WHEN UNLEARNING MEETS THE HARNESS: HELPFUL ASSISTANCE, HARMFUL RECOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) unlearning aims to mitigate privacy and safety risks by selectively removing targeted knowledge or capabilities while preserving useful behavior. In deployment, LLM agents rely on harnesses that provide reasoning guidance and output repair to improve task execution. However, the security issues of this assistance remain insufficiently understood, leaving a substantial gap between model-level unlearning and robustness in deployed agents. Using tool unlearning as a concrete setting, we identify a failure mode *harness-mediated recovery*: harness assistance intended to improve task performance restores apparently suppressed tool-use capabilities without changing model parameters, optimizing adversarial prompts, or retrieving forgotten content. Experiments on ToolAlpaca and the Berkeley Function Calling Leaderboard (BFCL) reveal two mechanisms for this recovery: reasoning assistance before generation elicits suppressed tool calls, while schema repair after generation reconstructs executable calls from malformed outputs. Therefore, we formulate unlearning robustness at the level of the complete model–harness system and introduce the *Harness Recovery Rate* (HRR) to quantify recovery among forget-set examples that fail under the default interface. We further propose *Harness-Aware Robust Unlearning* (HARU), which incorporates harness-recoverable behaviors into the unlearning objective. Across benchmarks, HARU strengthens robustness to harness-mediated recovery and achieves strong forgetting while preserving substantial retained-tool utility. Our findings further demonstrate that harness environments constitute a critical and underexplored security dimension, where deployment-time interactions can fundamentally alter intended security properties.

1 INTRODUCTION

Large language models (LLMs) support agents that reason, use tools, and perform complex tasks in external environments Yao et al. (2023); Yang et al. (2024). However, training and adaptation can also lead these models to retain sensitive information or acquire undesirable capabilities, creating privacy and safety concerns. *LLM unlearning* seeks to address these concerns by selectively removing targeted knowledge or capabilities while preserving the model's remaining utility Jang et al. (2023); Yao et al. (2024). Recent methods demonstrate that parameter updates can substantially suppress targeted responses while retaining performance on unrelated tasks Yao et al. (2024); Zhang et al. (2024). These advances motivate the use of unlearning to control undesirable behaviors in LLM-based agents.

As agents undertake increasingly complex tasks, they rely on *agent harnesses* to improve performance and execution reliability Meng et al. (2026). A harness is the software layer that manages the model's context, interprets its outputs, and coordinates interaction with external environments. By supplying reasoning guidance, validating outputs, and recovering from execution errors, a harness can improve task performance without changing the underlying model parameters Yang et al. (2024); Lewis (2026); Yao et al. (2026). The behavior of a deployed agent therefore depends on both the model and the assistance provided by its harness.

Although harness assistance improves execution reliability, its implications for the safety of deployed agents remain insufficiently understood. Prior work shows that high task utility can coexist with substantial security failures and that security outcomes vary across model–harness configura-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

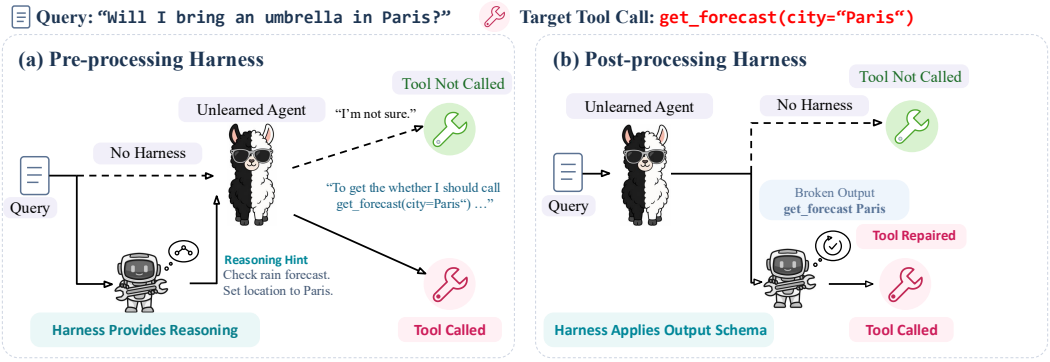

Figure 2: Two mechanisms of harness-mediated recovery for the same query and target tool. (a) Before generation, reasoning guidance enables the unlearned model to produce the target call. (b) After generation, schema repair converts a malformed output into an executable call. Dashed paths indicate failures under the default interface; solid paths indicate recovery with harness assistance. Model parameters remain fixed in both cases.

tions Bai et al. (2026). These observations motivate examining how harness assistance interacts with interventions intended to make models safer. In particular, it remains unclear whether the effects of unlearning persist when a model operates within a harness that guides its reasoning and processes its outputs.

