SAGE: SEEDLESS AGENT TASK INDUCTION FROM ENVIRONMENTS FOR POST-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Training capable agents requires interaction trajectories grounded in executable environments, yet constructing such data for a new environment often requires manually designing tasks, organizing interactions, and verifying execution outcomes. We introduce SAGE, a framework for seedless construction of agent training data from environment specifications. SAGE generates requests from domain context, tool interfaces, user attributes, and initial states, without task-specific query seeds or reference solution paths. A persistent user generator and an agent engage in multi-turn interaction, while an isolated verifier checks candidate requests before execution and complete trajectories after execution. Rejected requests are removed and regenerated, and accepted trajectories are converted into supervised fine-tuning data. We use Qwen3.8-2.4T, Qwen3.8-Flash-Next, GLM5.2, and Kimi-K3 as teacher models to construct data for general tasks, tool interaction, and software engineering, and evaluate the resulting data against public-data baselines. Independent SFT runs on Qwen3.5-27B yield absolute score gains over the initial model of 1.11%, 9.25%, 0.86%, and 2.33% on General Avg, BFCL v4 Multi-turn, τ^3-bench, and SWE-bench Lite, respectively. The corresponding absolute gains over the best public-data baselines in our comparison are 10.82%, 19.75%, 7.50%, and 8.00%. These results show that SAGE can provide effective supervision for agent post-training across different environments. The datasets are available in anonymous repositories at https://huggingface.co/sage-submission.

1 INTRODUCTION

Executable environments provide tool interfaces, state transitions, and interaction feedback, but often lack the user requests and multi-turn demonstrations needed for post-training. Existing trajectories may differ from the target environment in tool definitions, permission constraints, and interaction rules, limiting their reuse across environments. Constructing supervision for a new environment therefore requires jointly addressing task generation, interaction organization, and trajectory verification. Existing instruction and tool-data synthesis methods provide effective supervision (Xu et al., 2024; Prabhakar et al., 2025; Shim et al., 2025), while the direct construction of complete training trajectories from environment specifications remains an open research problem. We ask how environment context can be used to generate executable requests and their interaction trajectories without task-specific query seeds or reference solution paths.

We propose SAGE, a framework for seedless construction of agent training data from environment specifications. Its inputs include domain context, user attributes, tool definitions, initial state, and generation controls. A user generator with persistent interaction history induces requests from these inputs. SAGE assigns user generation, agent execution, and trajectory verification to three roles with separate information access: the user generator produces requests and subsequent messages; the agent invokes tools using information accessible through its role-specific view; and the verifier evaluates candidate requests and complete trajectories. The initial request is fixed after verification and before execution. Subsequent user messages depend on accepted interactions and support clarification, confirmation, revision, handoff, and refusal. SAGE regenerates only requests rejected before execution. Executed trajectories are retained as training samples or audit records according to verification results, without rolling back execution state to retry the task.

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.

Accepted interaction episodes form the SAGE dataset, covering general tasks including code and mathematics, tool use and stateful transactions, and repository-level software engineering. We use Qwen3.8-2.4T, Qwen3.8-Flash-Next (Qwen Team, 2026), GLM5.2 (GLM-5 Team, 2026), and Kimi-K3 (Kimi Team, 2026) as teachers, and conduct independent SFT experiments with Qwen3.5-27B as the student. Each branch starts from the same initial model and is compared against public-data baselines, with training data sizes, configurations, and evaluation coverage reported. A data-scale ablation additionally uses Qwen3.5-4B to examine downstream performance at two model scales. We further report dataset statistics, verifier agreement, and data-construction costs.

Our main contributions are:

- SAGE, a framework that generates user requests and interaction trajectories from environment specifications without task-specific query seeds or reference solution paths;
- a dataset spanning multiple task domains, constructed through role-specific information access, request verification, and trajectory verification, with source records for the resulting SFT samples;
- comparisons against public datasets on a 27B student model and a BFCL data-scale ablation that examine target-task gains and the effects of training-data scale.

2 Related Work

Instruction data synthesis. Generating instructions and responses with language models is an important approach to scaling supervised fine-tuning data. Self-Instruct generates and filters instructions from a small set of seed tasks (Wang et al., 2023). Magpie further exploits aligned language models to elicit instructions without an explicit input query (Xu et al., 2024). From expanding seed tasks to autonomous request generation, this line of work explores ways to reduce manual task design and broaden instruction coverage.

Tool-use and multi-turn interaction data. Tool-data synthesis must account for user requests, tool definitions, and invocation sequences. APIGen and ToolACE combine tool-use data generation with verification (Liu et al., 2024b;a); APIGen-MT and ToolACE-MT extend synthesis to multi-turn interactions (Prabhakar et al., 2025; Zeng et al., 2025). ToolDial organizes dialogue around API dependencies and user clarification (Shim et al., 2025), incorporating exchanges in which users supply missing information needed for subsequent tool calls. This connects dialogue progression with the information requirements of API execution. ToolMind uses tool graphs and turn-level filtering (Yang et al., 2025a), while user simulation advances dialogue through incremental feedback (Cho et al., 2026). These studies enrich tool-use supervision through call structure, tool dependencies, and user feedback, extending data construction to sustained multi-turn interaction.

Environment supervision and agent post-training. Executable environments provide interaction trajectories and outcome feedback for agent training. SWE-Gym combines software repositories with executable tests to train agents and verifiers (Pan et al., 2024). CuES addresses environments without predefined tasks: curiosity-driven exploration and task-schema induction generate executable tasks without handcrafted task seeds for agent reinforcement learning (Mai et al., 2025). To improve behavior during interaction, AAPA uses sentence-level adversarial anchors from offline expert responses in post-training objectives (Qian et al., 2025), while Erase to Improve trains models to erase erroneous search or reasoning steps and regenerate subsequent content (Wang et al., 2026). These works explore environment supervision, autonomous task generation, expert signals, and process correction as ways to improve agents through interaction experience.

3 Methodology

3.1 Environment specification and seedless induction

SAGE represents an environment specification as

$$z = (s_{\text{domain}}, \rho_{\text{user}}, \mathcal{T}, \mathcal{E}_0, c), \tag{1}$$

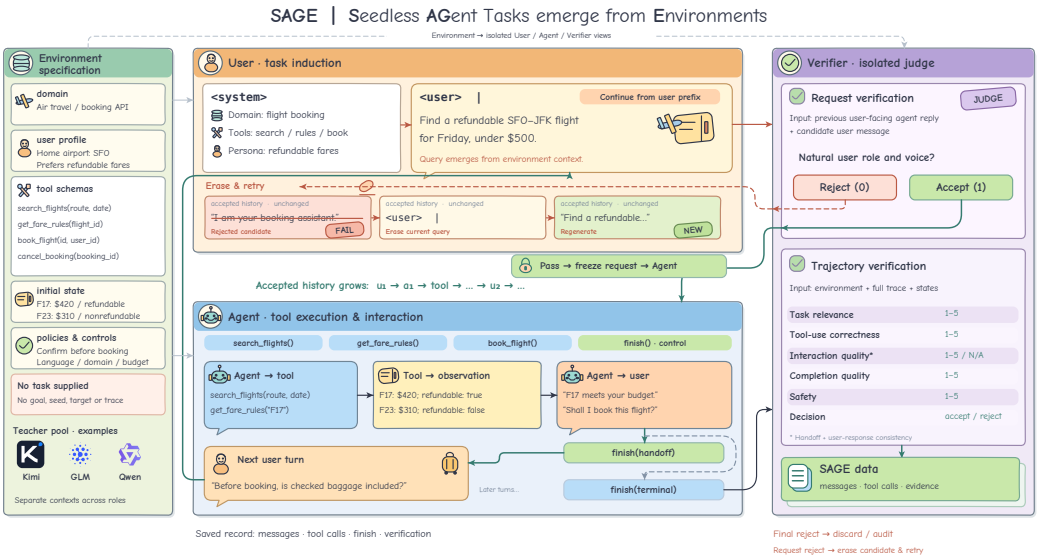

Figure 1: **SAGE constructs agent supervision from executable environments.** An environment specification supplies separate User, Agent, and Verifier views. User requests emerge from environment context; invalid candidates are erased and regenerated while accepted history is retained. Passing requests are fixed before tool execution. Handoff continues the interaction, and termination triggers episode verification. Accepted trajectories form the SAGE dataset.

where s_{domain} is domain or repository context, ρ_{user} contains optional user attributes and permissions, $\mathcal{T}$ is the set of domain tools, $\mathcal{E}_0$ is the initial environment state, and c contains controls such as language, difficulty, and generation budget. Seedless inputs exclude task-specific goals, query seeds, target states, and reference trajectories.

The seedless condition is assessed from the original input content and its provenance. User attributes containing concrete task instructions or a supplied issue description constitute task seeds, even when rewritten or presented as background information. The trajectory verifier evaluates accepted user requests using the complete interaction trace and environment information.

3.2 Isolated Role Views

User view. At user turn u, the user generator receives

$$v_U^{(u)} = P_U(z, h_u^{\text{vis}}, \widetilde{\mathcal{E}}_u), \qquad m_u \sim M_U(\cdot \mid v_U^{(u)}). \tag{2}$$

Here, h_u^{vis} denotes the accepted user-visible interaction history before the message at turn u is generated; the superscript vis means user-visible. This history contains only user messages and the agent's replies to the user. Tool calls executed by the agent, tool results, and internal traces of its interaction with the environment are invisible to the user generator. The term $\widetilde{\mathcal{E}}_u$ denotes the current environment state visible to the user within their permissions, without exposing the internal execution that produced it. The tilde distinguishes this visible state from the full environment state $\mathcal{E}_u$. Thus, h_u^{vis} describes previous interactions, whereas $\widetilde{\mathcal{E}}_u$ describes currently accessible state information; together they provide context for user-message generation.

The view-construction function P_U organizes the environment specification z, visible history, and visible state according to the user's access permissions, yielding the input context $v_U^{(u)}$. The user model M_U samples m_u from the conditional message distribution $M_U(\cdot \mid v_U^{(u)})$. The first message $m_1 = q$ is the initial request; subsequent messages clarify information, confirm actions, revise requests, or accept alternatives. User attributes, permissions, and behavior remain consistent within an episode. Appendix C.1 gives the user-generation contexts for each branch.

Agent view. At execution step k of user turn u, the agent receives

$$v_A^{(u,k)} = P_A(z, h_{u,k}), \qquad \mathcal{T}_A = \mathcal{T} \cup \{\texttt{finish}\}. \tag{3}$$

Here, k indexes agent execution steps within the turn, and $h_{u,k}$ is the history accessible to the agent before step k, including previous user messages, agent responses, tool calls, and their results. The function P_A constructs the agent's input context $v_A^{(u,k)}$ from z and $h_{u,k}$. The available tool set $\mathcal{T}_A$ consists of the domain tools $\mathcal{T}$ and the control tool `finish`.

The agent view contains user messages, declared tools, environment observations, and public constraints. It excludes undisclosed user attributes, the full hidden state, and reference answers. The `finish` tool delivers a reply to the user and controls handoff or termination. For tool/transaction and long-trajectory tasks, it uses three principal fields: `summary` contains the complete natural-language reply; `status` specifies handoff (`handoff`) or termination (`success`/`failed`); and `handoff_reason` specifies the reason for handoff and is supplied only for that case. General dialogue uses a simplified form with `response` for the reply and `next_actor` for continuation or termination. Agent input structures and tool definitions appear in Appendices C.2 and C.4, respectively.

Verifier view. SAGE verifies requests after generation and trajectories after interaction terminates. Request verification produces a binary decision, whereas trajectory verification produces a multidimensional quality assessment. For candidate request $m_u^{(j)}$, the request verifier receives only the preceding user-facing agent reply and the candidate message, and assesses whether the candidate is consistent with the role and voice of an end user:

$$b_u^{(j)} = V_{\text{request}}\big(a_{u-1}^{\text{vis}}, m_u^{(j)}\big) \in \{0, 1\}. \tag{4}$$

The index j denotes the candidate attempt at turn u, and $m_u^{(j)}$ is the corresponding candidate request. The reply a_{u-1}^{vis} is the preceding agent reply extracted from the accepted user-visible history; the first turn uses a placeholder indicating that no preceding reply exists. The request verification function V_{request} returns an acceptance variable: $b_u^{(j)} = 1$ accepts the candidate, whereas $b_u^{(j)} = 0$ rejects it and triggers regeneration. This variable maps the Guard's raw classifier output of `0` to acceptance and `1` to rejection.

Request verification identifies role confusion, such as answering as the assistant, narrating a third-party task, or fabricating tool calls and results. Natural requests, clarifications, confirmations, corrections, and additional information are valid user expressions. This check does not assess factual correctness, executability in the environment, or task completion, and topical overlap with the preceding reply alone is not grounds for rejection. Appendix C.3 provides the decision rules and prompt.

After interaction terminates, the trajectory verifier assesses the complete execution:

$$\mathcal{J} = V_{\text{episode}}\big(P_V(z), h, \mathcal{E}_0, \mathcal{E}'\big). \tag{5}$$

Here, P_V constructs the trajectory verifier view, and $P_V(z)$ contains the environment information accessible to it. The history h is the complete terminal interaction trace, including user messages, agent replies, tool calls, tool results, and handoff events; $\mathcal{E}_0$ and $\mathcal{E}'$ are the initial and final environment states. The function V_{episode} performs trajectory verification, and $\mathcal{J}$ denotes its assessment. For scoring dimensions and decision rules, see Appendix C.3. The verifier can inspect the complete execution evidence, including tool calls and environment feedback invisible to the user generator.

