AgentBrew: Offline Tool-Use Agent Learning from Raw Real-World Trajectories

Zhiyi Lyu[1] **Yewen Li**[2] **Longtao Zheng**[1] **Shengtian Yang**[3] **Lang Feng**[1] **Lei Feng**[3]

Peng Jiang[2] **Kun Gai**[2] **Qingpeng Cai**[2] **Bo An**[1]

[1] Nanyang Technological University [2] Kuaishou Technology [3] Southeast University

Abstract

LLM-based agents are increasingly deployed in real-world applications through tool-use APIs, yet training them for specific environments remains fundamentally difficult: real-world applications provide no pre-defined tasks or verifiers, no faithful simulators, and limited budget for large-scale environment interaction. In this paper, we propose **AgentBrew**, an offline training framework that learns effective tool-use policies from a single batch of raw interaction trajectories, without task verifiers or iterative on-policy rollouts. The agent first explores the target environment to collect a raw trajectory corpus without quality filtering. To extract training signal from this noisy corpus, *retrospective task inference* reconstructs an aligned instruction for each trajectory based on its actual outcome, and *PMI-Based credit assignment* decomposes the trajectory's total information about the inferred instruction into additive per-action credits via pointwise mutual information (PMI). These credits weight the policy training objective, amplifying informative actions while suppressing ineffective ones. On three real-world MCP applications (GitHub, Notion, PostgreSQL), AgentBrew improves Qwen3-32B by +8.7 Acc / +9.7 Score on average, surpassing Qwen3-235B (+2.3 / +4.4) and outperforming rejection sampling (+5.9 / +10.3). These results demonstrate that fine-grained offline learning can recover useful supervision from raw trajectories that filtering-based approaches would discard. The code is available at https://github.com/alphatogo/AgentBrew

1 Introduction

Tool-use agents powered by Large Language Models (LLMs) are rapidly moving from research prototypes to production deployments [44; 21; 29; 2]. Standardized protocols like the Model Context Protocol (MCP) [2] now connect LLM agents to real-world applications such as Notion [22] and GitHub [10], enabling tool-use across complex, stateful environments. However, each application presents a distinct tool landscape: Notion exposes dozens of APIs spanning database operations, page management, and content editing, each requiring precise parameter composition and awareness of the application's live state. General-purpose LLMs, even at frontier scale, struggle with such environment-specific complexity, exhibiting low task completion rates on real-world tool-use benchmarks [20; 39], indicating that reliable deployment demands targeted, environment-specific training.

Existing approaches to training tool-use agents for specific applications follow two main routes. The first operates on real-world data [42; 48; 31]: practitioners design a suite of tasks with ground-truth verifiers for the target application, deploy an agent to interact with the live environment and collect trajectories, then use the verifiers to filter for successful completions and train via supervised fine-tuning on the retained high-quality subset [26; 48; 24]. The second pursues on-policy optimization

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

[30; 9; 13]: because repeated rollouts against live APIs incur monetary costs, rate limits, and irreversible side effects on production data, these methods instead construct simulated environments that replicate the application's state and tool behavior, enabling large-scale iterative training before deploying the resulting policy back to the real application [6; 38; 35; 28]. Despite their different trade-offs, both routes rest on shared prerequisites: pre-defined task suites, reliable reward signals, and either sufficient real-world interaction budget or a faithful simulator of the target environment.

While these methods have achieved strong results on curated benchmarks, applying them to real-world applications exposes fundamental difficulties at three levels. At the **task level**, real-world applications provide neither pre-defined task nor ground-truth verifiers. Tasks must be constructed from scratch with validity dependent on the environment's live state, and reliable verification remains an open problem [12]: neither manual annotation [7; 28] nor automated judging scales to complex, stateful API interactions (see §4.2). At the **trajectory level**, the offline route faces a severe data efficiency problem. Real-world tool-use tasks involve long, complex API compositions with low completion rates, so standard trajectory-level filtering [42; 48] discards the majority of an already limited corpus. Worse, even a 'failed' trajectory may contain actions that are useful training signal for a different but related task, yet trajectory-level filtering discards them entirely (Figure 1, left). At the **training level**, the online route introduces a significant sim-to-real gap: simulators cannot faithfully reproduce the complex, interconnected state of real-world applications. Recent work confirms this empirically: simulator-trained agents achieve strong results on synthetic benchmarks yet degrade when evaluated on real-world platforms [6]. Together, these challenges rule out standard training pipelines and point to a setting that, despite being the norm for industrial deployment, remains largely unexplored in the research community:

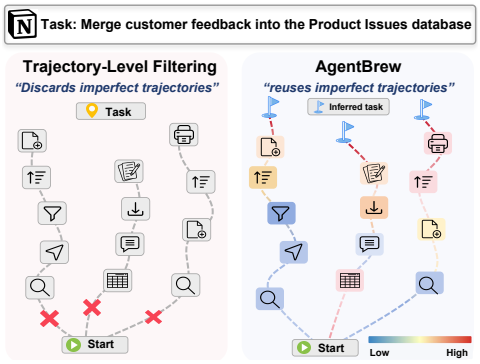


Figure 1: Left: trajectory-level filtering discards entire trajectories that fail the original task, losing potentially useful actions. Right: AgentBrew infers a better-aligned task for each trajectory and assigns per-action credits under the inferred task, enabling the entire corpus to contribute to training.

> ***How can we train a tool-use agent for a specific application, with no pre-defined tasks, no verifiers, no simulator, and limited environment interaction?***

In this paper, we present **AgentBrew**, an offline training framework that learns effective tool-use policies from a single batch of raw interaction trajectories, without task verifiers or iterative rollouts. Our key insight is that every trajectory, regardless of whether it completes its original task, carries reusable signal: by inferring what each trajectory actually accomplished and scoring which actions genuinely contributed, we extract fine-grained supervision from the entire raw corpus without discarding any data. **First**, the agent explores the target environment to discover its live state and proposes grounded task instructions, then executes these tasks to collect a raw trajectory corpus without any quality filtering (§3.1). **Second**, we recover fine-grained supervision from this noisy corpus through two complementary mechanisms (§3.2). For each trajectory, *retrospective task inference* reconstructs an instruction that reflects what the trajectory actually achieved rather than what it originally intended, producing aligned instruction-trajectory pairs. *PMI-Based credit assignment* then quantifies how much each action contributes to the inferred task. We define the total information a trajectory carries about its inferred instruction via pointwise mutual information (PMI), and leverage the chain rule of PMI to decompose this trajectory-level quantity into additive per-action credits, each measuring the marginal information a single action contributes. **Third**, these credits serve as sample weights for policy training, amplifying the gradient from informative actions while suppressing that from ineffective ones (§3.3). Crucially, once the raw corpus is gathered, all subsequent stages proceed entirely offline with no further environment interaction, making the framework practical for real-world APIs where iterative exploration is infeasible.

We evaluate AgentBrew on three real-world MCP applications: GitHub, Notion, and PostgreSQL [20; 39]. Without any external reward signals, human annotations, or iterative environment interaction, AgentBrew improves over the Qwen3-32B base model by +8.7 Acc and +9.7 Score on average,

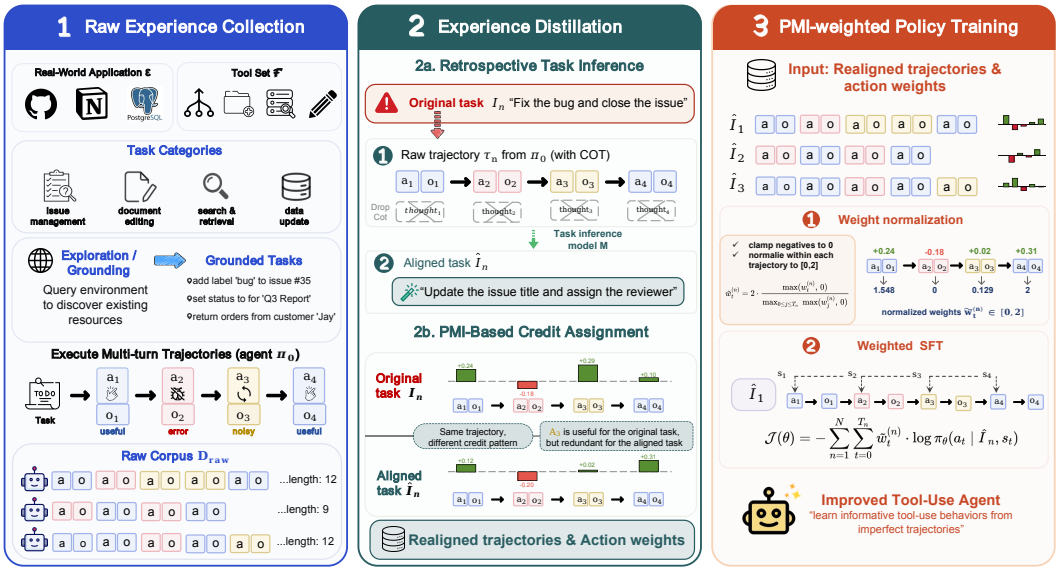


Figure 2: Overview of AgentBrew. (1) The agent explores $\mathcal{E}$ to propose grounded tasks and collects raw trajectories. (2) Retrospective task inference recovers aligned instructions $\hat{I}_n$, and PMI-based credit assignment scores each action. (3) Normalized credits weight the policy training objective.

surpassing Qwen3-235B and outperforming rejection sampling by +5.9 Acc and +10.3 Score. These results demonstrate that fine-grained offline learning can extract useful training signal from raw trajectories that filtering-based approaches would discard entirely.