The closest concurrent work, Agentic Tool Unlearning (ATU; Chen et al. (2026)), examines recovery through tools such as web search that can directly supply forgotten information. However, it does not establish whether ordinary helpful harness assistance can undermine forgetting without retrieving forgotten content (see **Appendix B** for further discussion). This gap motivates our central question:

Can ordinary harness assistance recover capabilities suppressed by unlearning?

We investigate this question through *tool unlearning*, which aims to remove the learned ability to invoke designated tools while preserving other tool-use capabilities Cheng & Amiri (2025). Tool use provides a concrete setting for examining this interaction because the harness connects model generation to executable actions, allowing recovery to be measured through successful target-tool calls. Our initial experiments on ToolAlpaca Tang et al. (2023) reveal substantial recovery with no changes to the unlearned model's parameters, as shown in **Figure 1**. We term this failure mode *harness-mediated recovery*: assistance intended to improve task execution restores tool-use behaviors that appear suppressed under the default interface.

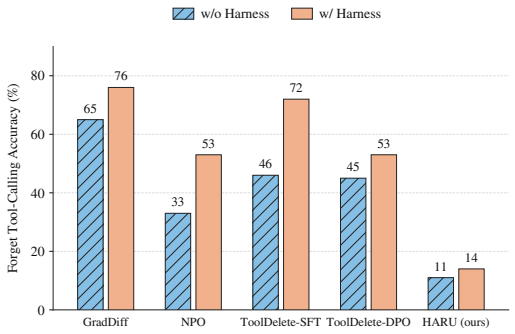

Figure 1: Forget tool-calling accuracy on ToolAlpaca without and with harness assistance from ToolAlpaca-13B. We compare GradDiff, NPO, ToolDelete-SFT, ToolDelete-DPO, and HARU. Lower accuracy indicates stronger suppression of targeted tool use.

To characterize this vulnerability, we evaluate two representative forms of harness assistance: reasoning guidance before generation and schema repair after generation. Our analysis identifies two corresponding recovery mechanisms, illustrated in **Figure 2**. Reasoning guidance elicits suppressed tool calls by modifying the model's intermediate context, while schema repair reconstructs executable calls from malformed outputs using the existing tool registry without additional model inference. These findings motivate evaluating tool unlearning at the level of the complete model–harness system. We formalize this requirement and introduce the *Harness Recovery Rate* (HRR), which measures the fraction of

forget-set examples that are unsuccessful under the default interface but become successful with harness assistance.

To address this vulnerability, we propose *Harness-Aware Robust Unlearning* (HARU), a framework that explicitly accounts for harness assistance during unlearning. HARU augments the unlearning objective to suppress behaviors that remain recoverable through the harness while preserving retained-tool capabilities. HARU reduces HRR from 32.8% to 4.5% under reasoning assistance on ToolAlpaca and from 80.0% to 1.5% under schema repair on BFCL. Under these harnesses, HARU achieves retained-tool accuracies of 69.0% and 71.2%, respectively. These results demonstrate that incorporating harness behavior into unlearning improves robustness to the evaluated forms of assistance while preserving substantial retained-tool utility.

Our contributions are threefold:

- We identify harness-mediated recovery as a failure mode of tool unlearning, showing that harness assistance can restore apparently suppressed tool-use capabilities without modifying model parameters.
- We characterize recovery through reasoning assistance and schema repair, formulate tool-unlearning robustness at the level of the model–harness system, and introduce HRR to quantify this vulnerability.
- We propose HARU, which accounts for harness-recoverable behaviors during unlearning, and demonstrate an improved balance between target-tool suppression, harness robustness, and retained-tool utility.

2 Why Conventional Tool Unlearning Fails in Harnessed Agents

2.1 Preliminaries

We consider a tool-using agent parameterized by θ that receives an instruction x, has access to a tool set $\mathcal{T}$, and generates a response that could call the tool a. Tool unlearning updates the agent's parameters to remove its ability to invoke a target subset $\mathcal{T}_f \subset \mathcal{T}$ while preserving performance on the remaining tools $\mathcal{T}_r = \mathcal{T} \setminus \mathcal{T}_f$ and general tasks Cheng & Amiri (2025). Conventional formulations typically assume a fixed prompt format, tool specification, and expected output format during optimization and evaluation. More precisely, the unlearning objective is written as

$$\min_{\theta} \ \mathbb{E}_{(x,a)\sim D_f}[\ell_f(\theta; x, a)] + \lambda \mathbb{E}_{(x,r)\sim D_r}[\ell_r(\theta; x, a)]\,, \tag{1}$$

where D_f denotes the forget set of instruction–action pairs (x, a), each demonstrating a target tool action a that should become inaccessible after unlearning, and D_r denotes the retain set of pairs representing behaviors to preserve. The functions ℓ_f and ℓ_r denote the forget and retain losses, respectively, and λ controls the relative balance between these two tasks. This objective targets reference responses under a **fixed** tool-use setup. However, a deployment harness can modify the input context or transform responses before execution. Consequently, suppression under this setup does not establish that the corresponding tool unlearning is robust in real-world deployment.

