustrate how SAP's provenance constraints shape trajectory structure, we present two simplified trajectories (Figures 4 and 5). Trajectory A uses only local sources; Trajectory B contains a cross-turn P_o dependency. Table 8 summarizes their dependency profiles.

In Trajectory A, every argument is a literal that the user supplies in the same turn that consumes it (P_c); no cross-turn propagation occurs, and all arguments have $L = 0$. In Trajectory B, by contrast, the `mention` call consumes two cross-turn values: `tweet_id` from Turn 2's `post_tweet` return ($L = 1$) and `mentioned_usernames` from Turn 1's `list_all_following` return ($L = 2$). SAP produces these dependencies by design: the FSM skeleton declares the P_o edges at plan time, and $\mathcal{A}_{\text{plan}}$ resolves them against the real executor state before binding. Both trajectories are valid multi-turn dialogues, but SAP's provenance constraints make Trajectory B the representative case, directly addressing the dependency-gap diagnostic of §3.

D Prompt Templates

This appendix lists condensed English templates of the four core prompts used by the three agents and the rewrite operators (Figures 6–10). Boilerplate (login etiquette, path conventions, output-format reminders) and few-shot examples are omitted; the released code carries the full prompts. Cyan placeholders such as `{N}` are filled at synthesis time.

Trajectory A: Local Grounding Only (Travel domain)

Turn 1

User: "Show me all the credit cards saved on my travel account."

Assistant:

```
<tool_call>
[get_all_credit_cards()]
</tool_call>
<tool_response>
[{"credit_card_list": {"CARD001": {...},"CARD002": {...}}}]
</tool_response>
```

Turn 2

User: "Now calculate how much 1.0 USD is in EUR."

Assistant:

```
<tool_call>
[compute_exchange_rate(base_currency="USD",target_currency="EUR", value=1.0)]
</tool_call>
<tool_response>
[{"exchanged_value": 0.91}]
</tool_response>
```

▷ `base_currency`, `target_currency`, `value`: P_c

Figure 4: Trajectory A with only local sources. The three argument slots of `compute_exchange_rate` are user-supplied literals (P_c); the preceding zero-argument call contributes no argument slot, and no argument depends on a prior turn (L=0).

Trajectory B: Cross-Turn P_o Dependency (Twitter domain)

Turn 1

User: "First check whether I'm logged in, then show me a list of all the accounts I'm currently following."

Assistant:

```
<tool_call>
[posting_get_login_status(), list_all_following()]
</tool_call>
<tool_response>
[{"login_status": true}, {"following_list": ["dev_tips","popular_science", "movie_reviews",
↪  "ai_updates", "nature_news"]}]
</tool_response>
```

Turn 2

User: "Now post a tweet with the text 'Trying to tweet more intentionally...'."

Assistant:

```
<tool_call>
[post_tweet(content="Trying to tweet more intentionally...")]
</tool_call>
<tool_response>
[{"id": 16, ...}]
</tool_response>
```

Turn 3

User: "Now mention all the accounts I follow in that tweet."

Assistant:

```
<tool_call>
[mention(tweet_id=16, mentioned_usernames=["dev_tips","popular_science", "movie_reviews",
↪  "ai_updates","nature_news"])]
</tool_call>
<tool_response>
[{"mention_status": "success"}]
</tool_response>
```

▷ `content`: P_c (Turn 2); `tweet_id`: P_o $(k'{=}2, j'{=}1, \texttt{id})$; `mentioned_usernames`: P_o $(k'{=}1, j'{=}2, \texttt{following_list})$

Figure 5: Trajectory B with cross-turn P_o dependencies: Turn 3's `mention` consumes `tweet_id` from Turn 2's `post_tweet` return and `mentioned_usernames` from Turn 1's `list_all_following` return (maximum span $L^{\star}{=}2$).