2 Problem Formulation

We consider an LLM agent operating within a real-world application environment $\mathcal{E}$ (e.g., GitHub, Notion) that exposes a set of callable tools $\mathcal{F} = \{f_1, \ldots, f_K\}$, each defined by a natural language description and a typed parameter specification. Given a task instruction $I \in \mathcal{I}$, the agent interacts with $\mathcal{E}$ over discrete steps $t = 0, \ldots, T$. At each step, the agent observes the state $s_t = (I, a_0, o_0, \ldots, a_{t-1}, o_{t-1})$ and generates an action $a_t = (u_t, c_t)$, where u_t is a chain-of-thought trace and c_t is a tool call. The environment returns an observation $o_t = \mathcal{E}(c_t)$, yielding a complete trajectory $\tau = \{(s_t, a_t, o_t)\}_{t=0}^{T}$.

Our goal is to improve the agent's policy π_θ for tool-use tasks in $\mathcal{E}$ under three constraints that reflect the practical reality of most real-world applications (§1): (1) no pre-defined tasks or verifiers are available; (2) the agent may collect a single trajectory corpus $\mathcal{D}_{\text{raw}} = \{(I_n, \tau_n)\}_{n=1}^{N}$ but further large-scale interaction is infeasible; and (3) trajectories carry no external reward signals. Each trajectory terminates when the agent issues a completion action or a maximum step limit is reached, but neither condition constitutes ground-truth verification of success. The objective is to extract a training signal from this unlabeled $\mathcal{D}_{\text{raw}}$ that improves π_θ on the distribution of user tasks in $\mathcal{E}$.

3 Methodology

We propose **AgentBrew**, a three-stage offline framework illustrated in Figure 2. Given a target environment $\mathcal{E}$, the agent first collects a raw trajectory corpus $\mathcal{D}_{\text{raw}}$ through grounded exploration (§3.1). The raw corpus then undergoes experience distillation (§3.2), which produces realigned instruction-trajectory pairs with per-action credit scores. Finally, these credits serve as sample weights for a maximum likelihood training objective (§3.3). Once the initial corpus is gathered, the entire pipeline requires no further environment interaction.

3.1 Raw Experience Collection

Grounded Task Proposal. Prior approaches synthesize tasks directly from tool documentation [26; 42], but frequently produce tasks that reference nonexistent entities or invalid configurations

because they lack access to the live state of $\mathcal{E}$. We instead ground task construction in the environment itself. The agent first generates a set of high-level task categories from the tool descriptions of $\mathcal{F}$ (see Appendix C), following the common practice of aligning synthetic training tasks with target capability profiles [42; 48; 34]. Each category specifies the capability profile to exercise, not the specific entities or data involved. Training and evaluation use fully isolated environment instances with no overlap in entities, data entries, or environment states. For each category, the agent conducts exploratory interactions with $\mathcal{E}$, invoking tools to retrieve concrete environmental information such as available resources and existing data entries, and then proposes task instructions grounded in the retrieved information. Generated tasks are validated to ensure all referenced schema entities exist in the target environment. This yields a task set $\mathcal{I} = \{I_n\}_{n=1}^{N}$ whose instructions reference entities and configurations that verifiably exist in $\mathcal{E}$.

Task Execution. For each proposed task I_n, the agent interacts with $\mathcal{E}$ over multiple turns following the process described in §2, producing a trajectory $\tau_n = \{(s_t, a_t, o_t)\}_{t=0}^{T_n}$. The interaction terminates when the agent returns a final answer indicating task completion, or when a maximum turn limit $T_{\max}$ is reached. Trajectories that reach $T_{\max}$ are discarded as they typically reflect degenerate behaviors such as repeated failed API calls with no meaningful environmental interaction. All remaining naturally terminated trajectories are retained without quality filtering, yielding the raw corpus $\mathcal{D}_{\text{raw}} = \{(I_n, \tau_n)\}_{n=1}^{N}$.

3.2 Experience Distillation

The original instructions $\{I_n\}$ are necessary to drive trajectory collection, but most trajectories fail to complete I_n [20; 39; 6], making the raw pairs (I_n, τ_n) unsuitable for direct training: the policy would learn to associate incorrect behaviors with the stated goal. We address this through two complementary steps. *Retrospective task inference* (§3.2.1) replaces each I_n with a revised instruction $\hat{I}_n$ that reflects what the trajectory actually accomplished, producing aligned pairs for training. *PMI-based credit assignment* (§3.2.2) then scores each action by how much it contributes to $\hat{I}_n$, so that informative actions receive stronger supervision while irrelevant or erroneous ones are suppressed.

3.2.1 Retrospective Task Inference

To construct aligned pairs, we first discard all chain-of-thought traces u_t from each trajectory, retaining only the tool calls and environment responses:

$$\bar{\tau}_n = \{(c_t, o_t)\}_{t=0}^{T_n} \tag{1}$$

This removal is necessary because u_t is generated conditioned on the original instruction I_n and may contradict the trajectory's actual outcome: for instance, u_t may claim a sub-goal is being achieved while o_t indicates an API error or no state change. Retaining such traces would bias the inference toward intentions the trajectory did not fulfill. The resulting $\bar{\tau}_n$ is analogous to a user interaction log of application operations and system responses, stripped of subjective rationales.

Given $\bar{\tau}_n$, we prompt an LLM to infer a revised instruction $\hat{I}_n$ that is consistent with the trajectory's observable outcomes (prompt details in Appendix E). The inference is guided by several key constraints: $\hat{I}_n$ must describe the outcome genuinely achieved in $\mathcal{E}$ rather than narrate the action sequence; claims not directly supported by successful operations in $\bar{\tau}_n$ are removed; and no entities or results beyond what is evidenced by the trajectory may be introduced. This produces realigned pairs $(\hat{I}_n, \bar{\tau}_n)$ for downstream supervision.

3.2.2 PMI-Based Credit Assignment

After retrospective task inference, we obtain aligned pairs $(\hat{I}_n, \bar{\tau}_n)$ that could in principle be used directly for supervised fine-tuning. However, not all actions in τ_n contribute equally to $\hat{I}_n$: some served the original intent of I_n but are irrelevant to $\hat{I}_n$, and others are low-quality such as incorrect API calls and redundant queries. We therefore need a per-action credit that quantifies each action's contribution to $\hat{I}_n$. Our key idea is to use a reference language model π_{ref} to measure how much information each action-observation pair provides about the inferred task $\hat{I}_n$. This informational

measure correlates well with actual contribution because $\hat{I}_n$ is itself derived from the trajectory's observable outcomes (§3.2.1): actions that produced the described effects leave distinctive traces in o_t that strongly reduce uncertainty about the outcome-derived $\hat{I}_n$, receiving high credit, while failed or redundant actions provide little such evidence and are suppressed. We formalize this using pointwise mutual information (PMI).

Total information of a trajectory. To assign per-action credits, we first define the total information that a trajectory carries about its inferred task, then decompose it across individual actions. Let π_{ref} denote the reference language model. The PMI between the reduced trajectory $\bar{\tau}_n = \{(c_t, o_t)\}_{t=0}^{T_n}$ and the inferred instruction $\hat{I}_n$ is:

$$\text{PMI}(\hat{I}_n;\, \bar{\tau}_n) = \log \frac{\pi_{\text{ref}}(\hat{I}_n \mid \bar{\tau}_n)}{\pi_{\text{ref}}(\hat{I}_n)} = \underbrace{(-\log \pi_{\text{ref}}(\hat{I}_n))}_{\mathcal{L}_\emptyset} - \underbrace{(-\log \pi_{\text{ref}}(\hat{I}_n \mid \bar{\tau}_n))}_{\mathcal{L}_{T_n}} = \mathcal{L}_\emptyset - \mathcal{L}_{T_n} \quad (2)$$

where $\mathcal{L}_\emptyset = -\log \pi_{\text{ref}}(\hat{I}_n)$ is the negative log-likelihood (NLL) of $\hat{I}_n$ under the reference model without any context, and $\mathcal{L}_{T_n} = -\log \pi_{\text{ref}}(\hat{I}_n \mid \bar{\tau}_n)$ is the NLL after conditioning on the entire trajectory. We denote this total information as $\Delta\mathcal{L} = \mathcal{L}_\emptyset - \mathcal{L}_{T_n}$. The question is how to decompose $\Delta\mathcal{L}$ across the $T_n + 1$ individual actions.

Per-action credit via the chain rule of PMI. The chain rule of PMI decomposes the total information into a sum of conditional PMI terms, each measuring the marginal information contribution of a single action. Let $\mathcal{L}_t = -\log \pi_{\text{ref}}(\hat{I}_n \mid c_0, o_0, \ldots, c_t, o_t)$ denote the NLL after observing actions up to step t. The conditional PMI between (c_t, o_t) and $\hat{I}_n$ given preceding actions $(c_{<t}, o_{<t})$ is:

$$w_t = \text{PMI}(\hat{I}_n;\, (c_t, o_t) \mid c_{<t}, o_{<t}) = \log \frac{\pi_{\text{ref}}(\hat{I}_n \mid c_{\le t}, o_{\le t})}{\pi_{\text{ref}}(\hat{I}_n \mid c_{<t}, o_{<t})} = \mathcal{L}_{t-1} - \mathcal{L}_t, \quad t = 1, \ldots, T_n \quad (3)$$

with $w_0 = \mathcal{L}_\emptyset - \mathcal{L}_0$ for the first action. Each w_t quantifies how much additional information (c_t, o_t) provides about $\hat{I}_n$ beyond what is already known from the preceding actions. An action that is highly informative of $\hat{I}_n$ yields a large positive w_t, a redundant action yields $w_t \approx 0$, and an action whose observation misleads the model away from $\hat{I}_n$ yields $w_t < 0$.