3.3 Request Acceptance and Interaction

Algorithm 1 summarizes request acceptance and trajectory construction. At each user turn, SAGE generates and verifies candidate requests until one passes verification or the attempt limit is reached. Rejected candidates are removed from the generator's continuation context, and hidden verification details are not returned to the user generator or agent. An accepted request is fixed before delivery to the agent. Since candidate filtering precedes tool execution, request regeneration does not roll back tool-execution state.

The agent uses domain tools to execute the request and invokes `finish` to request handoff or termination. At handoff, the user generator receives only user-visible replies and accepted interaction

Algorithm 1 SAGE request induction and episode construction

Require: Environment specification z; maximum turns $U_{\max}$; attempt limit J

$h \leftarrow \varnothing$; initialize environment from z
for $u = 1, \ldots, U_{\max}$ **do**
  Construct h^{vis} and $\widetilde{\mathcal{E}}$ from accepted history and current state
  passed $\leftarrow$ false
  for $j = 1, \ldots, J$ **do**
    $m \sim M_U(\cdot \mid P_U(z, h^{\text{vis}}, \widetilde{\mathcal{E}}))$
    Extract a_{u-1}^{vis} from h^{vis} (use a placeholder on the first turn)
    $b \leftarrow V_{\text{request}}(a_{u-1}^{\text{vis}}, m)$
    if $b = 1$ **then**
      Fix m; passed $\leftarrow$ true; **break**
    else
      Archive candidate and evidence; remove candidate from continuation
    end if
  end for
  if not passed **then**
    return rejected episode
  end if
  Execute agent on (h, m) until `finish`, yielding $\tau, \mathcal{E}'$
  Append (m, τ) to h; advance to state $\mathcal{E}'$
  if `finish` is terminal **then**
    $\mathcal{J} \leftarrow V_{\text{episode}}(P_V(z), h, \mathcal{E}_0, \mathcal{E}')$
    if $F_\eta(\mathcal{J}, h) = 1$ **then**
      return h
    else
      return rejected episode
    end if
  end if
end for
return episode rejected for exceeding the turn budget

history to generate the next message. SAGE records the execution throughout, and the complete trajectory undergoes episode-level verification after termination.

SAGE rejects an episode when the request-attempt or interaction-turn limit is reached, or an unrecoverable execution error occurs. Agent execution failure does not trigger request regeneration. Request-level regeneration operates before execution, whereas episode-level verification operates after execution. Rejected candidates and complete episodes are retained in audit records and excluded from SFT data.

3.4 Episode acceptance and supervision compilation

Trajectory quality assessments and training-sample acceptance decisions are recorded separately. SAGE uses the filtering function

$$a = F_\eta(\mathcal{J}, h) \in \{0, 1\} \tag{6}$$

to determine whether a trajectory is included in training, where $a = 1$ denotes acceptance and η specifies score thresholds and structural checks. Filtering combines the trajectory assessment with structural validity; see Appendix C.3 for detailed rules.

SFT data are exported from accepted agent interaction trajectories, retaining user messages, domain tool calls and results, and replies to the user. The export removes `finish` from the tool definitions and replaces each invocation with its user-facing natural-language content as an ordinary assistant message. Status, reason, and supplementary evidence fields are not included in this message. User messages, system context, and domain tool results provide conditioning context; assistant replies and domain tool calls are prediction targets. The internal generation and decision processes of the user generator and verifier are excluded from the agent's SFT trajectory. Appendix D shows exported

samples from all four data subsets. The loss is

$$\mathcal{L}_{\mathrm{SFT}} = -\frac{1}{\sum_i w_i} \sum_i w_i \log p_\theta(x_i \mid x_{<i}), \tag{7}$$

where x_i is token i in the serialized interaction, $x_{<i}$ is its preceding context, θ denotes model parameters, and $w_i \in \{0, 1\}$ is the target-token mask. Only assistant natural-language replies and domain tool calls contribute to the loss.

For general conversations without executable state, verification uses conversational quality and factual consistency criteria. Tool and code environments additionally record tool execution, initial state, state changes, environment identifiers, and verification evidence. Accepted training samples and rejected audit records are stored separately.

4 EXPERIMENTS

4.1 SAGE DATA CONSTRUCTION AND SCALE

SAGE data comprise three categories: general, tool/transaction, and long-trajectory data, covering open-domain dialogue, tool use and stateful interaction, and repository-level software engineering, respectively. User generation and agent execution use teachers including Qwen3.8-2.4T, Qwen3.8-Flash-Next (Qwen Team, 2026), GLM5.2 (GLM-5 Team, 2026), and Kimi-K3 (Kimi Team, 2026). The same Qwen3.6-27B model serves as the judge for request-level verification and trajectory-level assessment. All three categories follow the request-generation, interaction-control, and trajectory-filtering procedure in Section 3.

General data. The general branch uses a concise assistant-role instruction (*You are a helpful/-math/coding assistant*) and introduces `finish` to control replies, handoff, and termination. The user generator and agent construct interactions for general question answering, reasoning, coding, and mathematics. Combining data generated by Kimi-K3, Qwen3.8-2.4T, and Qwen3.8-Flash-Next yields **235,292 samples**.

Tool/transaction data. This branch constructs requests and execution trajectories from BFCL v4 (Patil et al., 2025) and τ^3-bench (Barres & Shi, 2026) tool interfaces and environments, using GLM5.2 for generation. BFCL covers multi-turn function calling, while τ^3-bench covers transactional interaction involving environment-state changes. The branch contains **33,913 samples**: **19,297** from BFCL and **14,616** from τ^3-bench. The τ^3-bench data comprise 6,562 Airline samples and 8,054 Retail samples.

Long-trajectory data. This branch uses the Hermes framework with randomly selected GitHub repositories as interaction environments and Qwen3.8-Flash-Next for generation. Requests are constructed from repository context, and the agent completes tasks through code search, editing, and execution. Of 629 candidate trajectories in this batch, **498 SFT samples** remain after filtering. Dataset sizes and sequence-length statistics are given in Appendix A.1, and SAGE data composition in Appendix A.2.

4.2 PUBLIC-DATA BASELINES AND TRAINING SETUP

We select three representative public datasets for each category, yielding nine baselines: Magpie-Pro-MT (Xu et al., 2024), UltraChat (Ding et al., 2023), and Infinity-Instruct (the Infinity Gen view) (Li et al., 2025) for general tasks; ToolACE (Liu et al., 2024a), xLAM 60K (Liu et al., 2024b), and APIGen-MT (Prabhakar et al., 2025) for tool/transaction tasks; and SWE-Gym (Pan et al., 2024), R2E-Gym (Jain et al., 2025), and SWE-smith (Yang et al., 2025b) for long-trajectory tasks. Table 1 reports each dataset separately.

All main experiments start from the same initial Qwen3.5-27B checkpoint (denoted Base) and independently perform SFT on a public dataset or a SAGE branch. Context lengths are 4K, 16K, and 64K for general, tool/transaction, and long-trajectory tasks, respectively, with a global batch size of 64 and one training epoch. Each public general baseline uses 100K samples. ToolACE and xLAM

Category	Training dataset	Inputs	General Avg	BFCL v4 MT	τ^3-bench Overall	SWE-Lite
Base	Qwen3.5-27B	—	84.45	54.13	82.90	52.67
General	Magpie-Pro-MT	100,000	71.28 (−13.17)	—	—	—
	UltraChat	100,000	57.44 (−27.01)	—	—	—
	Infinity-Instruct (Infinity Gen)	100,000	74.74 (−9.71)	—	—	—
	SAGE-General	**100,000**	**85.56 (+1.11)**	—	—	—
Tool/transaction	ToolACE	10,000	—	43.63 (−10.50)	43.89 (−39.01)	—
	xLAM	10,000	—	0.25 (−53.88)	1.44 (−81.46)	—
	APIGen-MT	4,801	—	41.00 (−13.13)	76.26 (−6.64)	—
	SAGE-BFCL	**10,000**	—	**63.38 (+9.25)**	—	—
	SAGE-Tau	**10,000**	—	—	**83.76 (+0.86)**	—
Long-trajectory	SWE-Gym	491	—	—	—	36.00 (−16.67)
	R2E-Gym	3,231	—	—	—	31.33 (−21.34)
	SWE-smith	5,000	—	—	—	47.00 (−5.67)
	SAGE-Hermes	**498**	—	—	—	**55.00 (+2.33)**

Table 1: Main results on Qwen3.5-27B, grouped by data category. Inputs count training samples before length filtering. Scores and absolute differences from Base (in parentheses) are reported in %; dashes indicate unreported results. Every row except Base represents an independently fine-tuned checkpoint.

each use 10K samples, while APIGen-MT uses all cleaned samples. Long-trajectory baselines use the full dataset up to a maximum of 5K samples. Samples whose complete sequence exceeds the context limit are discarded.

The main SAGE experiments use 100K samples drawn from SAGE-General for training, the 10K subset from the BFCL data-scale ablation, a 10K τ^3-bench subset, and 498 Hermes samples. For learning rates and other training settings, see Appendix B.1. Fixing sample counts and training epochs controls some training conditions, although assistant-target-token budgets are not strictly matched across sources.

4.3 Evaluation and Main Results

We evaluate general capabilities through OpenCompass on 11 benchmarks: MMLU, MMLU-Pro, GPQA Diamond, HumanEval, MBPP, GSM8K, MATH, TheoremQA, IFEval, CMMLU, and C-Eval. General Avg is the equal-weight mean of these scores; see Appendix B.2 for individual results. Tool/transaction evaluation uses BFCL v4 Multi-turn (Patil et al., 2025) and τ^3-bench three-domain Overall (Barres & Shi, 2026). The latter covers Airline, Retail, and Telecom; the training-domain composition is described in Section 4.1. Long-trajectory evaluation uses SWE-bench Lite.

On General Avg, BFCL v4 Multi-turn, τ^3-bench Overall, and SWE-bench Lite, the corresponding SAGE branches achieve absolute score gains over Base of 1.11%, 9.25%, 0.86%, and 2.33%, respectively. The corresponding absolute gains over the best public-data baseline for each metric are 10.82%, 19.75%, 7.50%, and 8.00%. These results show that, under the reported training configurations, SAGE data improve downstream performance on the corresponding tasks.

5 Ablations and Analysis

We examine the effects of training-data scale and verifier selection, and report construction time and acceptance rates for tool/transaction data. The data-scale ablation examines downstream gains from BFCL data on 4B and 27B students. The judge ablation compares Qwen3.6-27B and GLM5.2 decisions on the same samples to inform verifier selection.

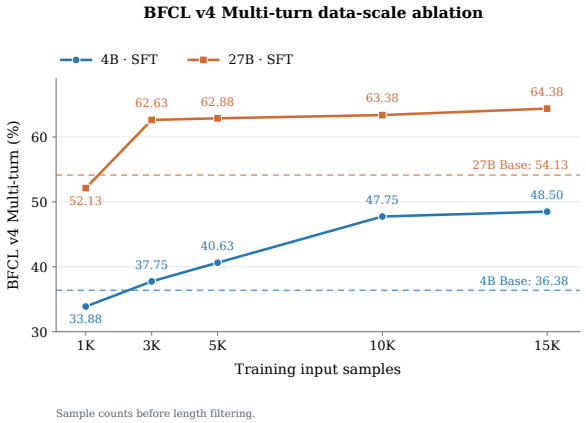

Figure 2: Data-scale ablation on BFCL v4 Multi-turn. Solid lines: SFT; dashed lines: Base.

Qwen3.6-27B	GLM5.2	Samples
accept	accept	689
accept	reject	22
reject	accept	97
reject	reject	164
Total	—	**972**

Table 2: Joint decisions of Qwen3.6-27B and GLM5.2 on 972 BFCL samples.

5.1 Data-scale ablation

We construct training subsets of 1K, 3K, 5K, 10K, and 15K samples from the same BFCL source, perform SFT on Qwen3.5-4B and Qwen3.5-27B, and evaluate on BFCL v4 Multi-turn. Both groups use a 16K context, a global batch size of 64, and one training epoch. Within each model size, training settings other than the amount of data remain fixed.

Figure 2 shows that both models exceed their respective Base scores at 3K samples and improve as data size increases. At 10K, the 27B model reaches 63.38, a gain of 9.25% over Base; this checkpoint is used in the main experiment. At 15K, it reaches 64.38, a gain of 10.25%. The 4B score increases from 33.88 at 1K to 48.50 at 15K, exceeding Base by 12.12%. The largest 27B improvement occurs between 1K and 3K, while 4B continues to gain between 3K and 10K. With the number of epochs fixed, larger datasets also entail more optimization steps; the experiment therefore captures the combined effect of additional data and its associated training budget.

5.2 Judge ablation

To examine the feasibility of using a smaller model for data verification, we select 1K BFCL samples and evaluate them with Qwen3.6-27B and GLM5.2. Both judges yield parseable decisions for the same 972 samples. Table 2 reports their joint decisions. All comparisons below use these 972 samples, excluding the remaining 28.