$\mathcal{A}_{\text{FSM}}$: FSM Skeleton Synthesis

<task>
You are an FSM designer. Build executable, logically sound, diverse FSM skeletons for multi-turn tool-calling dialogs over a target toolset.
</task>

<structural_constraints>
- Exactly one state with `type="initial"`; at least one terminal state.
- Each turn carries fields `from / to / action / function_calls / prov_tag`.
- `function_calls` has length 1–10; names must come exactly from the provided tool list.
</structural_constraints>

<design_constraints>
- DAG-structured (no cycles); depth $\geq$ `{N}` from σ_0.
- Call-count distribution: $\sim$40% turns with 2–4 calls, $\sim$45% with 5–7, $\sim$15% with one. Each combination must reflect a realistic single-step user intent.
- Cross-turn dependencies: later turns may reference earlier returns or user-supplied values.
- Order: login first; search before result-dependent actions; status-check tools only in "check" intents.
- Long-tail coverage: at least one turn must use a tool from the emphasis subset `{emphasis_set}`.
</design_constraints>

<provenance_tag_declaration>
For every required argument, declare `prov_tag[tool][param]` from one of:
- `initial_state`: value pre-exists in the initial-state summary (IDs, filenames, credentials).
- `prev_output`: value read from a prior tool's return.
- `self_create`: new user-supplied value (new filename, tweet text, search keyword).
- `prev_user_msg`: value already provided in an earlier user message.

Rules: every required param must appear; the first turn out of σ_0 may NOT use `prev_output` or `prev_user_msg`; login credentials must use `initial_state`.
</provenance_tag_declaration>

<inputs>
Available tools: `{tool_summaries}`
Initial-config summary: `{initial_state}`
</inputs>

<output>
Output ONLY `<fsm>JSON</fsm>` matching `{schema_hint}`. No explanatory text.
</output>

Figure 6: Prompt template for $\mathcal{A}_{\text{FSM}}$ (FSM skeleton synthesis).

$\mathcal{A}_{\text{plan}}$: Per-Call Argument Filling

<task>
You are filling parameters for a SINGLE tool call at turn `{turn_idx}`. Tool to call: `{func_name}`.
</task>

<inputs>
- Tool signature: description, required params, properties: `{tool_doc}`
- Declared Provenance Tags from Phase 1 FSM: `{prov_tag}`
- Previous turns history (executed calls and their returns): `{history}`
- Initial state (starting state; may be modified by earlier turns): `{initial_state}`
- Last attempt error (if refill): `{last_error}`
</inputs>

<output_format>
Output `<result>JSON</result>` with two keys:
- `args_for_exec`: the actual parameter values for execution. `null` is only acceptable for OPTIONAL params whose documentation explicitly states that `null` means "no filter / default behavior".
- `args_provenance`: per-argument provenance metadata. `src` MUST be one of:
  - `initial_state`: value from initial state (also set `config_path`)
  - `prev_output`: value from a prior turn's tool return (also set `ref_turn` and `ref_field`)
  - `self_create`: value you create in this turn
  - `prev_user_msg`: value the user provided in an earlier turn (also set `introduce_in_turn` $<$ current turn)
  - `fallback`: declared upstream is unavailable and this binding uses a recovery value; the resulting call must then be validated by execution (also set `fallback_from` to the original src)
</output_format>

<rules>
- If the declared source is unavailable, recover the argument from P_i or P_c when permitted. Record `fallback` whenever this recovery path replaces the declared source. Bind the recovery value and let the executor validate the resulting call.
- NEVER fabricate existing entities or credentials: IDs, tokens, usernames, passwords, or existing resource paths must be grounded in an executor-visible source.
- For `prev_user_msg`, `introduce_in_turn` must be strictly before the current turn.
- For `prev_output`, `ref_turn`/`ref_field` must identify a real prior return field.
</rules>

Figure 7: Prompt template for $\mathcal{A}_{\text{plan}}$ (per-call argument filling). The two-track output $(\boldsymbol{\theta}^{\text{exec}}, \boldsymbol{\theta}^{\text{prov}})$ corresponds to `args_for_exec` and `args_provenance`.

$\mathcal{A}_{\text{msg}}$: User-Message Synthesis

<task>
You are a multi-turn dialogue data generator. Given a tool-calling sequence (GT) for the current turn, generate the corresponding natural user message.
</task>

<inputs>
- Prior conversation history (the user said what, the assistant called which tools, what they returned): `{prior_text}`
- Current turn's expected tool calls (GT): `{turn_gt}`
- Per-argument source guide for this turn (the must-mention list `{must_mention}`, marker semantics below).
</inputs>