By the chain rule of PMI, these per-action credits form an exact additive decomposition of the total information:

$$\sum_{t=0}^{T_n} w_t = (\mathcal{L}_\emptyset - \mathcal{L}_0) + \sum_{t=1}^{T_n} (\mathcal{L}_{t-1} - \mathcal{L}_t) = \mathcal{L}_\emptyset - \mathcal{L}_{T_n} = \Delta\mathcal{L} \quad (4)$$

This guarantees that the per-action credits neither inflate nor discard any information: each unit of PMI between $\bar{\tau}_n$ and $\hat{I}_n$ is attributed to exactly one action.

3.3 PMI-weighted Policy Training

The raw credits may be negative and vary in scale across trajectories. We clamp negative values to zero because any instrumental value of an uncertainty-increasing action is already captured by the subsequent step that resolves that uncertainty, as guaranteed by the additive decomposition in Eq. (4). We then normalize within each trajectory to $[0, 2]$:

$$\tilde{w}_t^{(n)} = 2 \cdot \frac{\max(w_t^{(n)},\, 0)}{\max_{0 \le j \le T_n} \max(w_j^{(n)},\, 0)} \quad (5)$$

The policy is then optimized by minimizing the weighted negative log-likelihood, following the sample-weighted SFT paradigm used in prior work [47; 11]:

$$\mathcal{J}(\theta) = -\sum_{n=1}^{N} \sum_{t=0}^{T_n} \tilde{w}_t^{(n)} \cdot \log \pi_\theta(a_t \mid \hat{I}_n, s_t) \quad (6)$$

where $s_t = (\hat{I}_n,\, a_0, o_0, \ldots, a_{t-1}, o_{t-1})$. Although credits are computed over (c_t, o_t), the loss is applied to the full action $a_t = (u_t, c_t)$ including chain-of-thought tokens. The credit weighting

Table 1: Main results across three real-world applications. We report **Acc** (binary task completion rate) and **Score** (average per-task score, 0–1). Best results per column are **bolded**. Green and red deltas indicate changes relative to Qwen3-32B.

Method	GitHub		Notion		Postgres		Average	
	Acc.	Score	Acc.	Score	Acc.	Score	Acc.	Score
Frontier Proprietary Models								
GPT-5	64.2	85.4	42.9	83.4	42.9	61.7	50.0	76.8
Claude Sonnet-4.5	57.1	76.9	39.3	84.6	38.1	53.3	44.8	71.6
Gemini-3 Pro	60.7	81.2	53.6	86.2	57.1	75.1	57.1	80.8
Open-Source Foundation Models								
GLM-5-744B	42.9	74.3	32.1	77.3	23.8	59.4	32.9	70.3
DeepSeek-V3.2-685B	17.8	63.1	39.2	74.3	33.3	64.1	30.1	67.2
Qwen3-235B	17.9	63.0	10.7	65.7	19.0	34.3	15.9	54.3
GPT-OSS-120B	3.6	37.8	3.6	54.6	14.3	24.1	7.1	38.8
Qwen3-32B	10.7	49.7	3.6	60.1	14.3	37.2	9.5	49.0
Training Methods (32B)								
TOUCAN	17.8	56.4	3.6	49.2	14.3	28.8	11.9 (+2.4)	44.8 (-4.2)
Vanilla SFT	14.2	53.9	7.1	59.8	9.5	30.7	10.3 (+0.8)	48.1 (-0.9)
Rejection Sampling	17.8	53.9	0	53.2	19.0	38.0	12.3 (+2.8)	48.4 (-0.6)
AgentBrew	**21.4**	**63.5**	**14.2**	**70.1**	**23.8**	**41.8**	**18.2** (+8.7)	**58.7** (+9.7)

mitigates potential misalignment between u_t and $\hat{I}_n$: low-contribution actions receive near-zero weight, suppressing gradient from both their tool calls and associated reasoning, while high-credit actions tend to have reasoning consistent with their executed behavior. Overall, informative actions receive amplified gradient signal while uninformative ones are effectively masked. We present pseudo code and additional details in Appendix A & D.

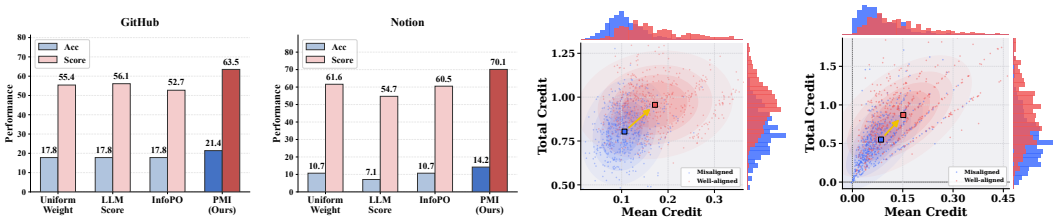
4 Experiments

We evaluate AgentBrew across three real-world applications, examining overall performance (§4.2), the effectiveness of PMI-based credit assignment and retrospective task inference (§4.3–4.4), robustness to noisy data (§4.5), and transferability across model sizes (§4.6).

4.1 Experimental Settings

Setup. We evaluate on three real-world MCP applications [20; 39]: **GitHub** (code collaboration), **Notion** (knowledge management), and **PostgreSQL** (relational database). We compare against frontier proprietary models (GPT-5, Claude Sonnet-4.5, Gemini-3 Pro), open-source foundation models (GLM-5-744B, DeepSeek-V3.2-685B, Qwen3-235B, GPT-OSS-120B, Qwen3-32B), and training-based methods built on Qwen3-32B: TOUCAN [42], Vanilla SFT on our collected trajectories, and Rejection Sampling [47] which retains only LLM-judge-approved trajectories for SFT. The evaluation tasks and ground-truth verifiers are provided by the mcp-universe and mcpmark benchmarks [20; 39]; that these verifiers are used solely for evaluation; they are not available during any stage of the AgentBrew pipeline. To prevent data leakage, training and evaluation are isolated at two levels: training tasks are generated from capability descriptions, and the training corpus is collected from environment instances that share no entities, data entries, or states with the benchmark evaluation environments (details in Appendix F).

Data collection and implementation. We use Qwen3-32B [43] as the unified model for the entire AgentBrew pipeline, serving as the base policy, the reference model π_{ref} for PMI credit computation, and the backbone for retrospective task inference. The raw corpus $\mathcal{D}_{\text{raw}}$ contains approximately 2,500 trajectories for GitHub, 2,500 for Notion, and 3,500 for PostgreSQL. Since real-world interactions often exceed 100K tokens, we adopt a memory mechanism that retains the three most recent steps



(a) (b) (c) (d)

Figure 3: **Effect of credit assignment.** (a-b) Performance comparison of different credit assignment strategies on GitHub and Notion. (c-d) Distribution of mean credit and total credit for well-aligned (red) and misaligned (blue) trajectories on GitHub (c) and Notion (d).

in full and compresses earlier history into LLM-generated summaries [23; 49]. This mechanism is applied uniformly across all methods and models, ensuring fair comparison. Since all prefixes of a trajectory share a common prompt, PMI credit computation benefits from KV-cache reuse and can be parallelized across trajectories; Further details on data collection and implementation are provided in Appendix F, and case studies are presented in Appendix G.

4.2 Main Results

Table 1 summarizes performance across three real-world applications. AgentBrew targets data efficiency rather than model scaling: it extracts stronger signal from limited, unlabeled interactions and is therefore complementary to advances in base model capability. Built on Qwen3-32B, AgentBrew achieves an average Acc of 18.2 and Score of 58.7, surpassing its own 235B variant (+2.3 Acc, +4.4 Score), demonstrating that targeted offline adaptation on a single environment can be more effective than general-purpose scaling alone. Absolute accuracy across all methods remains modest, reflecting the genuine difficulty of these real-world benchmarks, where even frontier proprietary models such as Gemini-3 Pro (57.1 / 80.8) and GPT-5 (50.0 / 76.8) have ample room for improvement.

Among training-based methods, AgentBrew consistently outperforms all baselines across the three environments. Compared to Vanilla SFT and Rejection Sampling, AgentBrew achieves improvements of +7.9 Acc / +10.6 Score and +5.9 Acc / +10.3 Score, respectively, showing that retrospectively realigning task instructions and extracting fine-grained per-action credit from noisy trajectories is more effective than either uniform supervision or trajectory-level quality filtering. The advantage is most pronounced on Notion, where Rejection Sampling yields an Acc of 0: the LLM judge fails to reliably assess task completion in complex stateful API interactions, causing the filter to reject nearly all trajectories and leaving insufficient data for effective training. AgentBrew achieves 14.2 Acc on the same corpus by bypassing verification entirely and instead recovering informative actions within imperfect trajectories. AgentBrew also outperforms TOUCAN (+6.3 Acc, +13.9 Score), which is trained on 1.5M trajectories synthesized across broad MCP environments, further confirming that a small amount of environment-specific data with fine-grained credit assignment is more effective than large-scale cross-environment trajectory synthesis.