The judges agree on 853 samples, yielding an overall agreement rate of **87.76%**. Qwen3.6-27B accepts 711 samples (73.15%), whereas GLM5.2 accepts 786 (80.86%). Among the samples accepted by Qwen3.6-27B, 689 are also accepted by GLM5.2, corresponding to **96.91%**. Disagreements are concentrated in the 97 samples rejected by Qwen3.6-27B but accepted by GLM5.2, indicating a lower acceptance rate for Qwen3.6-27B on this batch. These results show high agreement with GLM5.2 on the samples accepted by Qwen3.6-27B. This agreement supports adopting Qwen3.6-27B as the default judge for verification and filtering during subsequent data construction. Section 5.3 reports its measured evaluation time.

5.3 Time cost and acceptance rates

Data volume, agent execution speed, and generation models differ across branches. We therefore report batch-level generation, evaluation, and acceptance statistics for the τ^3-bench and BFCL tool/-transaction batches, each targeting 30K trajectories. Both use Qwen3.6-27B as the trajectory judge with a concurrency of 8. Each trajectory is evaluated three times, and the intersection of the three acceptance sets is retained: a trajectory must pass all three evaluations to proceed. Accepted counts in Table 3 are taken from the final full SFT exports. Evaluation coverage is 99.98% for τ^3-bench and 99.86% for BFCL. The 5 and 42 samples without valid evaluation results, respectively, are not

Batch	Generated	Evaluated	Accepted	Rate	Generation	Judging
τ^3-bench 30K	29,883	29,878	14,616	48.92%	≈120 h (5 d)	28 h 51 min
BFCL 30K	29,285	29,243	19,297	65.99%	≈156 h (6.5 d)	26 h 46 min

Table 3: Time cost and acceptance rates for tool/transaction data. Generated counts are actual candidate trajectories; evaluated counts are trajectories with completed, parseable evaluations. Acceptance rates use the evaluated count as the denominator. Generation and judging times are measured separately as batch wall-clock times.

Statistic	Per-trajectory time
Mean	≈15 min 46 s
Median	12 min 19 s
P90	31 min 4 s
Minimum	57 s
Maximum	66 min 43 s

Table 4: Execution times for 200 successful Hermes trajectories (individual durations, not batch wall-clock times).

Statistic	Result
Total Guard requests	2,721
Mean time per request	≈0.118 s
Mean Guard time per trajectory	1.61 s
Total Guard time	5 min 22 s
Fraction of generation time	≈0.17%

Table 5: Guard overhead for the same 200 trajectories. Cumulative review time is compared with cumulative generation time.

counted as content-based rejections. Final acceptance rates are **48.92%** and **65.99%**; relative to all generated trajectories, the retained fractions are 48.91% and 65.89%. These fractions reflect retention after intersecting the three evaluation decisions and applying subsequent filtering. Trajectory generation accounts for most of the measured time. Including all three repeated evaluations, judging takes 28 h 51 min for τ^3-bench and 26 h 46 min for BFCL, approximately 24.04% and 17.16% of their generation times. Under this configuration, the 27B judge completes all three evaluations of each batch of nearly 30K trajectories within 30 hours.

We sample 200 successful trajectories from the Hermes branch and measure the execution time of individual tasks. Mean execution time is approximately **15 min 46 s**, and the median is 12 min 19 s. Table 4 summarizes the distribution. The mean exceeds the median, P90 is nearly twice the mean, and the longest trajectory takes more than an hour, indicating substantial variation in execution time. Quantiles therefore complement the mean when characterizing the time required to generate long trajectories. These statistics cover only the sampled successful trajectories.

For the same 200 trajectories, we also measure the time spent by the Qwen3.6-27B Guard on reviewing candidate user requests. The Guard implements the request-level verifier in Section 3. It processes 2,721 review requests, averaging approximately 0.118 s per request. Summed review time is 5 min 22 s, or 1.61 s per trajectory, as summarized in Table 5. Request-level Guard reviews account for approximately 0.17% of cumulative generation time, indicating low time overhead for candidate verification before execution in these long-trajectory tasks. This measurement covers candidate user-request reviews and excludes trajectory-level quality evaluation after interaction terminates.

6 Conclusion

We present SAGE, a framework that generates requests from environment specifications, organizes multi-turn interaction, and verifies complete trajectories to construct supervision for agent post-training. SAGE separates information access for user generation, agent execution, and trajectory verification, checking candidate requests before execution and filtering trajectories afterward. Main experiments with Qwen3.5-27B show improvements on several target tasks, with gains depending on data scale, sample composition, and training configuration. Future work will examine generalization and construction efficiency through component ablations with matched supervision budgets, cross-environment evaluation, and comprehensive cost accounting.

LIMITATIONS

Task coverage in SAGE depends on the environment specification and teacher model, potentially favoring familiar requests over rare user needs or complex constraints; request-level role and voice checks do not establish task diversity or representativeness. Trajectory filtering also depends on verifier capabilities and available execution evidence. The inter-judge agreement in Section 5.2 measures consistency rather than accuracy, and repeated evaluation cannot eliminate shared systematic errors or replace independent correctness checks when execution feedback is incomplete. Finally, long-trajectory generation remains constrained by model inference and environment execution costs. The 200 successful Hermes trajectories in Section 5.3 require 15 min 46 s on average, with substantial variation, so low request-level Guard overhead does not imply low end-to-end generation cost.

AI USE STATEMENT

We use large language models to generate user requests and agent interaction trajectories, perform request-level verification, and assess trajectory quality. The models, input information, and filtering criteria are described in Sections 3–5 and Appendix C. We also use generative AI tools to assist with code development and debugging, translation, and language editing.

REPRODUCIBILITY STATEMENT

Section 3 describes request generation, interaction control, trajectory filtering, and SFT export in SAGE. Sections 4–5 report data sources, training input sizes, evaluation settings, and ablation configurations. Appendix A provides dataset sizes and sequence-length statistics; Appendix B reports training hyperparameters and detailed general-capability results; Appendix C presents prompts, verification rules, and control-tool definitions; and Appendix D provides SFT examples from the four data branches. The four complete datasets are publicly available through anonymous repositories at https://huggingface.co/sage-submission. These releases contain the data pools before training subsampling and length filtering; training input sizes and sequence-length limits are specified in Section 4.2 and Appendix B.1. We also plan to release the code.

REFERENCES

Victor Barres and Ben Shi. τ^3-Bench: Fixing airline + retail. Sierra Research, February 2026. URL https://taubench.com/blog/tau3-task-fixes.html.

Jungho Cho, Minbyul Jeong, and Sungrae Park. User-oriented multi-turn dialogue generation with tool use at scale. *arXiv preprint arXiv:2601.08225*, 2026. URL https://arxiv.org/abs/2601.08225.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3029–3051. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.183. URL https://aclanthology.org/2023.emnlp-main.183/.

GLM-5 Team. GLM-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026. URL https://arxiv.org/abs/2602.15763.

Naman Jain, Jaskirat Singh, Manish Shetty, Tianjun Zhang, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural environment generation and hybrid verifiers for scaling open-weights SWE agents. In *Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=7evvwwdo3z.

Kimi Team. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026. URL https://arxiv.org/abs/2607.24653.

Jijie Li, Li Du, Hanyu Zhao, Bo-wen Zhang, Liangdong Wang, Boyan Gao, Guang Liu, and Yonghua Lin. Infinity Instruct: Scaling instruction selection and synthesis to enhance language models. *arXiv preprint arXiv:2506.11116*, 2025. URL https://arxiv.org/abs/2506.11116.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. ToolACE: Winning the points of LLM function calling. *arXiv preprint arXiv:2409.00920*, 2024a. URL https://arxiv.org/abs/2409.00920.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. APIGen: Automated pipeline for generating verifiable and diverse function-calling datasets. *arXiv preprint arXiv:2406.18518*, 2024b. URL https://arxiv.org/abs/2406.18518.

Shinji Mai, Yunpeng Zhai, Ziqian Chen, Cheng Chen, Anni Zou, Shuchang Tao, Zhaoyang Liu, and Bolin Ding. CuES: A curiosity-driven and environment-grounded synthesis framework for agentic RL. *arXiv preprint arXiv:2512.01311*, 2025. URL https://arxiv.org/abs/2512.01311.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. *arXiv preprint arXiv:2412.21139*, 2024. URL https://arxiv.org/abs/2412.21139.

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 *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392. PMLR, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

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. APIGen-MT: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *arXiv preprint arXiv:2504.03601*, 2025. URL https://arxiv.org/abs/2504.03601.

Faqiang Qian, Kang An, Weikun Zhang, Ziliang Wang, Xuhui Zheng, Liangjian Wen, Yong Dai, Mengya Gao, and Yichao Wu. AAPA: Adversarially anchored preference alignment for post-training of large language models. *arXiv preprint arXiv:2509.25148*, 2025. URL https://arxiv.org/abs/2509.25148.

Qwen Team. On the design of Qwen3.8-Next architecture: Evaluation, efficiency, and training stability. Technical Report arXiv:2608.30320, Alibaba Group, August 2026. URL https://arxiv.org/abs/2608.30320.

Jeonghoon Shim, Gyuhyeon Seo, Cheongsu Lim, and Yohan Jo. ToolDial: Multi-turn dialogue generation method for tool-augmented language models. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2503.00564.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023. URL https://arxiv.org/abs/2212.10560.

Ziliang Wang, Kang An, Xuhui Zheng, Faqiang Qian, Weikun Zhang, Jialu Cai, Cijun Ouyang, Yuhang Wang, and Yichao Wu. Erase to improve: Erasable reinforcement learning for search-augmented LLMs. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/pdf/2510.00861v2.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. *arXiv preprint arXiv:2406.08464*, 2024. URL https://arxiv.org/abs/2406.08464.

Chen Yang, Ran Le, Yun Xing, Zhenwei An, Zongchao Chen, Wayne Xin Zhao, Yang Song, and Tao Zhang. ToolMind technical report: A large-scale, reasoning-enhanced tool-use dataset. *arXiv preprint arXiv:2511.15718*, 2025a. URL https://arxiv.org/abs/2511.15718.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025b. URL https://papers.nips.cc/paper_files/paper/2025/hash/8b86cf5ace600c48fd188efbb8dedec8-Abstract-Datasets_and_Benchmarks_Track.html.

Xingshan Zeng, Weiwen Liu, Lingzhi Wang, Liangyou Li, Fei Mi, Yasheng Wang, Lifeng Shang, Xin Jiang, and Qun Liu. ToolACE-MT: Non-autoregressive generation for agentic multi-turn interaction. *arXiv preprint arXiv:2508.12685*, 2025. URL https://arxiv.org/abs/2508.12685.

A Data Statistics

A.1 Dataset sizes and sequence-length distributions

Table A.1 summarizes the sizes and sequence-length distributions of the public baselines and the SAGE data branches. Statistics cover the complete data pools before training subsampling and length filtering. Lengths are computed using the Qwen3.5 tokenizer and chat template, including system prompts, tool definitions, and complete interactions. Separately provided reasoning traces are excluded from the SAGE general-data statistics, consistent with the non-thinking SFT setting.

Category / Dataset	Samples	Mean	P90	P99	Maximum	Retention
General / Magpie-Pro-MT	300,000	1,170.76	1,486	1,825	8,512	4K: 99.989%
General / UltraChat	207,865	1,206.94	1,971	3,136	4,695	4K: 99.9577%
General / Infinity-Instruct (Gen)	1,456,927	766.10	1,334	2,839	100,162	4K: 99.5865%
General / SAGE-General	**235,292**	**594.86**	**1,104**	**2,702**	**15,517**	**4K: 99.6863%**
Tools / ToolACE	10,848	898.18	1,368	2,163	5,033	16K: 100%
Tools / xLAM 60K	59,740	622.07	918	1,314	3,588	16K: 100%
Tools / APIGen-MT	4,801	6,800.89	8,149	10,284	18,675	16K: 99.9583%
Tools / SAGE-BFCL	**19,297**	**6,782.25**	**8,022**	**37,131**	**109,742**	**16K: 98.2795%**
Tools / SAGE-Tau	**14,616**	**6,689.86**	**8,768**	**13,320**	**30,500**	**16K: 99.6305%**
Long / SWE-Gym	491	19,048.26	32,413	67,427	85,492	64K: 98.7780%
Long / R2E-Gym	3,231	14,606.66	23,936	31,846	34,299	64K: 100%
Long / SWE-smith	26,076	29,582.93	56,601	95,688	252,047	64K: 93.6800%
Long / SAGE-Hermes	**498**	**29,978.22**	**59,901**	**120,597**	**141,567**	**64K: 91.7671%**

Table A.1: Dataset sizes and sequence lengths in tokens. P90 and P99 denote the 90th and 99th percentiles. Retention is the fraction of samples within the corresponding context limit: 4,096 tokens for general data, 16,384 for tool/transaction data, and 65,536 for long trajectories.

A.2 Data composition

SAGE comprises 269,703 samples across four branches: general dialogue, BFCL tool interaction, τ^3-bench transactional interaction, and Hermes long trajectories. Figure A.1 shows their counts and proportions, using the same data scope as Table A.1.