<must_mention_markers>
Express each parameter according to its source marker:
- `[NEW]`: explicitly state the value (e.g., filename, number, keyword).
- `[FROM_HISTORY]`: use a vague reference ("that file", "the ID you found earlier"); do NOT repeat the exact value, but the reference must be unambiguous.
- `[FROM_CONFIG]`: explicitly state the exact value verbatim. The assistant cannot read the initial state, so IDs, credentials, and token-like fields must appear character-for-character.
- `[FALLBACK_from_...]`: the original source was unavailable and a value was created; treat as `[NEW]` and state it explicitly.
</must_mention_markers>

<core_constraints>
1. **Assistant capability boundary**: the assistant can ONLY call tools, not reply with text. The user message is the only input driving its behavior; it must contain enough information for correct tool calls.
2. **Parameter completeness**: every parameter used this turn must be naturally expressed in the user message according to its marker.
3. **Semantic alignment**: the user message's semantic direction must match what the tools actually do (e.g., `unfollow` $\neq$ "follow", `rm` = "delete").
4. **Intent-shift handling**: if this turn's intent differs from prior turns, explicitly state the transition.
5. **History is fact**: write the message conditioned on the provided history; do not modify history.
6. **Full GT coverage**: if the GT lists n tool calls, the user message must naturally cover *every* call (chain them with "first ..., then ..., finally ..." when needed).
</core_constraints>

<style>
60% conversational + 40% instructional; 1–3 sentences; named values quoted; never reference tool internals.
</style>

<output>
`<msg>{"turn_idx": {turn_idx}, "user_msg": ". . . "}</msg>`
</output>

Figure 8: Prompt template for $\mathcal{A}_{\text{msg}}$ on user-message synthesis. The must-mention list is pre-rendered by the dispatcher of Appendix A.6 from $\boldsymbol{\theta}^{\text{prov}}$ and Δ_v, with each entry already tagged `[NEW]` / `[FROM_HISTORY]` / `[FROM_CONFIG]` / `[FALLBACK from ...]`.

$\mathcal{A}_{\text{msg}}$: Assistant Text Response (on no-tool-call turn)

<task>
You are the assistant in a multi-turn tool-calling conversation. Write the NEXT assistant reply body when this turn has no further tool calls.
</task>

<requirements>
1. Output ONLY natural language, no tool calls, no JSON, no placeholders.
2. If tasks have been completed, briefly summarize and ask if anything else is needed.
3. If the task cannot continue (missing tool, missing argument, permission), explain why politely and suggest next steps.
4. Match the language of the user.
5. Report errors honestly: never claim success when prior tool returns contain `error / fail / exception / empty / NULL`.
6. NEVER answer the requested tool-action from general knowledge, NEVER invent a result, NEVER claim the action was done unless prior tool messages prove it.
7. Blocking-reply rule: if later turns may supply more tools or missing arguments, this current reply must state what is missing rather than pre-computing the answer.
</requirements>

Figure 9: Prompt template for $\mathcal{A}_{\text{msg}}$ on assistant text responses (turns with $\mathcal{C}_k = \varnothing$).

Rewrite-Operator Hints (Ψ_{param}, Ψ_{func})

<MISS_PARAM_CLOSED_LOOP>
(injected into $\mathcal{A}_{plan}$ at the recovery turn $k^* + 1$)
The previous turn k^* should have called `{tool}` but lacked required parameters `{missing_params}`. The current turn's user message has supplied them. Call `{tool}` now with all required parameters (including the previously missing ones), unless the user's intent has clearly shifted, in which case call a more appropriate tool. Logical alignment with the user takes priority over closing the loop.
</MISS_PARAM_CLOSED_LOOP>

<TOOL_INJECTION>
(injected at the recovery turn $k^* + 1$)
The current turn is a tool-injection turn: the system has just registered new tools `{func_names}` into the visible toolkit. Prefer these newly registered tools when they match the user's intent. Even when `{question_k}` is the empty list `[]`, the ground truth must remain non-empty: extract the required parameters from the prior turn $k^* = k - 1$, whose unmet request triggered the injection.
</TOOL_INJECTION>

<MISSING_FUNCTION_REQUEST>
(injected at the request turn k^*, before the handoff message)
The current turn is a request turn: the user asks for tools `{func_names}` that are NOT yet in the visible function-doc. You MUST output `[]` for this turn, even if other tools are available; calling any tool here breaks the operator's semantics.
</MISSING_FUNCTION_REQUEST>

Figure 10: Hint templates injected by the rewrite operators Ψ_{param} and Ψ_{func}. Placeholders are filled at injection time.