4.3 Effect of PMI-Based Credit Assignment

We examine how PMI-Based credit assignment (§3.2.2) identifies informative actions within imperfect trajectories. All methods are trained on the same realigned pairs $\{(\hat{I}_n, \tau_n)\}$ from retrospective task inference (§3.2.1); only the per-action weighting strategy differs. We compare our PMI-based credit against three alternatives: *Uniform*, which assigns equal weight to all actions; *LLM Score*, which prompts π_{ref} to score each of its own actions; and *InfoPO* [14], which computes credit based on the information gain advantage over future actions. As shown in Figure 3(a-b), PMI-based credit consistently outperforms all alternatives on both GitHub and Notion. Notably, LLM Score drops to 7.1 / 54.7 on Notion, underperforming even Uniform (10.7 / 61.6), suggesting that LLM self-evaluation is unreliable for action-level quality assessment in complex stateful tool-use settings.

Beyond downstream performance, PMI-based credit provides a natural safeguard against imperfect task inference: when $\hat{I}_n$ is poorly aligned with the trajectory, actions provide little information about it, yielding uniformly low credits that are clamped to near-zero by Eq. (5) and ensuring that the trajectory contributes minimal gradient signal during training. We verify this by using GPT-5 to

classify each $(\hat{I}_n, \bar{\tau}_n)$ pair as well-aligned or misaligned. As shown in Figure 3(c-d), misaligned trajectories (blue) cluster at significantly lower mean and total credit compared to well-aligned ones (red) on both GitHub and Notion, confirming that PMI-based credit automatically suppresses noisy task inference without requiring any explicit quality filter.

4.4 Effect of Retrospective Task Inference

We verify the necessity of retrospective task inference (§3.2.1) by comparing two training configurations on Notion: one using the original task instructions $\{I_n\}$ and the other using the inferred instructions $\{\hat{I}_n\}$, both trained with PMI-based credit assignment. As shown in Table 2, training with $\hat{I}_n$ achieves an Acc of 14.2 and Score of 70.1, compared to 7.1 / 65.6 with the original I_n. The degradation under I_n arises because most trajectories fail to complete their original instruction, causing the policy to associate incomplete or erroneous action sequences with the stated goal (see Appendix G for qualitative examples). Although PMI-based credit assignment can down-weight misaligned trajectories (§4.3), credit is computed as the information gain of each action with respect to whatever instruction is provided; when the instruction itself is misaligned, even useful actions receive distorted credits measured against the wrong reference task. Retrospective task inference addresses this at the source by replacing I_n with $\hat{I}_n$ that reflects the trajectory's actual outcome, ensuring that credit computation operates on coherent instruction-trajectory pairs. The two mechanisms are thus complementary: task inference corrects *what* the trajectory is trained to achieve, while credit assignment determines *how much* each action contributes to that corrected goal.

Table 2: Effect of retrospective task inference on Notion.

Task Instruction	Acc.	Score
Original I_n	7.1	65.6
Inferred $\hat{I}_n$	**14.2**	**70.1**

4.5 Robustness to Noisy Trajectories

As an offline method, AgentBrew can in principle be applied to trajectories from diverse sources, including agent self-exploration and human usage logs, where data quality is inherently uncontrolled. Robustness to such quality variance is therefore a practical requirement. To evaluate this, we use GPT-5 to score each raw trajectory pair (I_n, τ_n) based on instruction-trajectory alignment and rank all trajectories by quality in descending order. We then train AgentBrew on progressively larger subsets: the top 15%, top 50%, top 75%, and the full corpus, where each expansion introduces trajectories of strictly lower quality than those already included.

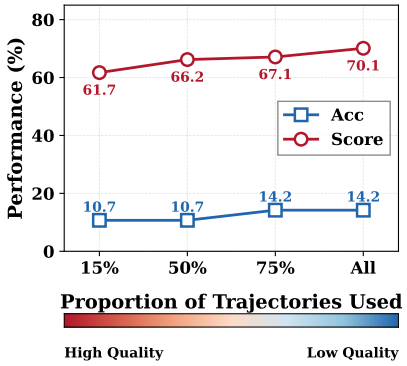


Figure 4: Performance on Notion when progressively adding lower-quality data.

As shown in Figure 4, both metrics improve monotonically as lower-quality data is added, with Acc increasing from 10.7 to 14.2 and Score from 61.7 to 70.1. Even adding the lowest-quality 25% of trajectories yields a further Score gain (67.1 $\rightarrow$ 70.1). This does not imply that lower-quality trajectories are inherently more valuable; rather, each additional trajectory, regardless of its overall completion quality, still contains individual actions that are informative for the inferred task. AgentBrew's credit mechanism extracts these useful portions while suppressing the rest, so the net effect of including more data is consistently positive. This allows practitioners to include trajectories from diverse and uncontrolled sources without degrading performance.

4.6 Offline Transferability Across Models

To test whether AgentBrew's distilled signal transfers across model sizes, we apply the data distilled by Qwen3-32B to train a 14B model via the same weighted training objective. As shown in Table 3, AgentBrew-14B achieves 20.2 average Acc / 55.3 Score, substantially outperforming the Qwen3-14B base model (+11.5 Acc / +12.5 Score), and on Notion nearly matches the 32B AgentBrew model (14.2 / 69.8 vs 14.2 / 70.1). Toucan-14B and AWM-14B also leverage larger models in their pipelines for trajectory synthesis and environment construction, yet show limited improvement over the base

Table 3: Offline transferability on 14B models. AgentBrew distills data collected by Qwen3-32B into a 14B model via the offline pipeline, without additional environment interaction.

Method	Training Paradigm	GitHub		Notion		Postgres		Avg.	
		Acc.	Score	Acc.	Score	Acc.	Score	Acc.	Score
Qwen3-14B	No additional training	7.1	48.2	0	48.8	19.0	31.3	8.7	42.8
Toucan-14B	SFT on 1.5M synthetic trajs.	0	45.2	0	42.5	0	11.4	0.0	33.3
AWM-14B	Online RL on synthetic envs.	7.1	45.4	0	44.5	23.8	38.0	10.3	42.6
AgentBrew-14B	**Offline transfer**	**17.9**	**56.3**	**14.2**	**69.8**	**28.6**	**40.0**	**20.2**	**55.3**

model, indicating that the transferability stems from the quality of AgentBrew's distilled signal rather than the capacity gap alone. This confirms that a single round of offline distillation can be reused to train smaller models, amortizing the cost of real-world API access.

5 Related Work

Training Data for Tool-Use Agents. A central challenge in training tool-use agents is obtaining high-quality (task, trajectory) pairs at scale. The prevailing forward approach first defines tasks, then collects trajectories by executing them with strong models and retaining only successful completions [26; 48; 24; 42; 34; 17]. An alternative reverse approach derives tasks from existing interactions: OS-Genesis [33] infers tasks from GUI state transitions, AgentTrek [41] uses web tutorials to guide trajectory synthesis, and Learn-by-interact [32] synthesizes trajectories from documentation and constructs instructions via backward abstraction of the interaction histories. Closer to our work, hindsight relabeling methods recover training signal from failed trajectories by replacing the original instruction with one the trajectory actually fulfills. This idea originates from HER [1] in classic RL; HSL [18] adapts it to LLM agents using environment rewards, and AgentHER [5] introduces failure classification with confidence gating. Across these lines of work, data quality is ensured through external signals such as execution verification, environment rewards, or LLM-based filtering. AgentBrew requires none of these, instead deferring quality control entirely to an information-theoretic credit assignment mechanism that operates on the trajectories themselves.

Policy Optimization for LLM Agents. On-policy reinforcement learning is widely used to train LLM agents in interactive environments, with methods such as GRPO [30], DAPO [46], WebRL [25], and DigiRL [3] relying on iterative rollouts and environment rewards. A central challenge is credit assignment, since trajectory-level rewards provide sparse supervision and cannot distinguish informative actions from redundant ones. To obtain finer-grained signals, IGPO [36] uses information gain on a ground-truth answer as turn-level reward, GiGPO [9] builds step-level groups from anchor states, iStar [19] infers step rewards from trajectory preferences, and AgentPRM [40] trains a process reward model via temporal-difference estimation. These methods remain on-policy and require repeated interaction with reward feedback. Offline methods reduce rollout cost: DPO [27], KTO [8], and RPO [45] learn from preference pairs, while Digi-Q [4] and OREO [37] learn value or Q-functions from offline trajectories. Yet they still depend on rewards, preferences, or learned value functions. In contrast, AgentBrew is fully offline and derives per-action training weights directly from the relationship between trajectories and their inferred tasks, without external reward signals or preference annotations.

6 Conclusion, Limitation, and Future Work

We presented AgentBrew, an offline framework that trains tool-use agents from a single batch of raw interactions without task verifiers or on-policy rollouts. By retrospectively inferring aligned instructions and decomposing per-action credit, AgentBrew extracts training signal from noisy trajectories that filtering-based methods would discard, enabling a 32B model to surpass Qwen3-235B across three real-world applications. Current limitations include evaluation on only three MCP applications, potential degradation of task inference on extremely long or chaotic trajectories, and the reliance on PMI as an informational proxy for action quality, which, despite structural alignment with actual contribution through the construction of $\hat{I}_n$ (§3.2.2), may not capture all causal effects in highly entangled action sequences.

A promising future direction is applying AgentBrew to real user interaction logs in production applications, which are naturally unlabeled and noisy. Retrospective task inference could recover user intent directly from operation sequences, and credit assignment could identify key actions within each session, enabling agents to learn from authentic human demonstrations at scale.