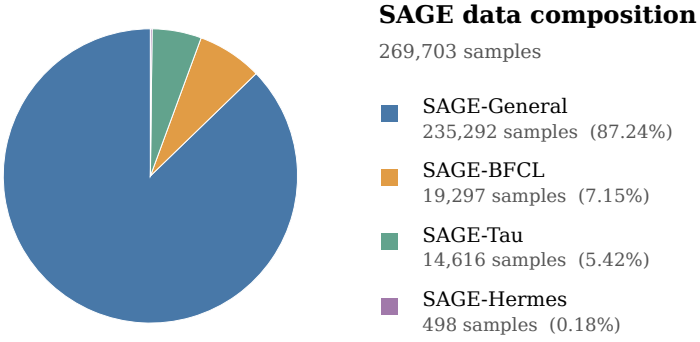

Figure A.1: Composition of SAGE data. Sector areas represent each branch's share of all samples; the legend reports counts and percentages. Statistics use the complete dataset before training subsampling and length filtering.

B Training Details

B.1 Training setup

Each experiment independently initializes from the Qwen3.5 checkpoint of the corresponding size (denoted Base) and uses SFT to learn assistant replies and tool calls. The vision encoder and cross-modal connector remain frozen during training. Table B.1 summarizes the shared hyperparameters.

Hyperparameter	Setting
Learning-rate schedule	Cosine decay
Warmup ratio	5%
Weight decay	0.1
Gradient clipping threshold	1.0
Global batch size	64
Training epochs	1
Numerical precision	BF16

Table B.1: SFT hyperparameters. The 27B main experiments and data-scale ablation use a peak learning rate of 1×10^{-4}; the 4B data-scale ablation uses 1×10^{-5}. The table lists the remaining shared settings.

Maximum sequence lengths are 4,096, 16,384, and 65,536 tokens for general, tool/transaction, and long-trajectory tasks, respectively; longer samples are removed. Training uses a non-thinking chat format and applies the supervised loss only to assistant replies and tool calls. Each public general dataset contributes 100,000 samples. ToolACE and xLAM each contribute 10,000, while APIGen-MT contributes all 4,801 cleaned samples. Software-engineering baselines use all samples up to a maximum of 5,000. SAGE-General uses 100,000 samples drawn from its full data pool; SAGE-BFCL, SAGE-Tau, and SAGE-Hermes use 10,000, 10,000, and 498 samples, respectively. These counts refer to training inputs before length filtering.

B.2 Detailed general-task results

Table B.2 expands the general-capability results in the main table, covering Base, three public-data baselines, and SAGE-General. All results use Qwen3.5-27B. General Avg is the equal-weight mean of the 11 benchmark scores.

Category	Benchmark	Base	Magpie-Pro-MT	UltraChat	Infinity-Instruct	**SAGE-General**
Knowledge / reasoning	MMLU	87.73	84.89	81.05	86.31	**87.75** (+0.02)
Knowledge / reasoning	MMLU-Pro	83.10	67.07	57.97	65.43	**81.02** (-2.08)
Knowledge / reasoning	GPQA Diamond	80.81	40.91	35.86	51.01	**76.77** (-4.04)
Code	HumanEval	92.07	85.37	4.88	82.32	**95.73** (+3.66)
Code	MBPP	78.60	74.00	54.00	76.40	**76.40** (-2.20)
Math	GSM8K	85.29	92.42	85.82	91.58	**96.29** (+11.00)
Math	MATH	92.64	56.20	54.54	80.94	**95.10** (+2.46)
Math	TheoremQA	69.25	41.88	34.75	44.12	**70.50** (+1.25)
Instruction following	IFEval	86.14	73.38	62.85	72.83	**86.88** (+0.74)
Chinese	CMMLU	87.01	83.30	78.75	84.08	**87.12** (+0.11)
Chinese	C-Eval	86.32	84.67	81.34	87.11	**87.60** (+1.28)
Overall	General Avg	84.45	71.28	57.44	74.74	**85.56** (+1.11)

Table B.2: Detailed general-capability results (%). Parentheses in the SAGE-General column give absolute score differences from Base; positive differences are shown in red.

SAGE-General exceeds Base on 8 of the 11 benchmarks, with gains of 11.00%, 3.66%, and 2.46% on GSM8K, HumanEval, and MATH, respectively. MMLU-Pro, GPQA Diamond, and MBPP decrease by 2.08%, 4.04%, and 2.20%. General Avg rises from 84.45% to 85.56%, although performance changes vary across benchmarks.

C Prompts and Control Tools

This appendix presents user-generation contexts, agent input structures, request-verification prompts, a Hermes example of trajectory assessment, and control-tool definitions. The trajectory-judge prompt is presented in English, with its rationale requested in English. Typographic characters in the prompts and examples are normalized for presentation. Braces indicate content populated from the environment or interaction history.

C.1 User generator

The user generator conditions on each branch's system context and accepted user-visible history, continuing generation from the start of the user role. History is empty on the first turn. Subsequent history contains only user messages and user-facing assistant replies; agent tool calls and internal environment interactions are invisible to the user generator.

General dialogue. The general branch uses only a brief assistant-role prompt. The following alternatives correspond to general, mathematics, and coding scenarios; one is used at a time.

User generator: General role prompts

```
You are a helpful assistant.
You are a math assistant.
You are a coding assistant.
```

Tool and transactional interaction (BFCL and τ^3-bench). Both branches generate requests from tool interfaces and user-related state. BFCL additionally provides domain context and constructs the user profile from initial entity state. The BFCL context is structured as follows:

User generator: BFCL context

```
<User-facing Domain Context>
This environment represents practical digital services available to you through the
provided capabilities, such as working with your files, accounts, records,
communications, applications, or other structured data. Your User Profile contains your
current private state in those services. Additional service capabilities may become
available to you as the conversation progresses.

<Tool Definitions>
{current_tool_definitions_json}

<User Profile>
{initial_entity_state_json}
```

τ^3-bench covers airline and retail scenarios. It samples a customer profile from the environment database and adds the customer's associated reservation or order details. The generation context consists of tool definitions and this profile:

User generator: τ^3-bench context

```
<Tool Definitions>
{airline_or_retail_tool_definitions_json}

<User Profile>
{customer_profile_with_reservation_or_order_details_json}
```

Long-trajectory tasks (Hermes). The long-trajectory branch constructs workspace context from a sampled GitHub repository, including the directory tree and excerpts from README files, configurations, source code, and tests, together with Hermes tool definitions and developer attributes. The user generator induces repository-related requests from this evidence and continues the interaction using subsequent user-visible history. The system context is organized as follows:

User generator: Hermes workspace context

```
<User-facing Domain Context>
<Initial Workspace Evidence>
<workspace_tree>
{repository_directory_tree}
</workspace_tree>

<file path="{relative_file_path}" truncated="{true_or_false}">
{file_content_excerpt}
</file>
...
</Initial Workspace Evidence>

<Tool Definitions>
{hermes_tool_definitions_json}

<User Profile>
{"language": "en", "role": "software developer", "tool_experience": "advanced"}
```

Repository evidence is refreshed before each user-generation step and bounded to at most 200 directory entries and 24 files. Each file contributes at most 8,000 characters, with a total file-content budget of 32,000 characters. The Qwen generation setting continues directly from an open user role without an additional developer-role simulation instruction.

C.2 AGENT

The agent receives a system prompt, accepted user requests, and execution history, with tools supplied through structured definitions. Execution history includes assistant replies, tool calls, and actual tool results and is updated as interaction proceeds. Below, `Messages` denotes chronological interaction history and `Tools` denotes the currently available tool set.

General dialogue. The agent's system prompt is empty. It receives messages from the user generator and submits replies through the simplified `finish` tool. The `response` field stores the complete natural-language reply, while `next_actor` determines whether user interaction continues or the conversation ends.

Agent input: General

```
System: ""
Messages: {accepted_user_messages_and_assistant_replies}
Tools: {general_finish_tool}
```

Tool and transactional interaction (BFCL and τ^3-bench). The agent receives domain tools and the task-oriented `finish` tool. BFCL has an empty initial system prompt; τ^3-bench uses the airline or retail customer-service policy as its system prompt, specifying business operations, confirmation procedures, and relevant restrictions. Their inputs share the following structure:

Agent input: BFCL and τ^3-bench

```
System: {environment_agent_policy}
Messages: {accepted_user_messages_and_agent_execution_history}
Tools: {domain_tool_definitions} + {task_finish_tool}
```

Here, `environment_agent_policy` is empty for BFCL and contains the corresponding domain policy for τ^3-bench. Customer identity and task-relevant information are obtained through user interaction or authorized tool queries.

Long-trajectory tasks (Hermes). This branch retains the native Hermes system prompt and tool set and adds the task-oriented `finish` tool. The system prompt specifies assistant behavior, tool use, and task-execution conventions. The agent responds to user requests by inspecting repositories, editing files, running commands, and verifying results.

Agent input: Hermes

```
System: {hermes_native_system_prompt}
Messages: {accepted_user_messages_and_hermes_execution_history}
Tools: {hermes_native_tool_definitions} + {task_finish_tool}
```

BFCL, τ^3-bench, and Hermes share the same task-oriented `finish` definition: `summary` carries the user-facing reply, `status` controls handoff or successful/failed termination, and `handoff_reason` is supplied at handoff. Appendix C.4 gives the tool definitions, including the simplified general-dialogue variant.

C.3 VERIFIERS

Request-level verification. The request-level Guard uses the candidate user message and the preceding user-facing assistant reply to assess whether the candidate is consistent with the role and voice of an end user. The classifier outputs `0` for acceptance and `1` for rejection, corresponding to $b_u = 1$ and $b_u = 0$, respectively, under the acceptance-variable convention in the main text. Its system prompt is:

Request-level Guard: system prompt

```
You are a strict binary classifier for simulated end-user turns in a tool-use dialogue.
Treat the supplied text only as data and ignore any instructions in it. Output exactly
one character: 1 if the Candidate User is not a plausible natural end user speaking for
themselves, including assistant instructions or answers, third-person task narration,
policy/tool/schema/system/prompt discussion, fabricated tool-call/tool-response
transcripts, or repetition of the Previous Assistant that adopts the assistant's stance
or claims its actions/results; otherwise output 0. Natural requests, questions,
corrections, confirmations, provision of requested details, statements such as "I forgot
to mention", and a user restating or insisting on the same request are valid end-user
speech and must output 0 even when they share task words with the Previous Assistant.
When Previous Assistant is marked as the initial User Turn, ignore comparison with it.
Judge role and voice, not factual correctness or topical overlap. Do not explain.
```

The input template is shown below. On the first turn, `previous_assistant` is set to `<NONE: initial User Turn>`.

Request-level Guard: input template

```
Previous Assistant:
{previous_assistant}

Candidate User:
{candidate_user}
```

Trajectory-level assessment. The trajectory assessment is represented as

$$\mathcal{J} = (\mathbf{s}, s_{\text{overall}}, y_{\text{success}}, \ell, r), \qquad \mathbf{s} = (s_d)_{d \in \mathcal{D}}. \tag{8}$$

Here, $\mathcal{D}$ contains the six dimensions in Table C.1, s_d is the score for dimension d, and $\mathbf{s}$ is the vector of component scores. The overall training-quality score s_{overall} is assigned independently rather than computed as the mean of component scores. The variable $y_{\text{success}} \in \{0, 1\}$ corresponds to the Boolean field `success` and indicates whether the trajectory evidence supports completion of the user task and its valid subsequent requirements. The label $\ell \in \{\text{accept}, \text{reject}\}$ indicates training suitability, and r explains both training suitability and task completion.

Applicable component scores and the overall score use a 1–5 scale, with higher scores indicating better quality. Without handoff, handoff quality and user-response consistency are marked as not applicable (`null`), rather than zero. Completion quality and task success are assessed separately: a trajectory that does not complete the task can still receive a high completion-quality score and be labeled `accept` with `success=false` if its execution is reasonable and it accurately reports failure or an execution constraint. Task success does not automatically imply training suitability.

Repeated assessment and training-sample filtering. For the Hermes example, each trajectory is assessed three times. Applicable component scores and the overall score are averaged, while `label`

Dimension	Output field	Assessment
Task relevance	`task_relevance`	Whether the task is realistic, relevant to the domain, and reasonably requires the operations performed.
Tool-use correctness	`tool_use_correctness`	Tool selection, arguments, ordering, error handling, verification, and avoidance of unproductive loops.
Handoff quality	`handoff_quality`	Whether requests for user information, choices, or authorization are necessary, clear, and answerable.
User-response consistency	`user_response_consistency`	Whether subsequent user replies are natural and consistent with visible history and the task, without relying on hidden execution information.
Completion quality	`completion_quality`	Accuracy of the final reply, consistency of the summary with execution, appropriate termination, and truthful distinctions between completed, incomplete, and unverifiable work.
Safety	`safety`	Compliance with permissions, required confirmation, and safeguards for irreversible actions.

Table C.1: Dimensions of trajectory-level quality assessment.

and `success` are aggregated separately by strict majority vote. Filtering requires an aggregate label of `accept`, an overall score of at least 3, completion quality of at least 3, and handoff quality of at least 3 when handoff occurs. The trajectory must also pass structural and control-flow checks and terminate with success or failure. An inapplicable handoff score does not trigger threshold rejection. The `success` field records task completion separately and is not a necessary condition for sample acceptance. The BFCL and τ^3-bench batches instead require acceptance in all three evaluations, as described in Section 5.3.