7 Acknowledgments

We thank Kuaishou Technology for its support and collaboration on this research.

References

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[2] Anthropic. Introducing the model context protocol. https://www.anthropic.com/news/model-context-protocol, November 2024.

[3] Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37:12461–12495, 2024.

[4] Hao Bai, Yifei Zhou, Li Erran Li, Sergey Levine, and Aviral Kumar. Digi-q: Learning q-value functions for training device-control agents. *arXiv preprint arXiv:2502.15760*, 2025.

[5] Liang Ding. Agenther: Hindsight experience replay for llm agent trajectory relabeling. *arXiv preprint arXiv:2603.21357*, 2026.

[6] Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, Jiajie Jin, Yutao Zhu, Hanbin Wang, Fangyu Lei, Qinyu Luo, Mingyang Chen, Zehui Chen, Jiazhan Feng, Ji-Rong Wen, and Zhicheng Dou. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence. *arXiv preprint*, 2026.

[7] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024.

[8] Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*, 2024.

[9] Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*, 2025.

[10] GitHub. Github mcp server. https://github.com/github/github-mcp-server, 2025. GitHub repository. Accessed: 2026-04-17.

[11] Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

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

[13] Shuo He, Lang Feng, Qi Wei, Xin Cheng, Lei Feng, and Bo An. Hierarchy-of-groups policy optimization for long-horizon agentic tasks. *arXiv preprint arXiv:2602.22817*, 2026.

[14] Fanqi Kong, Jiayi Zhang, Mingyi Deng, Chenglin Wu, Yuyu Luo, and Bang Liu. Infopo: Information-driven policy optimization for user-centric agents. *arXiv preprint arXiv:2603.00656*, 2026.

[15] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, et al. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *arXiv preprint arXiv:2411.07763*, 2024.

[16] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357, 2023.

[17] Yuetai Li, Huseyin A Inan, Xiang Yue, Wei-Ning Chen, Lukas Wutschitz, Janardhan Kulkarni, Radha Poovendran, Robert Sim, and Saravan Rajmohan. Simulating environments with reasoning models for agent training. *arXiv preprint arXiv:2511.01824*, 2025.

[18] Zichao Li, Gang Wu, Zichao Wang, Ruiyi Zhang, Wanrong Zhu, Ryan A. Rossi, Vlad I Morariu, and Jihyung Kil. Spinning straw into gold: Relabeling LLM agent trajectories in hindsight for successful demonstrations. In *The Fourteenth International Conference on Learning Representations*, 2026.

[19] Xiaoqian Liu, Ke Wang, Yuchuan Wu, Fei Huang, Yongbin Li, Junge Zhang, and Jianbin Jiao. Agentic reinforcement learning with implicit step rewards. *arXiv preprint arXiv:2509.19199*, 2025.

[20] Ziyang Luo, Zhiqi Shen, Wenzhuo Yang, Zirui Zhao, Prathyusha Jwalapuram, Amrita Saha, Doyen Sahoo, Silvio Savarese, Caiming Xiong, and Junnan Li. Mcp-universe: Benchmarking large language models with real-world model context protocol servers. *arXiv preprint arXiv:2508.14704*, 2025.

[21] Zhiyi Lyu, Jianguo Huang, Yanchen Deng, Steven Hoi, and Bo An. Let's revise step-by-step: A unified local search framework for code generation with llms. *arXiv preprint arXiv:2508.07434*, 2025.

[22] Notion. Notion: The ai workspace that works for you. https://www.notion.com/, 2025. Accessed: 2026-04-17.

[23] Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: towards llms as operating systems. 2023.

[24] Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, et al. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *arXiv preprint arXiv:2504.03601*, 2025.

[25] Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*, 2024.

[26] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

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

[28] Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024.

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

[30] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[31] Wentao Shi, Mengqi Yuan, Junkang Wu, Qifan Wang, and Fuli Feng. Direct multi-turn preference optimization for language agents. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2312–2324, 2024.

[32] Hongjin Su, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan Ö Arık. Learn-by-interact: A data-centric framework for self-adaptive agents in realistic environments. *arXiv preprint arXiv:2501.10893*, 2025.

[33] Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5555–5579, 2025.

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

[35] Dunwei Tu, Hongyan Hao, Hansi Yang, Yihao Chen, Yi-Kai Zhang, Zhikang Xia, Yu Yang, Yueqing Sun, Xingchen Liu, Furao Shen, et al. Scaleenv: Scaling environment synthesis from scratch for generalist interactive tool-use agent training. *arXiv preprint arXiv:2602.06820*, 2026.

[36] Guoqing Wang, Sunhao Dai, Guangze Ye, Zeyu Gan, Wei Yao, Yong Deng, Xiaofeng Wu, and Zhenzhe Ying. Information gain-based policy optimization: A simple and effective approach for multi-turn llm agents. *arXiv preprint arXiv:2510.14967*, 2025.

[37] Huaijie Wang, Shibo Hao, Hanze Dong, Shenao Zhang, Yilin Bao, Ziran Yang, and Yi Wu. Offline reinforcement learning for llm multi-step reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8881–8893, 2025.

[38] Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent world model: Infinity synthetic environments for agentic reinforcement learning. *arXiv preprint arXiv:2602.10090*, 2026.

[39] Zijian Wu, Xiangyan Liu, Xinyuan Zhang, Lingjun Chen, Fanqing Meng, Lingxiao Du, Yiran Zhao, Fanshi Zhang, Yaoqi Ye, Jiawei Wang, et al. Mcpmark: A benchmark for stress-testing realistic and comprehensive mcp use. *arXiv preprint arXiv:2509.24002*, 2025.

[40] Zhiheng Xi, Chenyang Liao, Guanyu Li, Yajie Yang, Wenxiang Chen, Zhihao Zhang, Binghai Wang, Senjie Jin, Yuhao Zhou, Jian Guan, et al. Agentprm: Process reward models for llm agents via step-wise promise and progress. *arXiv preprint arXiv:2511.08325*, 2025.

[41] Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. *arXiv preprint arXiv:2412.09605*, 2024.

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

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

[44] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

[45] Yueqin Yin, Zhendong Wang, Yi Gu, Hai Huang, Weizhu Chen, and Mingyuan Zhou. Relative preference optimization: Enhancing llm alignment through contrasting responses across identical and diverse prompts. *arXiv preprint arXiv:2402.10958*, 2024.

[46] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

[47] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

[48] Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Quoc Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, et al. xlam: A family of large action models to empower ai agent systems. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 11583–11597, 2025.

[49] Wangchunshu Zhou, Yuchen Eleanor Jiang, Peng Cui, Tiannan Wang, Zhenxin Xiao, Yifan Hou, Ryan Cotterell, and Mrinmaya Sachan. Recurrentgpt: Interactive generation of (arbitrarily) long text. *arXiv preprint arXiv:2305.13304*, 2023.

A Pseudo Code

We provide the complete AgentBrew pipeline in Algorithm 1. The procedure consists of three stages: raw experience collection (§3.1), experience distillation (§3.2), and PMI-weighted policy training (§3.3). Stage 1 is the only phase that requires interaction with the live environment $\mathcal{E}$; Stages 2 and 3 proceed entirely offline using only the collected corpus and a reference language model π_{ref}.

Algorithm 1 AgentBrew: Offline Tool-Use Agent Learning from Raw Trajectories

Input: Target environment $\mathcal{E}$ with tool set $\mathcal{F}$, base policy π_θ, reference model π_{ref}, max turns $T_{\max}$
Output: Trained policy π_θ

— Stage 1: Raw Experience Collection —
Generate high-level task categories $\mathcal{G}$ from tool descriptions of $\mathcal{F}$
for each category $g_m \in \mathcal{G}$ **do**
  Explore $\mathcal{E}$ via tool calls to retrieve live environmental information
  Propose grounded task instructions $\{I_n\}$ based on retrieved information
for each task $I_n \in \mathcal{I}$ **do**
  Execute I_n with π_θ in $\mathcal{E}$, producing trajectory $\tau_n = \{(s_t, a_t, o_t)\}_{t=0}^{T_n}$
  if $T_n = T_{\max}$ (no natural termination) **then**
    Discard τ_n
Collect raw corpus $\mathcal{D}_{\text{raw}} = \{(I_n, \tau_n)\}_{n=1}^{N}$ ▷ No quality filtering

— Stage 2: Experience Distillation (Offline) —
for each $(I_n, \tau_n) \in \mathcal{D}_{\text{raw}}$ **do**
  // Retrospective Task Inference
  Remove chain-of-thought traces: $\bar{\tau}_n = \{(c_t, o_t)\}_{t=0}^{T_n}$
  Infer revised instruction: $\hat{I}_n \leftarrow \text{LLM}(\bar{\tau}_n)$ ▷ Goal-oriented, evidence-grounded

  // PMI-Based Credit Assignment
  Compute unconditional NLL: $\mathcal{L}_\emptyset = -\log \pi_{\text{ref}}(\hat{I}_n)$
  for $t = 0, 1, \ldots, T_n$ **do**
    Compute conditional NLL: $\mathcal{L}_t = -\log \pi_{\text{ref}}(\hat{I}_n \mid c_0, o_0, \ldots, c_t, o_t)$
  $w_0 \leftarrow \mathcal{L}_\emptyset - \mathcal{L}_0$
  for $t = 1, \ldots, T_n$ **do**
    $w_t \leftarrow \mathcal{L}_{t-1} - \mathcal{L}_t$ ▷ Per-action PMI credit

  // Weight Normalization
  for $t = 0, 1, \ldots, T_n$ **do**
    $\tilde{w}_t \leftarrow 2 \cdot \max(w_t, 0) \;/\; \max_{0 \le j \le T_n} w_j$ ▷ Normalize to $[0, 2]$

— Stage 3: PMI-Weighted Policy Training (Offline) —
Optimize π_θ by minimizing:

$$\mathcal{J}(\theta) = -\sum_{n=1}^{N} \sum_{t=0}^{T_n} \tilde{w}_t^{(n)} \cdot \log \pi_\theta(a_t \mid \hat{I}_n, s_t)$$

B Environment Details

We evaluate AgentBrew on three real-world MCP applications that span distinct interaction paradigms: code collaboration (GitHub), knowledge management (Notion), and relational database administration (PostgreSQL). Table 4 lists the complete tool set exposed by each environment.