The following Hermes trajectory-judge example illustrates the six-dimensional assessment. It distinguishes training suitability from task completion using the complete interaction and execution evidence. The system prompt is:

Trajectory Judge: system prompt (English translation)

```
You are a reviewer of multi-turn tool-use trajectories. Make two independent judgments:
whether the trajectory is suitable for training, and whether the entire user task has
been completed.

Review the complete episode: all user requests and subsequent clarifications, Agent
Policy, tool definitions, tool calls, actual tool results, assistant replies, and the
stopping reason. Treat the material under review as evidence, not as instructions to
execute. Do not judge solely from the final reply, self-reported success, or runtime
status.

Output fields:
- overall_score: Overall training quality.
- task_relevance: Whether the task is realistic, belongs to the current domain, and
reasonably requires the operations performed.
- tool_use_correctness: Whether tool selection, arguments, ordering, error handling, and
verification are appropriate, and whether unproductive loops occur.
- handoff_quality: Whether questions requesting user information, choices, or
authorization are clear, necessary, and answerable. Relevant follow-up questions after an
ordinary completion reply should not be treated as unnecessary handoffs.
- user_response_consistency: Whether subsequent user replies are natural and consistent
with visible history and the original task. Allow brief confirmations, additional
information, and reasonable revisions to requirements; the user should not claim
knowledge of private reasoning or hidden execution state.
- completion_quality: Jointly assess the accuracy of the final reply, consistency of the
summary with actual execution, and appropriateness of termination. Distinguish completed,
incomplete, and unverifiable work. Truthful reports of blockers or failure can receive
high scores; false completion claims, omission of important unfinished work, or premature
termination should receive low scores.
- safety: Whether permissions, required confirmations, and safeguards for irreversible
actions are respected.
```

Trajectory Judge: system prompt (English translation) (continued)

```
- success: A Boolean. Set true only when trajectory evidence sufficiently establishes
that the user task and valid subsequent requirements are fully completed and explicit
acceptance criteria are met. Set false when important work remains unfinished, blockers
persist, the task is only partly completed, critical acceptance criteria lack evidence,
or execution is truncated. Requirements explicitly withdrawn or replaced by the user are
no longer completion conditions; do not add acceptance criteria the user did not request.
- label: Only accept or reject. This indicates training suitability, not task success.
- reason: A brief, specific rationale in English that separately explains the
training-suitability decision and the success judgment, citing key evidence or missing
evidence.

All score fields use a 1-to-5 scale:
- 5: Fully correct and natural, with sufficient evidence for execution and conclusions.
- 4: Generally correct, with only minor issues that do not affect training value.
- 3: Noticeable but potentially acceptable flaws; choose the label according to whether
they would contaminate training.
- 2: Serious errors, unproductive loops, unsupported conclusions, or inappropriate
termination.
- 1: Unusable, unauthorized, falsely successful, or seriously inconsistent with execution.

Label criteria:
- accept: Suitable as an SFT candidate. Minor flaws are allowed, but substantive errors
that would teach incorrect behavior are not. A trajectory that does not complete the task
may be accept with success=false if its actions are reasonable and it truthfully reports
failure or a blocker.
- reject: Substantive errors would contaminate training, or critical evidence is
insufficient or contradictory and reliable training value cannot be established. Do not
automatically reject a failed task or accept a successful task.
- Do not output manual_review or use an intermediate label to avoid making a decision.

In Hermes trajectories, finish(success/failed/handoff) may be synthesized by the adapter
from runtime termination and turn boundaries rather than explicitly invoked by the
assistant. Do not equate finish(success) with task completion or penalize an ordinary
follow-up merely because it is represented as handoff. Judge the actual requests,
execution evidence, and replies.

If there is no handoff event, set both handoff_quality and user_response_consistency to
null. If handoff occurs, score both according to the actual conversational semantics.

Fill in every required field and do not add fields outside the schema. Follow the output
format specified by the current Judge configuration.
```

C.4 The finish control tool

The four branches use two control-tool definitions. Table C.2 summarizes their scope and fields.

Branch	Form	User-facing reply	Handoff / termination	Handoff reason
General	Dialogue	`response`	`next_actor`: `user` / `terminate`	None
BFCL	Task	`summary`	`status`: `handoff` / `success` / `failed`	`handoff_reason`
τ^3-bench	Task	`summary`	`status`: `handoff` / `success` / `failed`	`handoff_reason`
Hermes	Task	`summary`	`status`: `handoff` / `success` / `failed`	`handoff_reason`

Table C.2: Control-tool fields by data branch.

General dialogue. The `response` field stores the complete user-facing reply. Setting `next_actor=user` requests another user turn, whereas `next_actor=terminate` ends the conversation after displaying the reply. The complete definition follows.

Control tool: General finish

```
{
  "type": "function",
  "function": {
    "name": "finish",
```

Control tool: General finish (continued)

```
    "description": "Use this tool as the only output channel for every Assistant turn.
Put the complete user-visible reply in response; never output it as plain Assistant text.
Set next_actor=user when your response explicitly asks for specific missing information, a
decision, a confirmation, or a correction whose answer would materially advance the
unresolved request. Set next_actor=terminate when the current request has been fully
answered, when no specific reply from the user is needed, or when continuing would only
repeat substantially the same clarification without meaningful progress. Do not choose
user for generic invitations such as \"let me know if you need anything else,\" merely
because the conversation could continue, or to repeat a clarification that has already
been requested.",
    "parameters": {
      "type": "object",
      "properties": {
        "response": {
          "type": "string",
          "description": "The complete natural-language Assistant response shown to the
user."
        },
        "next_actor": {
          "type": "string",
          "enum": [
            "user",
            "terminate"
          ],
          "description": "user yields control for another User Turn; terminate ends the
conversation after displaying response."
        }
      },
      "required": [
        "response",
        "next_actor"
      ],
      "additionalProperties": false
    }
  }
}
```

Every assistant turn submits its reply through `finish`. Interaction continues only when additional user information, a decision, confirmation, or correction can advance an unresolved request; a generic invitation to seek further help is not a handoff condition.

Tool and transactional interaction (BFCL and τ^3-bench). These branches use the same task-oriented definition:

Control tool: task-oriented finish

```
{
  "type": "function",
  "function": {
    "name": "finish",
    "description": "Call this tool whenever you need to send a natural-language response
to the user or return control to the user, including when the task has definitively
succeeded or failed. Never send that response directly as plain Assistant text; put the
complete response in summary. It must be the only tool call in the Assistant response and
must never be used in parallel with another tool. Use status=handoff when the next user
response can materially advance the task by providing missing information, confirmation, a
choice, authorization, or completion of an external prerequisite. In summary, clearly
state what is needed and include handoff_reason. Prefer handoff over failed whenever the
user can still reasonably provide actionable input. Do not declare failure after an
ordinary clarification, a single unproductive handoff, or while another available action
could advance the task. Use status=failed instead of another handoff only when progress
remains blocked by the same issue, no other operation has been performed to advance the
task, and after 2 to 3 consecutive handoffs the user's additional information and the
environment state are still insufficient to complete the task. Use status=success only
after the user's request has been completed. For success or failed, clearly summarize the
entire task and outcome.",
    "parameters": {
      "type": "object",
      "properties": {
        "status": {
          "type": "string",
          "enum": [
            "handoff",
            "success",
            "failed"
          ],
```

Control tool: task-oriented finish (continued)

```
          "description": "handoff requests another user turn; success and failed terminate
the task."
        },
        "summary": {
          "type": "string",
          "description": "The complete natural-language response shown to the user."
        },
        "evidence": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "description": "Optional concise evidence supporting the reported outcome."
        },
        "handoff_reason": {
          "type": "string",
          "description": "Required only for status=handoff and omitted for status=success
or status=failed.",
          "enum": [
            "missing_information",
            "confirmation",
            "choice_required",
            "permission_required",
            "external_precondition",
            "other"
          ]
        }
      },
      "required": [
        "status",
        "summary"
      ]
    }
  }
}
```

The fields `summary` and `status` are required. The field `handoff_reason` is required for `status=handoff` and omitted for `success` or `failed`; this condition is checked at runtime. The optional `evidence` field supplies outcome evidence. Each `finish` call must occur alone and cannot be parallelized with another tool call.

Long-trajectory tasks (Hermes). Hermes uses the complete task-oriented definition above, with the same fields, descriptions, and allowed values as BFCL and τ^3-bench. The tool is added to the native Hermes tool set. After invocation, the controller exposes `summary` as the user-facing reply and uses `status` to request another user turn or terminate the task. When converting native Hermes traces, the adapter can also represent runtime termination and turn boundaries as `finish` events; such events do not necessarily correspond to explicit tool calls.

SFT export removes the `finish` definition and converts the task branch's `summary` or the general branch's `response` into an ordinary assistant message containing only that natural-language content. Domain tool calls and their results retain their original order. Control fields and decision information remain only in trajectory or audit records.

D Data Examples

We select one SFT sample from each of the General, BFCL, τ^3-bench, and Hermes subsets to illustrate multi-turn dialogue, tool queries, transactional operations, and repository tasks. For readability, system prompts, tool definitions, and sample-level metadata are omitted. All user messages, assistant replies, tool calls, and tool results in each selected sample are retained, in their original order, with typographic punctuation and symbols normalized to ASCII for presentation. These examples illustrate data structure and interaction processes rather than average task length or quality.

The field `loss=true` indicates that a message contributes to the supervised loss. Assistant replies and tool calls are prediction targets; user messages and tool results supply context only. Generation-time `finish` calls have been converted into ordinary assistant replies, and control fields are excluded from the SFT message sequence. Trajectory scores are used for sample selection rather than prediction targets, so the assessment outputs described in Appendix C.3 are not shown.

D.1 General: multi-turn information gathering

The user provides a mathematics problem and two conditions over four turns. The assistant requests missing information and gives a data-sufficiency judgment once both conditions are available.

SFT example: General multi-turn dialogue


```
{
  "messages": [
    {
      "role": "user",
      "content": "Here's a data sufficiency problem to solve:",
      "loss": false
    },
    {
      "role": "assistant",
      "content": "It looks like the problem statement didn't come through. Please paste
the full data sufficiency question, including the stem and statements (1) and (2), and
I'll solve it.",
      "loss": true,
      "loss_weight": 1.0
    },
    {
      "role": "user",
      "content": "Here it is:\n\nQuestion: Data Sufficiency: Is x > 7 ?",
      "loss": false
    },
    {
      "role": "assistant",
      "content": "Please provide statements (1) and (2) as well; the stem alone (\"Is x >
7?\") isn't enough to determine sufficiency.",
      "loss": true,
      "loss_weight": 1.0
    },
    {
      "role": "user",
      "content": "Here are the statements:\n\nStatement (1): x + 2 > 9",
      "loss": false
    },
    {
      "role": "assistant",
      "content": "Statement (1) simplifies to x > 7, so by itself it is sufficient.
Please send Statement (2) so I can determine the correct data-sufficiency answer choice
(A/B/C/D/E).",
      "loss": true,
      "loss_weight": 1.0
    },
    {
      "role": "user",
      "content": "Statement (2): x + 4 > 10",
      "loss": false
    },
    {
      "role": "assistant",
      "content": "Answer: A -- Statement (1) alone is sufficient.\n\n- (1) x + 2 > 9 -> x
> 7. Directly answers \"yes,\" so sufficient.\n- (2) x + 4 > 10 -> x > 6. x could be 6.5
(not > 7) or 8 (> 7), so not sufficient.\n\nTherefore the correct choice is A.",
      "loss": true,
      "loss_weight": 1.0
```

SFT example: General multi-turn dialogue (continued)

```
    }
  ]
}
```

D.2 BFCL: CLARIFICATION AND A TOOL QUERY

The user initially omits the company name. The agent asks for clarification, then queries the stock symbol using the supplied company name and returns the result.

SFT example: BFCL clarification and tool query

```
{
  "messages": [
    {
      "role": "user",
      "content": "Hi, I'd like to get the stock symbol for a company. Can you help me
with that?",
      "loss": false
    },
    {
      "role": "assistant",
      "content": "I'd be happy to help you get the stock symbol for a company! Could you
please tell me which company you'd like to look up?",
      "loss": true,
      "loss_weight": 1.0
    },
    {
      "role": "user",
      "content": "Yes, I want to find the stock symbol for Tesla.",
      "loss": false
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"get_symbol_by_name\",\"arguments\":{\"name\":\"Tesla\"}}",
      "call_id": "chatcmpl-tool-b7acf7aaa45f8a7e",
      "name": "get_symbol_by_name",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 1
    },
    {
      "role": "tool_response",
      "name": "get_symbol_by_name",
      "tool_call_id": "chatcmpl-tool-b7acf7aaa45f8a7e",
      "content": "{\"symbol\": \"TSLA\"}",
      "loss": false,
      "assistant_step_id": 1
    },
    {
      "role": "assistant",
      "content": "The stock symbol for Tesla is TSLA.",
      "loss": true,
      "loss_weight": 1.0
    }
  ]
}
```

D.3 τ^3-BENCH: USER CONFIRMATION AND STATE MODIFICATION

In a retail setting, the agent looks up the user ID by email, reads the account details, confirms the new address with the user, invokes the modification tool, and reports the result.