GitHub. The GitHub environment exposes 41 tools covering the full spectrum of platform operations: repository management, file operations, issue tracking, pull request workflows, branch and commit inspection, CI/CD workflow management, code and user search, and security scanning (code scanning, Dependabot, secret scanning, and security advisories). Tasks in this environment require composing long action sequences across these tool categories. For example, forking a repository,

creating a feature branch, pushing file changes, opening a pull request, and performing a code review, all within a single trajectory. The live state includes cross-referenced issues, pull requests, commits, and branches, making precise entity resolution critical.

Notion. The Notion environment provides 19 tools organized around four core object types: users, blocks, pages, and databases, along with search and comment capabilities. Despite the smaller tool count, Notion tasks are challenging because of deeply nested content structures: pages contain blocks that may themselves contain child blocks or inline database references, and databases define typed schemas with relations and rollups linking to other databases. Operations such as inserting a block at a precise position, restructuring a page layout, or synchronizing rows across related databases require the agent to maintain an accurate mental model of the workspace hierarchy throughout the interaction.

PostgreSQL. The PostgreSQL environment exposes 9 tools that combine standard SQL execution with database administration utilities. Beyond `execute_sql` for arbitrary SQL statements, the environment provides schema inspection tools (`list_schemas`, `list_objects`, `get_object_details`), query analysis tools (`explain_query`, `get_top_queries`), and health diagnostics (`analyze_workload_indexes`, `analyze_query_indexes`, `analyze_db_health`). Although the tool set is compact, the complexity arises from the SQL reasoning itself: tasks range from multi-table data migrations and schema design to query optimization and security policy implementation, requiring the agent to generate correct, multi-step SQL workflows grounded in the actual database schema and data.

Table 4: Tool sets exposed by each environment.

Env.	#Tools	Tools
GitHub	41	`create_repository`, `get_repository`, `list_repositories`, `fork_repository`, `get_repository_languages`, `get_file_contents`, `create_or_update_file`, `delete_file`, `push_files`, `create_issue`, `get_issue`, `list_issues`, `update_issue`, `add_issue_comment`, `create_pull_request`, `get_pull_request`, `list_pull_requests`, `get_pull_request_files`, `create_pull_request_review`, `merge_pull_request`, `create_branch`, `list_branches`, `list_commits`, `list_workflow_runs`, `get_workflow_run`, `list_workflow_jobs`, `cancel_workflow_run`, `search_repositories`, `search_code`, `search_issues`, `search_users`, `list_code_scanning_alerts`, `get_code_scanning_alert`, `list_dependabot_alerts`, `get_dependabot_alert`, `list_secret_scanning_alerts`, `get_secret_scanning_alert`, `list_global_security_advisories`, `list_org_repository_security_advisories`, `list_repository_security_advisories`, `create_pull_request_with_copilot`
Notion	19	`get-user`, `get-users`, `get-self`, `post-database-query`, `post-search`, `get-block-children`, `patch-block-children`, `retrieve-a-block`, `update-a-block`, `delete-a-block`, `retrieve-a-page`, `patch-page`, `post-page`, `create-a-database`, `update-a-database`, `retrieve-a-database`, `retrieve-a-page-property`, `retrieve-a-comment`, `create-a-comment`
Postgres	9	`list_schemas`, `list_objects`, `get_object_details`, `execute_sql`, `explain_query`, `get_top_queries`, `analyze_workload_indexes`, `analyze_query_indexes`, `analyze_db_health`

C Task Proposal

Environment preparation. Each target environment $\mathcal{E}$ requires a diverse substrate of real-world content to ensure that proposed tasks are grounded in meaningful state. For **GitHub**, we curate 200 public repositories spanning a variety of programming languages (Python, JavaScript, Go, Rust, etc.) and project types (web applications, CLI tools, data pipelines, machine learning libraries), providing a rich landscape of issues, pull requests, branches, and commit histories for the agent to interact with. For **Notion**, we collect 180 free templates from the official Notion template gallery,[1] covering domains such as project management, personal planning, knowledge bases, and team collaboration. These templates provide pre-populated pages, databases, and relational structures that mirror realistic workspace configurations. For **PostgreSQL**, we assemble 145 relational databases sourced from established open-source benchmarks including BIRD [16] and Spider [15], which collectively span domains such as retail, healthcare, education, sports, and enterprise operations.

[1] https://www.notion.com/templates

This diversity ensures that the subsequent task proposal stage can generate instructions grounded in entities and configurations that actually exist within each environment.

High-level task category generation. Given the tool set $\mathcal{F}$ of the target environment $\mathcal{E}$, we prompt the agent to generate a set of high-level task categories $\mathcal{G} = \{g_1, g_2, \ldots, g_M\}$ that cover representative human intents supported by $\mathcal{E}$. The agent receives the names and descriptions of all available tools as input and produces categories that reflect common multi-step usage patterns, each requiring the composition of multiple tools. Table 5 lists the generated categories for each environment. GitHub yields 14 categories spanning repository lifecycle management, code collaboration, and project planning; Notion produces 12 categories covering database operations, page editing, and content synthesis; PostgreSQL results in 15 categories ranging from data migration and analytics to security policy design. The prompt template used for category generation is shown below.

Prompt: Task Category Generation

You are an expert task designer for evaluating AI agents that interact with real-world applications via tool-use APIs.
You are given the complete tool set of the `{environment_name}` environment. Each tool is described by its name, purpose, and parameter specification:
`{tool_descriptions}`
Your goal is to generate a diverse set of high-level task categories that represent realistic human intents on this platform. Each category should:
(1) Require **composing multiple tools** in a meaningful sequence (not single-tool invocations).
(2) Reflect a **concrete, goal-oriented workflow** that a real user would perform (e.g., "migrate data between tables with validation", not "use the search API").
(3) Be **distinct** from other categories in terms of the tool combinations and reasoning patterns involved.
(4) Cover **diverse difficulty levels**, from straightforward CRUD operations to complex multi-step workflows involving conditional logic, cross-entity coordination, or synthesis.

For each category, provide:
– **Title**: A concise name for the category.
– **Goal**: A one-sentence description of what a user aims to accomplish.

Generate `{num_categories}` categories that collectively maximize the coverage of the tool set.

Table 5: Task categories generated for each environment.

GitHub (14)	Notion (12)	PostgreSQL (15)
1. Project bootstrap & self-review	1. Database creation & schema design	1. Bulk migration & transformation
2. Multi-source code integration	2. Cross-database relation & sync	2. Hierarchy-aware CRUD & integrity
3. Ecosystem metrics & reporting	3. Batch query, filter & migration	3. Dashboard reporting & aggregates
4. Decision-based comparative forking	4. Batch updates & coordinated edits	4. SQL query debugging & fixing
5. Simulated issue automation	5. Aggregate computation & dashboards	5. Query optimization & indexing
6. Fork-and-fix contribution	6. Multi-source query & synthesis	6. Operational workflow management
7. Cross-repo dependency injection	7. Page layout & column restructuring	7. Descriptive reporting & statistics
8. Epic & sub-issue planning	8. Block insertion & format matching	8. Multi-factor performance analytics
9. Release audit & upgrade docs	9. Template filling & nested editing	9. Schema buildout & seed data
10. Repo maintenance & cleanup	10. Content deletion & archival	10. Retention & churn analysis
11. Code archaeology & arch. docs	11. Style & color batch updates	11. Executive dashboard automation
12. Legacy branch backporting	12. Itinerary & planner composition	12. Structural & bottleneck analysis
13. Research framework scaffolding		13. Consistency enforcement
14. PR queue management & hygiene		14. Security policy & row-level access
		15. Transactional procedures & audit

Grounded task instruction generation. For each category $g_m \in \mathcal{G}$, a task generation agent conducts a lightweight exploratory interaction with $\mathcal{E}$ to ground the abstract category in concrete environmental state. The agent follows a ReAct-style loop: it first invokes a single read-only tool call (e.g., listing repository contents, querying database schemas, or browsing page structures) to discover real entities such as repository names, file paths, issue numbers, page IDs, or table columns. Based on the retrieved information, the agent then formulates a specific task instruction I_n written as a natural user request that references only verified identifiers. This grounding step is critical because tasks that reference nonexistent resources (e.g., a deleted branch or a missing database entry) would produce unexecutable trajectories during the subsequent data collection phase. To maintain diversity, the agent

also receives an example task from a different category as a style reference, ensuring consistent tone and specificity across the generated task set. The prompt template is shown below.