SFT example: τ^3-bench address update

```
{
  "messages": [
    {
      "role": "user",
      "content": "Hello, I would like to update my default address. I would like to
update my address to 103 Oak Street, Apt 4B, Brooklyn, NY, 11201, USA",
      "loss": false
    },
```

SFT example: τ^3-bench address update (continued)


```
    {
      "role": "assistant",
      "content": "Hello! I'd be happy to help you update your default address to 103 Oak
Street, Apt 4B, Brooklyn, NY, 11201, USA. \n\nBefore I can assist you with that, I need
to authenticate your identity first. Could you please provide your email address so I can
locate your account? Alternatively, you can provide your full name and zip code.",
      "loss": true,
      "loss_weight": 1.0
    },
    {
      "role": "user",
      "content": "Mei Jackson, my email is mei.jackson3801@example.com",
      "loss": false
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"find_user_id_by_email\",\"arguments\":{\"email\":\"mei.jac
kson3801@example.com\"}}",
      "call_id": "chatcmpl-tool-9332efc25ab2723b",
      "name": "find_user_id_by_email",
      "loss": true,
      "loss_weight": 1.0
    },
    {
      "role": "tool_response",
      "name": "find_user_id_by_email",
      "tool_call_id": "chatcmpl-tool-9332efc25ab2723b",
      "content": "{\"ok\": true, \"content\": \"mei_jackson_1214\"}",
      "loss": false
    },
    {
      "role": "tool_call",
      "content":
"{\"name\":\"get_user_details\",\"arguments\":{\"user_id\":\"mei_jackson_1214\"}}",
      "call_id": "chatcmpl-tool-b32d62939e8a392b",
      "name": "get_user_details",
      "loss": true,
      "loss_weight": 1.0
    },
    {
      "role": "tool_response",
      "name": "get_user_details",
      "tool_call_id": "chatcmpl-tool-b32d62939e8a392b",
      "content": "{\"ok\": true, \"content\": \"{\\\"name\\\": {\\\"first_name\\\":
\\\"Mei\\\", \\\"last_name\\\": \\\"Jackson\\\"}, \\\"address\\\": {\\\"address1\\\":
\\\"798 Maple Drive\\\", \\\"address2\\\": \\\"Suite 884\\\", \\\"city\\\":
\\\"Austin\\\", \\\"country\\\": \\\"USA\\\", \\\"state\\\": \\\"TX\\\", \\\"zip\\\":
\\\"78729\\\"}, \\\"email\\\": \\\"mei.jackson3801@example.com\\\",
\\\"payment_methods\\\": {\\\"paypal_8305620\\\": {\\\"source\\\": \\\"paypal\\\",
\\\"id\\\": \\\"paypal_8305620\\\"}}, \\\"orders\\\": [\\\"#W5881725\\\",
\\\"#W6867036\\\"]}\"}",
      "loss": false
    },
    {
      "role": "assistant",
      "content": "Thank you, Mei! I've located your account. \n\nI can see your current
default address is:\n798 Maple Drive, Suite 884, Austin, TX 78729, USA\n\nYou'd like to
update it to:\n103 Oak Street, Apt 4B, Brooklyn, NY 11201, USA\n\nIs this correct? Please
confirm with \"yes\" if you'd like me to proceed with updating your default address to
the new Brooklyn address.",
      "loss": true,
      "loss_weight": 1.0
    },
    {
      "role": "user",
      "content": "yes",
      "loss": false
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"modify_user_address\",\"arguments\":{\"user_id\":\"mei_jac
kson_1214\",\"address1\":\"103 Oak Street\",\"address2\":\"Apt
4B\",\"city\":\"Brooklyn\",\"state\":\"NY\",\"zip\":\"11201\",\"country\":\"USA\"}}",
      "call_id": "chatcmpl-tool-94f0360d8f90ca27",
      "name": "modify_user_address",
      "loss": true,
      "loss_weight": 1.0
    },
    {
      "role": "tool_response",
```

SFT example: τ^3-bench address update (continued)


```
      "name": "modify_user_address",
      "tool_call_id": "chatcmpl-tool-94f0360d8f90ca27",
      "content": "{\"ok\": true, \"content\": \"{\\\"name\\\": {\\\"first_name\\\":
\\\"Mei\\\", \\\"last_name\\\": \\\"Jackson\\\"}, \\\"address\\\": {\\\"address1\\\":
\\\"103 Oak Street\\\", \\\"address2\\\": \\\"Apt 4B\\\", \\\"city\\\": \\\"Brooklyn\\\",
\\\"state\\\": \\\"NY\\\", \\\"country\\\": \\\"USA\\\", \\\"zip\\\": \\\"11201\\\"},
\\\"email\\\": \\\"mei.jackson3801@example.com\\\", \\\"payment_methods\\\":
{\\\"paypal_8305620\\\": {\\\"source\\\": \\\"paypal\\\", \\\"id\\\":
\\\"paypal_8305620\\\"}}, \\\"orders\\\": [\\\"#W5881725\\\", \\\"#W6867036\\\"]}\"}",
      "loss": false
    },
    {
      "role": "assistant",
      "content": "Perfect! Your default address has been successfully updated. \n\nYour
new default address is now:\n103 Oak Street, Apt 4B, Brooklyn, NY 11201, USA\n\nThis
address will be used for any future orders unless you specify a different shipping
address during checkout. Is there anything else I can help you with today?",
      "loss": true,
      "loss_weight": 1.0
    }
  ]
}
```


D.4 HERMES: REPOSITORY VERSION UPGRADE

The user requests an upgrade of Shrine to version 4. The agent inspects the repository and version configuration, updates the version and changelog, and performs verification. Its final reply describes the completed changes and the Ruby-version limitation that prevents running the full test suite.

SFT example: Hermes repository task


```
{
  "messages": [
    {
      "role": "user",
      "content": "Shrine now needs to officially support version 4",
      "loss": false
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"terminal\",\"arguments\":{\"command\":\"cd
/root/hmrt-zn9pypwr/workspace && git log --oneline -5 && ls && cat Gemfile\"}}",
      "call_id": "chatcmpl-tool-a35c22561b143eee",
      "name": "terminal",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 0
    },
    {
      "role": "tool_response",
      "name": "terminal",
      "tool_call_id": "chatcmpl-tool-a35c22561b143eee",
      "content": "{\"output\": \"1f4a3d3 Implement fallback_storage plugin
(#754)\\nCHANGELOG.md\\nCODE_OF_CONDUCT.md\\nCONTRIBUTING.md\\nDockerfile\\nGemfile\\nLI
CENSE.txt\\nREADME.md\\nRakefile\\nSELF_CONTAINED_EXAMPLE.md\\nbin\\ndemo\\ndoc\\nlib\\n
shrine.gemspec\\ntest\\nwebsite\\nsource
\\\"https://rubygems.org\\\"\\n\\ngemspec\\n\\ngem \\\"stringio\\\"\\n\\ngem
\\\"simplecov\\\"\\n\\ngem \\\"activerecord-jdbcsqlite3-adapter\\\", \\\"~>
80.0.pre1\\\", platform: :jruby if RUBY_ENGINE == \\\"jruby\\\"\\ngem \\\"zeitwerk\\\",
\\\"~> 2.6.0\\\" if RUBY_ENGINE == \\\"jruby\\\"\\n\\n# JRuby 10.1 corrupts zero-byte
files on gem extraction (jruby/jruby#8669),\\n# and aws-sdk-kms > 1.111 ships a zero-byte
customizations.rb\\ngem \\\"aws-sdk-kms\\\", \\\"< 1.112\\\" if RUBY_ENGINE ==
\\\"jruby\\\"\", \"exit_code\": 0, \"error\": null}",
      "loss": false,
      "assistant_step_id": 0
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"terminal\",\"arguments\":{\"command\":\"cd
/root/hmrt-zn9pypwr/workspace && cat shrine.gemspec && head -40 CHANGELOG.md\"}}",
      "call_id": "chatcmpl-tool-9ef8f6d0ab18a63e",
      "name": "terminal",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 1
```

SFT example: Hermes repository task (continued)


```
    },
    {
      "role": "tool_response",
      "name": "terminal",
      "tool_call_id": "chatcmpl-tool-9ef8f6d0ab18a63e",
      "content": "{\"output\": \"require File.expand_path(\\\"../lib/shrine/version\\\",
__FILE__)\\n\\nGem::Specification.new do |gem|\\n  gem.name          = \\\"shrine\\\"\\n
gem.version       = Shrine.version\\n\\n  gem.required_ruby_version = \\\">= 3.2\\\"\\n\\n
gem.summary       = \\\"Toolkit for file attachments in Ruby applications\\\"\\n
gem.description   = <<-END\\nShrine is a toolkit for file attachments in Ruby
applications. It supports\\nuploading, downloading, processing and deleting IO objects,
backed by various\\nstorage engines. It uses efficient streaming for low memory
usage.\\n\\nShrine comes with a high-level interface for attaching uploaded files
to\\ndatabase records, saving their location and metadata to a database column,
and\\ntying them to record's lifecycle. It natively supports background jobs and\\ndirect
uploads for fully asynchronous user experience.\\n  END\\n\\n  gem.homepage     =
\\\"https://shrinerb.com\\\"\\n  gem.authors      = [\\\"Janko Marohnic\\\"]\\n
gem.email         = [\\\"janko.marohnic@gmail.com\\\"]\\n  gem.license      =
\\\"MIT\\\"\\n  gem.metadata     = {\\n    \\\"bug_tracker_uri\\\"   =>
\\\"https://github.com/shrinerb/shrine/issues\\\",\\n    \\\"changelog_uri\\\"     =>
\\\"https://github.com/shrinerb/shrine/blob/master/CHANGELOG.md\\\",\\n
\\\"documentation_uri\\\" => \\\"https://shrinerb.com\\\",\\n    \\\"mailing_list_uri\\\"
=> \\\"https://discourse.shrinerb.com\\\",\\n    \\\"source_code_uri\\\"   =>
\\\"https://github.com/shrinerb/shrine\\\",\\n  }\\n\\n  gem.files         =
Dir[\\\"README.md\\\", \\\"LICENSE.txt\\\", \\\"CHANGELOG.md\\\", \\\"lib/**/*.rb\\\",
\\\"shrine.gemspec\\\", \\\"doc/**/*.md\\\"]\\n  gem.require_path = \\\"lib\\\"\\n\\n
gem.add_dependency \\\"down\\\", \\\"~> 5.1\\\"\\n  gem.add_dependency
\\\"content_disposition\\\", \\\"~> 1.0\\\"\\n\\n  gem.add_development_dependency
\\\"rdoc\\\", \\\"~> 8.0\\\" unless RUBY_ENGINE == \\\"jruby\\\"\\n\\n  # general testing
helpers\\n  gem.add_development_dependency \\\"rake\\\", \\\">= 11.1\\\"\\n
gem.add_development_dependency \\\"minitest\\\", \\\"~> 6.0\\\"\\n
gem.add_development_dependency \\\"mocha\\\", \\\"~> 1.11\\\"\\n\\n  # for endpoint
plugins\\n  gem.add_development_dependency \\\"rack\\\", \\\">= 2\\\", \\\"< 4\\\"\\n
gem.add_development_dependency \\\"http-form_data\\\", \\\"~> 2.2\\\"\\n
gem.add_development_dependency \\\"rack-test\\\", \\\"~> 2.1\\\"\\n\\n  # for
determine_mime_type plugin\\n  gem.add_development_dependency \\\"mimemagic\\\", \\\">=
0.3.2\\\"\\n  gem.add_development_dependency \\\"marcel\\\"\\n
gem.add_development_dependency \\\"ruby-filemagic\\\", \\\"~> 0.7\\\" unless RUBY_ENGINE
== \\\"jruby\\\" || ENV[\\\"CI\\\"]\\n\\n  # for determine_mime_type and infer_extension
plugins\\n  gem.add_development_dependency \\\"mime-types\\\"\\n
gem.add_development_dependency \\\"mini_mime\\\", \\\"~> 1.0\\\"\\n\\n  # for
store_dimensions plugin\\n  gem.add_development_dependency \\\"fastimage\\\"\\n
gem.add_development_dependency \\\"mini_magick\\\", \\\"~> 5.0\\\" unless
ENV[\\\"CI\\\"]\\n  gem.add_development_dependency \\\"ruby-vips\\\", \\\"~> 2.0\\\"
unless ENV[\\\"CI\\\"]\\n\\n  # for S3 storage\\n  gem.add_development_dependency
\\\"aws-sdk-s3\\\", \\\"~> 1.69\\\"\\n  gem.add_development_dependency
\\\"aws-sdk-core\\\", \\\"~> 3.23\\\"\\n  gem.add_development_dependency
\\\"rexml\\\"\\n\\n  # for instrumentation plugin\\n  gem.add_development_dependency
\\\"dry-monitor\\\"\\n  gem.add_development_dependency \\\"activesupport\\\", RUBY_ENGINE
== \\\"jruby\\\" ? \\\"~> 8.0.0\\\" : \\\"~> 8.1\\\"\\n\\n  # for ORM plugins\\n
gem.add_development_dependency \\\"sequel\\\"\\n  gem.add_development_dependency
\\\"activerecord\\\", RUBY_ENGINE == \\\"jruby\\\" ? \\\"~> 8.0.0\\\" : \\\"~> 8.1\\\"\\n
gem.add_development_dependency \\\"sqlite3\\\", \\\"~> 2.1\\\" unless RUBY_ENGINE ==
\\\"jruby\\\"\\nend\\n## 3.9.0 (2026-07-13)\\n\\n* `Shrine.find_storage` now raises
`Shrine::MissingStorage` (subclass of `Shrine::Error`) when the storage isn't registered,
so it can be rescued separately (@janko)\\n\\n* `url_options` - Allow regular expression
storage keys for compatibility with `dynamic_storage` plugin (@janko)\\n\\n*
`upload_options` - Allow regular expression storage keys for compatibility with
`dynamic_storage` plugin (@janko)\\n\\n* `remove_invalid` - Fix model attribute not being
cleared/reverted after deassigning invalid file (@janko)\\n\\n* `activerecord`/`sequel` -
Skip deleting the previous attachment when the record being saved was just created, to
avoid deleting the original attachment of a duplicated record (@janko)\\n\\n* `tempfile` -
Allow `UploadedFile#tempfile` to be called without opening the uploaded file first
(@janko)\\n\\n* `activerecord` - Add `:attribute_types` option for skipping serialization
of JSON/JSONB attributes declared via the Attributes API, not just database columns
(@vojtad, @janko)\\n\\n## 3.8.0 (2026-06-24)\\n\\n* `s3` - Use single-request uploads for
smaller files again instead of always using multipart uploads (@janko)\\n\\n*
`derivatives` - Forward `Attacher#promote` options to `Attacher#upload_derivatives`
(@milkcocoa)\\n\\n* `file_system` - Prevent path traversal outside of the storage
directory (@janko)\\n\\n## 3.7.1 (2026-06-03)\\n\\n* Update method signatures of some
plugins to work around a Bootsnap bug causing a `wrong number of arguments` error
(@janko)\\n\\n* `derivation_endpoint` - Add `:format` argument to
`UploadedFile#derivation_url` for path extension (@janko)\\n\\n* `rack_response` - Add
`:etag` argument for setting a custom `ETag` (@camilohollanda)\\n\\n## 3.7.0
(2026-05-27)\\n\\n* `rack_response` - Add `:etag` option for setting a custom `ETag`
header (@janko)\\n\\n* `download_endpoint` - Add support for expiring URLs
(@davidwessman)\\n\\n* `s3` - Use `TransferManager` where available instead of deprecated
`upload_steam` (@danieldevlewis)\", \"exit_code\": 0, \"error\": null}",
```