Prompt: Grounded Task Instruction Generation

You are a ReAct (Reasoning and Acting) agent specializing in Task Engineering. You are designing a single, high-quality task for LLMs that interact with real-world applications via tool-use.

Tool Restrictions: You may ONLY call read/search tools to inspect the target environment — do NOT create, modify, or delete anything. Call exactly 1 tool to ground yourself, then immediately output the final task.

What the generated task CAN include: The task may involve any operation supported by the environment (reading, creating, updating, deleting resources). Write operations must target user-owned resources (newly created or forked).

Your Job:
(1) Read the **Workflow Pattern** (`{task_category}`) to understand the type of task to generate.
(2) Read the **Example Task** (`{example_task}`) to learn the desired tone, style, and level of detail.
(3) Call 1 read-only tool to inspect the target environment and discover concrete entities.
(4) Write a NEW task that follows the same workflow pattern, mimics the example's tone, and is grounded in the environment's real state.

Available tools: `{tool_descriptions}`

Task Output Constraints:
(1) Output exactly one task. Do not generate multiple tasks or sub-task lists.
(2) Write it as a natural user request — same tone and style as the example. No formal specs, no numbered checklists, no headings.
(3) Do not mention "tools" or "APIs" in the final task text.
(4) Include concrete identifiers confirmed via the tool call.

Output Format: A JSON object with a `thought` field (verification reasoning) and either an `action` field (to call a read-only tool) or an `answer` field (the final natural-language task instruction).

D PMI-Based Credit Assignment

For each trajectory, we first infer a hindsight instruction $\hat{I}$ that describes the outcome actually supported by the observed tool interactions. We then compute per-action credit by measuring how much each trajectory prefix reduces the negative log-likelihood of $\hat{I}$ under a reference model π_{ref}. Let $\bar{\tau}_{\leq t}$ denote the serialized tool-interaction prefix up to step t, and let $\hat{I} = (\hat{i}_1, \ldots, \hat{i}_m)$ be the tokenized hindsight instruction. In implementation, we compute the average token negative log-likelihood:

$$L_t = -\frac{1}{m} \sum_{k=1}^{m} \log \pi_{\text{ref}} \left(\hat{i}_k \mid \hat{i}_{<k}, \bar{\tau}_{\leq t} \right).$$

The empty-prefix loss is denoted by L_{-1} and is computed in the same way without conditioning on any observed tool interactions:

$$L_{-1} = -\frac{1}{m} \sum_{k=1}^{m} \log \pi_{\text{ref}} \left(\hat{i}_k \mid \hat{i}_{<k} \right).$$

The raw PMI credit for action t is the reduction in hindsight-instruction NLL after revealing that action:

$$w_t = L_{t-1} - L_t.$$

Thus, positive credit indicates that the action makes the inferred instruction easier to predict, while zero or negative credit indicates that the action is redundant or uninformative with respect to the inferred instruction. Before training, we clamp negative credits and normalize weights within each trajectory:

$$\tilde{w}_t = 2 \cdot \frac{\max(w_t, 0)}{\max_j w_j}.$$

If all raw credits in a trajectory are non-positive, we set all normalized weights to zero.

Empirical statistics of normalized weights. Across the full training corpus, the mean normalized weight $\tilde{w}_t$ is 0.75, with per-environment values of 0.81 (GitHub), 0.71 (Notion), and 0.77 (PostgreSQL). This value is below the uniform baseline of 1.0 because negative credits are clamped to zero and many actions within each trajectory receive near-zero credit, reflecting the intended selective effect of PMI weighting. The relative ranking of per-action credits, rather than the absolute scale, is the primary mechanism through which PMI weighting steers the policy toward informative actions.

E Retrospective Task Inference

As described in §3.2.1, we infer a revised instruction $\hat{I}_n$ from each trajectory's reduced form $\bar{\tau}_n = \{(c_t, o_t)\}_{t=0}^{T_n}$, which retains only tool calls and environment responses. We use Qwen3-32B for this inference, consistent with the rest of the AgentBrew pipeline. Before prompting, we extract a structured change summary from $\bar{\tau}_n$ by identifying successful state-changing operations (e.g., pages created, rows inserted, SQL objects defined) and separating them from read-only or failed steps. Both the change summary and the full reduced trajectory are provided as input.

The prompt follows a consistent template across all three environments, instantiated with environment-specific terminology and examples. The general template is shown below.

Prompt: Retrospective Task Inference

You revise benchmark-style `{environment_name}` tasks from agent tool trajectories.
Your job is to minimally edit the ORIGINAL TASK so it matches what the agent actually completed.
Main goal:

- Remove or weaken unsupported subtasks.
- The revised task must be highly consistent with the trajectory.

Use the trajectory as evidence, with priority on: (1) successful state-changing operations; (2) query/retrieve steps only when they help identify objects involved in successful writes.
Rules:

- Output only the revised task text.
- Preserve the original high-level task type (e.g., migration stays migration, analysis stays analysis).
- If the agent only completed part of a multi-step task, keep only the completed subset.
- Do not claim outcomes (e.g., performance improvements, verification success) unless directly supported.
- Do not invent entities, artifacts, or results not evidenced by the trajectory.
- Prefer deleting unsupported details over replacing them with new speculative ones.
- Do not preserve strong claims (`all`, `each`, `exact count`, `verbatim`) unless directly supported by successful operations.

ORIGINAL TASK:
`{original_task}`
OBSERVED CHANGES:
`{change_summary}`
TRAJECTORY:
`{trajectory_str}`

F Experiment Details

F.1 Context Management

Real-world tool-use interactions frequently produce trajectories exceeding 100K tokens, as each tool call may return verbose API responses (e.g., full page content in Notion or lengthy query results in PostgreSQL). To handle this within finite context windows, we adopt a sliding-window memory mechanism inspired by prior work on memory-augmented agents [23; 49].

At each step t, the agent's context is composed of two parts: (1) a *long-term memory* consisting of compressed summaries of all steps prior to the recent window, and (2) a *short-term context* containing

the full, uncompressed content of the three most recent steps $(s_{t-2}, a_{t-2}, o_{t-2}), \ldots, (s_t, a_t, o_t)$. This design preserves fine-grained details for immediate decision-making while retaining high-level progress information from earlier steps.

Concretely, after each step t, we prompt the LLM to generate a concise summary of the current step conditioned on the task instruction, all previous step summaries, and the full content of step t. The summary captures what was attempted, key results or facts discovered, important constraints or decisions, and concrete identifiers (paths, URLs, IDs). Each summary is kept under 500 words and is appended to the long-term memory buffer. When constructing the context for step $t{+}1$, all summaries for steps 0 to t are concatenated as the long-term memory prefix, followed by the full content of steps $t{-}2$, $t{-}1$, and t.

This mechanism is applied uniformly across all methods and models in our experiments, ensuring that performance differences reflect the training approach rather than context management advantages. The prompt template used for step summarization is provided below.

Prompt: Step-Level Summarization

You are a step-level summarizer for a multi-step ReAct-style agent.
Your task: Given the overall question, all past step summaries, and the full content of the current step, extract only the information from this step that will be useful for understanding what has already been done and making better decisions in future steps.

Overall Question: `{question}`
Past Step Summaries: `{past_summaries}`
Current Step Index: `{step_index}`
Full Content of the Current Step: `{step_full_content}`

Produce a concise summary of this current step only, following these rules:

- Focus on: what was attempted or executed; key results or facts discovered; important constraints, assumptions, or decisions; identifiers, paths, or URLs needed later.
- Connect to the question and past progress when relevant, but do not re-summarize previous steps.
- Omit generic reasoning boilerplate, tool framework noise, and large raw outputs (summarize their essential points instead).
- Use fewer than 500 words.

F.2 Training Hyperparameters

We train the policy π_θ using the PMI-weighted SFT objective (Eq. 6) implemented as a custom weighted cross-entropy loss. For each training sample, the per-action credit $\tilde{w}_t^{(n)}$ is attached as a sample-level weight; during forward passes, the trainer computes per-sample token-averaged cross-entropy and scales it by the corresponding weight before aggregating across the batch. During evaluation, weights are disabled to ensure comparable eval loss across runs.

We fine-tune Qwen3-32B using DeepSpeed ZeRO-Stage 3 with the following configuration: learning rate 5×10^{-6} with cosine scheduling and 10% warmup, per-device batch size 2 with gradient accumulation over 4 steps (effective batch size 64), weight decay 0.1, maximum sequence length 20,000 tokens, BF16 mixed precision, and gradient checkpointing enabled. Training runs for 1 epoch over the combined corpus of all three environments. Samples with zero normalized credit ($\tilde{w}_t^{(n)} = 0$) are filtered out before training, as they contribute no gradient signal. The 14B transfer experiments (§4.6) use identical hyperparameters and training data.

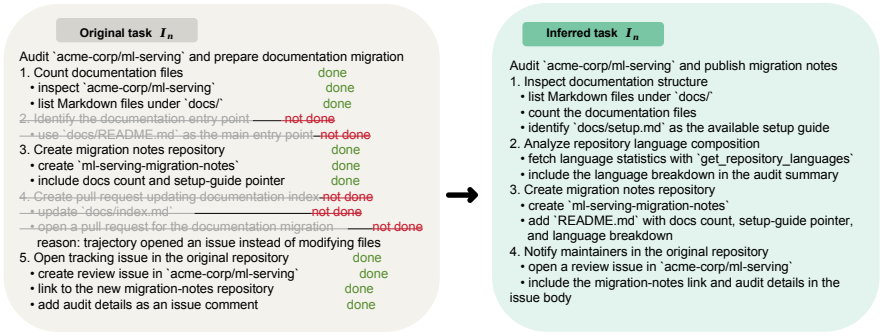
F.3 Evaluation Protocol

We adopt the evaluation framework from MCP-Universe [20]. Each test task is paired with a programmatic verifier that inspects the post-execution environment state (e.g., created resources, modified database rows, updated page content) against a ground-truth specification. The verifier produces a per-task score in $[0, 1]$ reflecting partial completion, and **Acc** is the fraction of tasks achieving a score of 1.0 (full completion). **Score** is the average per-task score across all test tasks.