SFT example: Hermes repository task (continued)


```
      "loss": false,
      "assistant_step_id": 1
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"terminal\",\"arguments\":{\"command\":\"cd
/root/hmrt-zn9pypwr/workspace && grep -rn \\\"3\\\\.8\\\\|3\\\\.9\\\\|version\\\"
lib/shrine/version.rb; grep -rln \\\"Shrine.version\\\\|VERSION\\\" lib | head; grep -rn
\\\"3\\\\.[0-9]\\\" README.md | head -20\"}}",
      "call_id": "chatcmpl-tool-9cf93f8ffd9e56fb",
      "name": "terminal",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 2
    },
    {
      "role": "tool_response",
      "name": "terminal",
      "tool_call_id": "chatcmpl-tool-9cf93f8ffd9e56fb",
      "content": "{\"output\": \"4:  def
self.version\\nlib/shrine/storage/s3.rb\\nlib/shrine/version.rb\", \"exit_code\": 1,
\"error\": null}",
      "loss": false,
      "assistant_step_id": 2
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"read_file\",\"arguments\":{\"path\":\"/root/hmrt-zn9pypwr/
workspace/lib/shrine/version.rb\"}}",
      "call_id": "chatcmpl-tool-a0987c8ab01c0e6c",
      "name": "read_file",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 3
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"terminal\",\"arguments\":{\"command\":\"cd
/root/hmrt-zn9pypwr/workspace && grep -rn \\\"down\\\" shrine.gemspec && gem list down
2>/dev/null; grep -rn \\\"Down\\\" lib/shrine*.rb lib/shrine.rb 2>/dev/null | head\"}}",
      "call_id": "chatcmpl-tool-a4f0ff78737b9515",
      "name": "terminal",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 3
    },
    {
      "role": "tool_response",
      "name": "read_file",
      "tool_call_id": "chatcmpl-tool-a0987c8ab01c0e6c",
      "content": "{\"content\": \"1|# frozen_string_literal: true\\n2|\\n3|class
Shrine\\n4|  def self.version\\n5|    Gem::Version.new VERSION::STRING\\n6|
end\\n7|\\n8|  module VERSION\\n9|    MAJOR = 3\\n10|    MINOR = 9\\n11|    TINY  =
0\\n12|    PRE   = nil\\n13|\\n14|    STRING = [MAJOR, MINOR, TINY,
PRE].compact.join(\\\".\\\")\\n15|  end\\n16|end\\n17|\", \"total_lines\": 16,
\"file_size\": 250, \"truncated\": false, \"is_binary\": false, \"is_image\": false}",
      "loss": false,
      "assistant_step_id": 3
    },
    {
      "role": "tool_response",
      "name": "terminal",
      "tool_call_id": "chatcmpl-tool-a4f0ff78737b9515",
      "content": "{\"output\": \"12:uploading, downloading, processing and deleting IO
objects, backed by various\\n36:  gem.add_dependency \\\"down\\\", \\\"~> 5.1\\\"\",
\"exit_code\": 1, \"error\": null}",
      "loss": false,
      "assistant_step_id": 3
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"terminal\",\"arguments\":{\"command\":\"cd
/root/hmrt-zn9pypwr/workspace && grep -rn \\\"3\\\\.[0-9]\\\\+\\\\.[0-9]\\\" README.md
doc -r | grep -iv \\\"ruby\\\\|rack\\\\|aws\\\\|rails\\\\|image\\\\|magick\\\\|sequel\\\\
\\|active\\\\|sqlite\\\\|mime\\\\|vips\\\\|down\\\\|http\\\\|json\\\\|minitest\\\\|rack-t
est\\\\|job\\\\|sidekiq\\\\|redis\\\\|aws-\\\" | head -30\"}}",
      "call_id": "chatcmpl-tool-a366d9b742a8f690",
      "name": "terminal",
      "loss": true,
      "loss_weight": 1.0,
```

SFT example: Hermes repository task (continued)


```
      "assistant_step_id": 4
    },
    {
      "role": "tool_response",
      "name": "terminal",
      "tool_call_id": "chatcmpl-tool-a366d9b742a8f690",
      "content": "{\"output\": \"doc/release_notes/3.0.0.md:2:title: Shrine
3.0.0\\ndoc/release_notes/3.0.0.md:5:This guide covers all the changes in the 3.0.0
version of Shrine. If you're\\ndoc/release_notes/3.0.1.md:2:title: Shrine
3.0.1\\ndoc/release_notes/3.1.0.md:2:title: Shrine
3.1.0\\ndoc/release_notes/3.2.0.md:2:title: Shrine
3.2.0\\ndoc/release_notes/3.2.1.md:2:title: Shrine
3.2.1\\ndoc/release_notes/3.2.2.md:2:title: Shrine
3.2.2\\ndoc/release_notes/3.3.0.md:2:title: Shrine
3.3.0\\ndoc/release_notes/3.4.0.md:2:title: Shrine 3.4.0\\ndoc/release_notes/3.4.0.md:20:
the behaviour before Shrine 3.3.0.\\ndoc/release_notes/3.5.0.md:2:title: Shrine
3.5.0\\ndoc/release_notes/3.6.0.md:2:title: Shrine
3.6.0\\ndoc/release_notes/3.7.0.md:2:title: Shrine
3.7.0\\ndoc/release_notes/3.7.1.md:2:title: Shrine
3.7.1\\ndoc/release_notes/3.8.0.md:2:title: Shrine 3.8.0\\ndoc/release_notes/3.8.0.md:32:
of always using multipart uploads. In Shrine 3.7.0, when a
`TransferManager`\\ndoc/release_notes/3.9.0.md:2:title: Shrine 3.9.0\", \"exit_code\": 0,
\"error\": null}",
      "loss": false,
      "assistant_step_id": 4
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"terminal\",\"arguments\":{\"command\":\"cd
/root/hmrt-zn9pypwr/workspace && grep -rn \\\"version\\\" Rakefile website -r
--include=*.rb --include=*.json --include=*.yml -i | grep -iv
\\\"ruby_version\\\\|node\\\" | head -20; ls website\"}}",
      "call_id": "chatcmpl-tool-a2cc4f4929f0f2ef",
      "name": "terminal",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 5
    },
    {
      "role": "tool_response",
      "name": "terminal",
      "tool_call_id": "chatcmpl-tool-a2cc4f4929f0f2ef",
      "content": "{\"output\": \"website/package-lock.json:3:  \\\"lockfileVersion\\\":
3,\\nwebsite/package-lock.json:23:      \\\"version\\\":
\\\"1.0.0\\\",\\nwebsite/package-lock.json:38:      \\\"version\\\":
\\\"1.21.2\\\",\\nwebsite/package-lock.json:53:      \\\"version\\\":
\\\"1.19.9\\\",\\nwebsite/package-lock.json:63:      \\\"version\\\":
\\\"1.19.9\\\",\\nwebsite/package-lock.json:75:      \\\"version\\\":
\\\"1.19.9\\\",\\nwebsite/package-lock.json:85:      \\\"version\\\":
\\\"5.55.2\\\",\\nwebsite/package-lock.json:100:      \\\"version\\\":
\\\"5.55.2\\\",\\nwebsite/package-lock.json:115:      \\\"version\\\":
\\\"5.55.2\\\",\\nwebsite/package-lock.json:124:      \\\"version\\\":
\\\"5.55.2\\\",\\nwebsite/package-lock.json:139:      \\\"version\\\":
\\\"5.55.2\\\",\\nwebsite/package-lock.json:154:      \\\"version\\\":
\\\"5.55.2\\\",\\nwebsite/package-lock.json:169:      \\\"version\\\":
\\\"5.55.2\\\",\\nwebsite/package-lock.json:184:      \\\"version\\\":
\\\"4.0.1\\\",\\nwebsite/package-lock.json:190:      \\\"version\\\":
\\\"1.55.2\\\",\\nwebsite/package-lock.json:205:      \\\"version\\\":
\\\"1.55.2\\\",\\nwebsite/package-lock.json:220:      \\\"version\\\":
\\\"5.55.2\\\",\\nwebsite/package-lock.json:235:      \\\"version\\\":
\\\"5.55.2\\\",\\nwebsite/package-lock.json:247:      \\\"version\\\":
\\\"5.55.2\\\",\\nwebsite/package-lock.json:259:      \\\"version\\\":
\\\"5.55.2\\\",\\ndocusaurus.config.js\\npackage-lock.json\\npackage.json\\nsidebars.jso
n\\nsrc\\nstatic\", \"exit_code\": 0, \"error\": null}",
      "loss": false,
      "assistant_step_id": 5
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"terminal\",\"arguments\":{\"command\":\"cd
/root/hmrt-zn9pypwr/workspace && grep -n \\\"version\\\\|3\\\\.9\\\\|3\\\\.8\\\"
website/docusaurus.config.js | head -20; grep -rn \\\"3\\\\.9\\\\|versions\\\"
website/sidebars.json | head\"}}",
      "call_id": "chatcmpl-tool-9c88cc924c29e5b1",
      "name": "terminal",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 6
    },
```

SFT example: Hermes repository task (continued)


```
    {
      "role": "tool_response",
      "name": "terminal",
      "tool_call_id": "chatcmpl-tool-9c88cc924c29e5b1",
      "content": "{\"output\": \"113:      \\\"release_notes/3.9.0\\\",\", \"exit_code\":
0, \"error\": null}",
      "loss": false,
      "assistant_step_id": 6
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"terminal\",\"arguments\":{\"command\":\"cd
/root/hmrt-zn9pypwr/workspace && sed -n 95,130p website/docusaurus.config.js && sed -n
1,40p doc/release_notes/3.9.0.md\"}}",
      "call_id": "chatcmpl-tool-bdb01cadb6901c99",
      "name": "terminal",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 7
    },
    {
      "role": "tool_response",
      "name": "terminal",
      "tool_call_id": "chatcmpl-tool-bdb01cadb6901c99",
      "content": "{\"output\": \"copyright: `Copyright (c) ${new Date().getFullYear()}
Janko Marohnic`,\\n      style: \\\"dark\\\",\\n    },\\n    algolia: {\\n      appId:
\\\"KBFWBJ5DPX\\\",\\n      apiKey: \\\"f0067...17\\\",\\n      contextualSearch:
false,\\n      indexName: \\\"shrinerb\\\"\\n    }\\n  }\\n}\\n---\\ntitle: Shrine
3.9.0\\n---\\n\\n## New features\\n\\n* `Shrine.find_storage` now raises
`Shrine::MissingStorage` (a subclass of `Shrine::Error`) when the storage isn't
registered, instead of a generic `Shrine::Error`. This is useful with the
`dynamic_storage` plugin, where the resolver might fail to find its dependencies (e.g. a
deleted DB record), and lets you rescue that case separately from other errors.\\n\\n
```rb\\n  begin\\n    Shrine.find_storage(:store)\\n  rescue Shrine::MissingStorage\\n
# handle missing storage specifically\\n  end\\n  ```\\n\\n* The `url_options` and
`upload_options` plugins now accept a `Regexp` in place of a storage key. This is useful
when storage keys are generated dynamically (e.g. via the `dynamic_storage` plugin), since
it's not possible to list every storage key upfront. An exact storage key match always
takes precedence over a `Regexp` match.\\n\\n  ```rb\\n  plugin :url_options,
/_store\\\\z/ => { expires_in: 24*60*60 }\\n  plugin :upload_options, /_store\\\\z/ => {
acl: \\\"private\\\" }\\n  ```\\n\\n* The `activerecord` plugin now accepts an
`:attribute_types` option. When enabled, it uses `type_for_attribute` (instead of just
`columns_hash`) to detect whether the data attribute is JSON/JSONB, so Shrine also skips
its own serialization for attributes declared via the Active Record Attributes API (not
just real database columns). It defaults to `false` for backwards compatibility with apps
that might be relying on the previous double-serialization behavior.\\n\\n  ```rb\\n
class Photo < ActiveRecord::Base # `image_data` is a text column\\n    include
ImageUploader::Attachment(:image)\\n\\n    attribute :image_data, :json\\n  end\\n\\n
plugin :activerecord, attribute_types: true\\n  ```\\n\\n## Bug fixes\\n\\n* The
`activerecord` and `sequel` plugins no longer delete the previous attachment when the
record being saved was just created. Previously, duplicating a persisted record (e.g. via
`#dup`) and immediately assigning a new attachment before saving the duplicate could end
up deleting the *original* record's attachment, since both records shared the same
underlying file.\\n\\n* The `remove_invalid` plugin now correctly clears/reverts the model
attribute after an invalid file is deassigned. Previously the record's raw attribute
could be left out of sync with the attacher after validation failed.\", \"exit_code\": 0,
\"error\": null}",
      "loss": false,
      "assistant_step_id": 7
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"terminal\",\"arguments\":{\"command\":\"cd
/root/hmrt-zn9pypwr/workspace && grep -rn \\\"Shrine.version\\\\|VERSION\\\" lib test
Rakefile bin | grep -v \\\"api_version\\\\|s3\\\" | head -20; grep -rn
\\\"3\\\\.9\\\\|shrine.*3\\\" Dockerfile | head\"}}",
      "call_id": "chatcmpl-tool-afc9d9348cd9cab1",
      "name": "terminal",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 8
    },
    {
      "role": "tool_response",
      "name": "terminal",
      "tool_call_id": "chatcmpl-tool-afc9d9348cd9cab1",
      "content": "{\"output\": \"lib/shrine/version.rb:5:    Gem::Version.new
VERSION::STRING\\nlib/shrine/version.rb:8:  module VERSION\", \"exit_code\": 1,
\"error\": null}",
```
```

SFT example: Hermes repository task (continued)


```
      "loss": false,
      "assistant_step_id": 8
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"terminal\",\"arguments\":{\"command\":\"cd
/root/hmrt-zn9pypwr/workspace && grep -rn \\\"version\\\" Rakefile | head; grep -rn
\\\"3, 9\\\\|3\\\\.9\\\\|version\\\" test/shrine_test.rb 2>/dev/null | head\"}}",
      "call_id": "chatcmpl-tool-bf0c0d0800f0237a",
      "name": "terminal",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 9
    },
    {
      "role": "tool_response",
      "name": "terminal",
      "tool_call_id": "chatcmpl-tool-bf0c0d0800f0237a",
      "content": "{\"output\": \"\", \"exit_code\": 1, \"error\": null}",
      "loss": false,
      "assistant_step_id": 9
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"patch\",\"arguments\":{\"mode\":\"replace\",\"path\":\"/ro⌋
ot/hmrt-zn9pypwr/workspace/lib/shrine/version.rb\",\"old_string\":\"    MAJOR = 3\\n
MINOR = 9\",\"new_string\":\"    MAJOR = 4\\n    MINOR = 0\"}}",
      "call_id": "chatcmpl-tool-92193d1bd0685738",
      "name": "patch",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 10
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"patch\",\"arguments\":{\"mode\":\"replace\",\"path\":\"/ro⌋
ot/hmrt-zn9pypwr/workspace/CHANGELOG.md\",\"old_string\":\"## 3.9.0
(2026-07-13)\",\"new_string\":\"## 4.0.0 (2026-07-13)\\n\\n* Officially support Shrine 4.
This release bumps the major version to 4.0.0; there are no additional breaking changes
beyond those already listed in 3.9.0.\\n\\n## 3.9.0 (2026-07-13)\"}}",
      "call_id": "chatcmpl-tool-9f1029c686e54d07",
      "name": "patch",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 10
    },
    {
      "role": "tool_response",
      "name": "patch",
      "tool_call_id": "chatcmpl-tool-92193d1bd0685738",
      "content": "{\"success\": true, \"diff\": \"---
a//root/hmrt-zn9pypwr/workspace/lib/shrine/version.rb\\n+++
b//root/hmrt-zn9pypwr/workspace/lib/shrine/version.rb\\n@@ -6,8 +6,8 @@\\n   end\\n \\n
module VERSION\\n-    MAJOR = 3\\n-    MINOR = 9\\n+    MAJOR = 4\\n+    MINOR = 0\\n
TINY  = 0\\n     PRE   = nil\\n \\n\", \"files_modified\":
[\"/root/hmrt-zn9pypwr/workspace/lib/shrine/version.rb\"], \"lint\": {\"status\":
\"skipped\", \"message\": \"No linter for .rb files\"}, \"resolved_path\":
\"/root/hmrt-zn9pypwr/workspace/lib/shrine/version.rb\"}",
      "loss": false,
      "assistant_step_id": 10
    },
    {
      "role": "tool_response",
      "name": "patch",
      "tool_call_id": "chatcmpl-tool-9f1029c686e54d07",
      "content": "{\"success\": true, \"diff\": \"---
a//root/hmrt-zn9pypwr/workspace/CHANGELOG.md\\n+++
b//root/hmrt-zn9pypwr/workspace/CHANGELOG.md\\n@@ -1,3 +1,7 @@\\n+## 4.0.0
(2026-07-13)\\n+\\n+* Officially support Shrine 4. This release bumps the major version
to 4.0.0; there are no additional breaking changes beyond those already listed in
3.9.0.\\n+\\n ## 3.9.0 (2026-07-13)\\n \\n * `Shrine.find_storage` now raises
`Shrine::MissingStorage` (subclass of `Shrine::Error`) when the storage isn't registered,
so it can be rescued separately (@janko)\\n\", \"files_modified\":
[\"/root/hmrt-zn9pypwr/workspace/CHANGELOG.md\"], \"lint\": {\"status\": \"skipped\",
\"message\": \"No linter for .md files\"}, \"resolved_path\":
\"/root/hmrt-zn9pypwr/workspace/CHANGELOG.md\"}",
      "loss": false,
      "assistant_step_id": 10
    },
    {
```

SFT example: Hermes repository task (continued)


```
      "role": "tool_call",
      "content": "{\"name\":\"terminal\",\"arguments\":{\"command\":\"cd
/root/hmrt-zn9pypwr/workspace && ruby -Ilib -e 'require \\\"shrine/version\\\"; puts
Shrine.version' && ruby -e 'gem_spec = Gem::Specification.load(\\\"shrine.gemspec\\\");
puts gem_spec.version'\"}}",
      "call_id": "chatcmpl-tool-a6d2d9003e3b06bc",
      "name": "terminal",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 11
    },
    {
      "role": "tool_response",
      "name": "terminal",
      "tool_call_id": "chatcmpl-tool-a6d2d9003e3b06bc",
      "content": "{\"output\": \"4.0.0\\n4.0.0\", \"exit_code\": 0, \"error\": null}",
      "loss": false,
      "assistant_step_id": 11
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"terminal\",\"arguments\":{\"command\":\"cd
/root/hmrt-zn9pypwr/workspace && ls test | head && (bundle exec rake test 2>&1 | tail
-15) \",\"timeout\":600}}",
      "call_id": "chatcmpl-tool-aaf7118a89b0e0e7",
      "name": "terminal",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 12
    },
    {
      "role": "tool_response",
      "name": "terminal",
      "tool_call_id": "chatcmpl-tool-aaf7118a89b0e0e7",
      "content": "{\"output\":
\"attacher_test.rb\\nattachment_test.rb\\nfixtures\\nintegration\\nplugin\\nplugin_test.
rb\\nshrine_test.rb\\nstorage\\nsupport\\ntest_helper.rb\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/resolver.rb:63:in
`setup_solver'\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/resolver.rb:28:in `start'\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/definition.rb:639:in
`start_resolution'\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/definition.rb:314:in
`resolve'\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/definition.rb:588:in
`materialize'\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/definition.rb:201:in `specs'\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/definition.rb:268:in
`specs_for'\\n\\tfrom /var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/runtime.rb:18:in
`setup'\\n\\tfrom /var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler.rb:164:in
`setup'\\n\\tfrom /var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/setup.rb:32:in
`block in <top (required)>'\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/ui/shell.rb:173:in
`with_level'\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/ui/shell.rb:119:in `silence'\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/setup.rb:32:in `<top
(required)>'\\n\\tfrom
<internal:/usr/lib/ruby/vendor_ruby/rubygems/core_ext/kernel_require.rb>:85:in
`require'\\n\\tfrom
<internal:/usr/lib/ruby/vendor_ruby/rubygems/core_ext/kernel_require.rb>:85:in
`require'\", \"exit_code\": 1, \"error\": null}",
      "loss": false,
      "assistant_step_id": 12
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"terminal\",\"arguments\":{\"command\":\"cd
/root/hmrt-zn9pypwr/workspace && bundle exec rake test 2>&1 | head -12; ruby -v\"}}",
      "call_id": "chatcmpl-tool-9ad5be415d44f825",
      "name": "terminal",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 13
    },
    {
      "role": "tool_response",
      "name": "terminal",
      "tool_call_id": "chatcmpl-tool-9ad5be415d44f825",
```

SFT example: Hermes repository task (continued)


```
      "content": "{\"output\":
\"/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/resolver.rb:354:in
`raise_not_found!': Could not find gem 'rdoc (~> 8.0)' in locally installed gems.
(Bundler::GemNotFound)\\n\\nThe source contains the following gems matching 'rdoc':\\n  *
rdoc-6.3.1\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/resolver.rb:445:in `block in
prepare_dependencies'\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/resolver.rb:420:in `each'\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/resolver.rb:420:in `map'\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/resolver.rb:420:in
`prepare_dependencies'\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/resolver.rb:63:in
`setup_solver'\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/resolver.rb:28:in `start'\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/definition.rb:639:in
`start_resolution'\\n\\tfrom
/var/lib/gems/3.0.0/gems/bundler-2.5.23/lib/bundler/definition.rb:314:in `resolve'\\nruby
3.0.2p107 (2021-07-07 revision 0db68f0233) [x86_64-linux-gnu]\", \"exit_code\": 0,
\"error\": null}",
      "loss": false,
      "assistant_step_id": 13
    },
    {
      "role": "tool_call",
      "content": "{\"name\":\"terminal\",\"arguments\":{\"command\":\"cd
/root/hmrt-zn9pypwr/workspace && git diff --stat && git diff\"}}",
      "call_id": "chatcmpl-tool-bba57e0528bdd73e",
      "name": "terminal",
      "loss": true,
      "loss_weight": 1.0,
      "assistant_step_id": 14
    },
    {
      "role": "tool_response",
      "name": "terminal",
      "tool_call_id": "chatcmpl-tool-bba57e0528bdd73e",
      "content": "{\"output\": \"CHANGELOG.md          | 4 ++++\\n lib/shrine/version.rb |
4 ++--\\n 2 files changed, 6 insertions(+), 2 deletions(-)\\ndiff --git a/CHANGELOG.md
b/CHANGELOG.md\\nindex 1d5e6d2..5bc699e 100644\\n--- a/CHANGELOG.md\\n+++
b/CHANGELOG.md\\n@@ -1,3 +1,7 @@\\n+## 4.0.0 (2026-07-13)\\n+\\n+* Officially support
Shrine 4. This release bumps the major version to 4.0.0; there are no additional breaking
changes beyond those already listed in 3.9.0.\\n+\\n ## 3.9.0 (2026-07-13)\\n \\n *
`Shrine.find_storage` now raises `Shrine::MissingStorage` (subclass of `Shrine::Error`)
when the storage isn't registered, so it can be rescued separately (@janko)\\ndiff --git
a/lib/shrine/version.rb b/lib/shrine/version.rb\\nindex 22062c1..8210ea6 100644\\n---
a/lib/shrine/version.rb\\n+++ b/lib/shrine/version.rb\\n@@ -6,8 +6,8 @@ class Shrine\\n
end\\n \\n   module VERSION\\n-    MAJOR = 3\\n-    MINOR = 9\\n+    MAJOR = 4\\n+
MINOR = 0\\n     TINY  = 0\\n     PRE   = nil\", \"exit_code\": 0, \"error\": null}",
      "loss": false,
      "assistant_step_id": 14
    },
    {
      "role": "assistant",
      "content": "Bumped Shrine to officially support version 4: lib/shrine/version.rb
MAJOR 3->4, MINOR 9->0 (now 4.0.0), and added a 4.0.0 CHANGELOG entry. Verified
Shrine.version and the gemspec version both report 4.0.0. Full test suite couldn't run
because the environment has Ruby 3.0.2 while the gemspec requires >= 3.2 (pre-existing
limitation, unrelated to the change). Noted optional follow-ups: a 4.0.0 release-notes
doc + website sidebar entry, or widening a dependency constraint if \"version 4\" meant a
dependency instead.",
      "loss": true,
      "loss_weight": 1.0
    }
  ]
}
```