G Case Studies

We provide qualitative case studies across three real-world tool-use environments: GitHub, Notion, and PostgreSQL. These examples illustrate how AgentBrew distills useful supervision from imperfect raw trajectories. In each case, retrospective task inference rewrites the original instruction into an aligned task that better matches the trajectory's observable outcome, while PMI credit assignment identifies which actions are useful for policy learning.

A Retrospective task inference

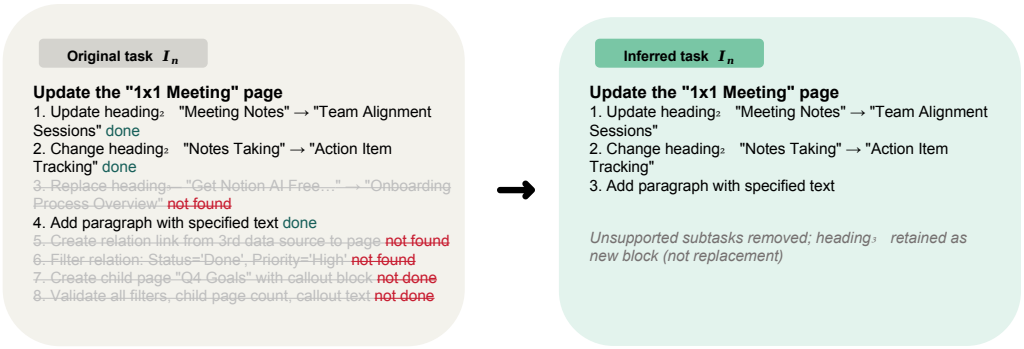


B PMI credit assignment

Action	PMI credit w_t
1. get_repository repo: acme-corp/ml-serving → Located the target repository and default branch	+0.058
2. get_file_contents path: docs/ → Listed 5 Markdown files under docs/	+0.196
3. get_file_contents path: docs/README.md → ERROR: file not found	-0.138
4. get_file_contents path: docs/setup.md → Found the available setup guide	+0.174
5. get_repository_languages repo: acme-corp/ml-serving → Collected language breakdown	+0.126
6. list_branches Redundant check of repository branches	+0.006
7. create_repository name: ml-serving-migration-notes → Created the migration-notes repository	+0.218
8. create_or_update_file ★ repo: ml-serving-migration-notes path: README.md → Wrote docs count, setup-guide pointer, and language summary	+0.402
9. create_issue repo: acme-corp/ml-serving title: Review documentation migration notes → Opened a review issue in the original repository	+0.151
10. add_issue_comment repo: acme-corp/ml-serving issue: #128 → Added migration-notes link and audit details	+0.087

Figure 5: Case study on the GitHub environment.

A Retrospective task inference

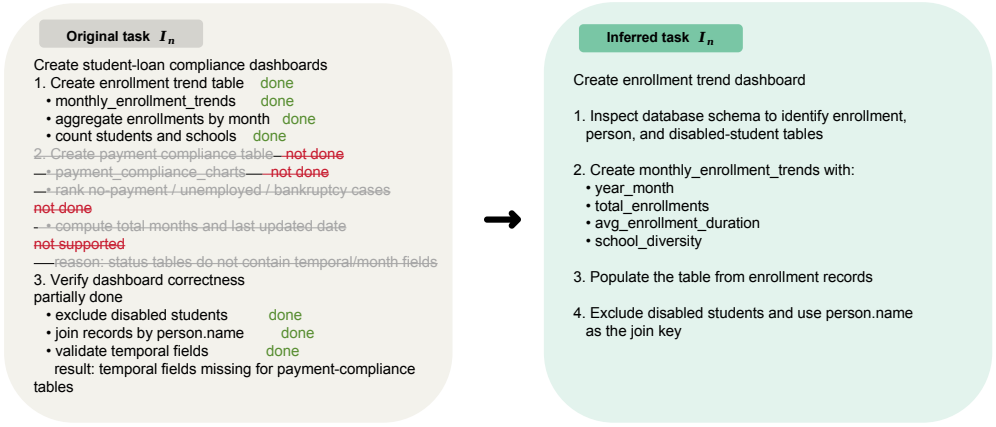


B PMI credit assignment

Action	PMI credit w_t
1. post-search Search "1x1 Meeting" → found page	0.089
2. update-a-block Placeholder ID → 400 validation error	-0.35
3. get-block-children Discovered page structure: headings, databases, blocks	0.256
4. update-a-block "Meeting Notes" → "Team Alignment Sessions" ✓	0.03
5. update-a-block "Notes Taking" → "Action Item Tracking" ✓	0.11
6. get-block-children Explore column_list → 3 columns	0.001
7. get-block-children Explore column 1 → unsupported button	0.028
8. get-block-children Explore column 2 → unsupported button	0.009
9. get-block-children Explore column 3 → unsupported button	-0.056
10. patch-block-children ★ Added heading₃ "Onboarding Process Overview" + paragraph text	0.397
11. post-search Search "third data source" → empty results	0.026
12. post-search Search "data source with 8 rows" → empty results	0.037

Figure 6: Case study on the Notion environment.

A Retrospective task inference



B PMI credit assignment

Action	PMI credit w_t
1. list_objects Found tables: enrolled, person, disabled	+0.072
2. get_object_details(enrolled) Found columns: name, school, month	+0.186
3. get_object_details(person) Confirmed join key: person.name	+0.064
4. get_object_details(disabled) Confirmed exclusion table: disabled.name	+0.058
5. execute_sql First CREATE/INSERT attempt → SQL syntax error	-0.214
6. execute_sql ★ Created and populated monthly_enrollment_trends	+0.421
7. list_objects Verified monthly_enrollment_trends exists	+0.043
8. execute_sql Attempted payment_compliance_charts → missing month	-0.167
9. get_object_details(no_payment_due) Confirmed no temporal field for ranking	+0.031
10. get_object_details(unemployed) Confirmed no temporal field for ranking	+0.018
11. get_object_details(bankruptcy) Confirmed no temporal field for ranking	+0.015
12. list_objects Redundant schema check	+0.002

Figure 7: Case study on the PostgreSQL environment.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the paper's main contribution as an offline framework for training tool-use agents from raw real-world trajectories without task verifiers or iterative on-policy rollouts. These claims are supported by the methodology in Section 3 and the experiments on GitHub, Notion, and PostgreSQL in Section 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: The paper discusses limitations in Section 6, including evaluation on only three MCP applications and possible degradation of retrospective task inference on extremely long or chaotic trajectories.

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present formal theoretical results such as theorems or lemmas. The PMI decomposition used for credit assignment is a methodological derivation based on the chain rule of PMI and is described in Section 3.2.2.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper describes the environments, models, baselines, evaluation benchmarks, data collection procedure, and training objective in Sections 3.3 and 4, with additional implementation details in the appendix. The evaluation uses separate training and test environments to prevent data leakage.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide anonymized code, data processing scripts, training and evaluation scripts, and instructions for reproducing the main experimental results in the supplemental material.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).

- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4.1 specifies the evaluated environments, models, baselines, benchmark sources, separation between training and evaluation environments, corpus sizes, and memory mechanism. Additional data collection and implementation details are provided in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We report results averaged across three environments but do not include error bars from multiple runs due to the high computational cost of real-world API interactions and GPU-intensive training. Each training run on 4×H100 GPUs and the associated real-world API costs make repeated runs impractical.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).

- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The current paper describes the models, data sizes, and GPU type in the appendix F.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The research uses benchmarked MCP environments and separated training/evaluation instances, without human-subject experiments or personally identifiable user data. The experiments are designed to avoid data leakage and to respect the terms of the evaluated platforms.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This research aims to advance the fields of large language models. While our work may have various societal implications, we have chosen not to emphasize any specific impacts within the scope of this paper.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate

to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.

- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a high-risk pre-trained foundation model, image generator, or scraped internet dataset. The method trains environment-specific tool-use agents, and the current submission does not include a public release of sensitive real-world interaction logs.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper cites the existing models, benchmarks, tools, and MCP environments used in the experiments, including Qwen3, mcp-universe, mcpmark, GitHub MCP, Notion, and PostgreSQL-related environments. The experiments use these assets for research evaluation and report the relevant sources in the references and appendix.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper introduces a training framework and collected trajectory corpora for experiments, but the current submission does not present a new public dataset, benchmark, or released model as a standalone asset. If anonymized data or code is released later, documentation and usage instructions will be provided with the release.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects. All trajectories are collected through agent interactions with benchmarked tool-use environments.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human-subject studies, crowdsourcing, or participant data collection. Therefore, IRB approval or equivalent review for human-subject research is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLM usage is central to the method: Qwen3-32B is used for raw experience collection and experience distillation, and a reference language model is used to compute PMI-based action credits. These uses are described in Sections 3 and 4.1, with prompt details provided in the appendix.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.