2.2 Tool-Calling Leakage in Pre- and Post-Harness Settings

We examine the robustness of conventional tool unlearning under the two harnesses illustrated in **Figure 2**. Both harnesses recover tool-use behaviors that appear to have been unlearned under the default setup.

Example 1: Tool Leakage via Harness-Injected Thoughts. We first examine a ToolAlpaca-7B actor Tang et al. (2023) unlearned using the Negative Preference Optimization (NPO) algorithm Zhang et al. (2024). As shown in **Figure 3(a)**, the actor generates both its thought and its action under default inference. Although the query asks for COVID-19 deaths in India, its thought shifts toward global statistics, leading to `getGlobalStats` instead of the target tool `getCountryStats`. A reasoning harness uses a frozen LLM planner to supply a country-specific thought without naming any tool. Given this thought, the same unlearned actor generates the target call with the correct country argument. Hence, suppression along the actor's native reasoning trajectory leaves its ability to map task-relevant reasoning to the undesired action accessible.

(a) Reasoning assistance

ToolAlpaca-7B; NPO on ToolAlpaca.

Query: What is the total number of deaths in India due to COVID-19?

Forget target: `getCountryStats`

Without harness

Agent reasoning: *"I need to find the real-time global statistics on COVID-19 cases and deaths."*

Tool call: `getGlobalStats({})`

Mismatch: the agent issues a call to a tool unsuited to the India-specific query.

With reasoning harness

Supplied reasoning: *"I need to find the total number of deaths in India."*

Tool call:
```
getCountryStats(
 {"countryName": "India"})
```

Target recovered!

(b) Schema repair

xLAM-1B; NPO on BFCL.

Query: Compute the compound annual growth rate from 2000 to 3000 over 4 years.

Forget target: `calculate_cagr`

Without harness

Output:
```
{"initial_value": 2000,
 "final_value": 3000,
 "period_in_years": 4}
```

Call rejected: missing function name and `arguments` wrapper.

With schema harness

Repair: matches argument keys to a registered schema and restores the call.

Tool call:
```
calculate_cagr(
 initial_value=2000,
 final_value=3000,
 period_in_years=4)
```

Target recovered!

Figure 3: Harness-mediated recovery after tool unlearning. Green denotes successful unlearning; red denotes target-tool invocation.

Example 2: Tool Leakage via Harness-Induced Schema. We next examine an xLAM-1B model Zhang et al. (2025) unlearned using the NPO algorithm Zhang et al. (2024) on BFCL Patil et al. (2025). For the compound annual growth rate query in **Figure 3(b)**, the model generates valid JSON containing all three correct arguments but removes the function identifier and the canonical `arguments` wrapper. The default tool-call parser rejects this response, making the target behavior appear forgotten. A deterministic schema harness matches the produced argument keys to the registered schemas, identifies `calculate_cagr`, and reconstructs the target call from the same response. Here, suppression depends on an invalid call representation. The output retains sufficient argument information for the harness to recover the target tool using the existing tool registry.

The examples above expose two critical gaps in conventional tool unlearning.

1. Evaluation gap. Evaluation under a fixed setup does not assess robustness to deployment-time harness assistance, which can recover apparently suppressed target-tool invocations.

2. Methodological gap. Conventional unlearning can induce superficial suppression by altering reasoning trajectories or disrupting tool-call structure, while leaving the underlying tool-use capability recoverable under harness assistance.

3 Harness-Robust Unlearning Formulation and Evaluation

In this section, we introduce a harness-robust formulation of tool unlearning and an evaluation metric that measures the strength of the harness in recovering behaviors that appear suppressed under the default interface.

New Formulation. As **Figure 3** demonstrates, the conventional unlearning, as formulated in (1), does not provide sufficient robustness in the harnessed-agent setting. To address this, we introduce a new formulation that explicitly considers for both pre-action and post-action harnesses, denoted by h_0 and h_1, respectively. For a harness $h = (h_0, h_1)$ and an instruction-action pair (x, a), we define

$$\Pi_{\theta,h}(a \mid x, \mathcal{T}) = \Pr_{r \sim p_\theta(\cdot \mid h_0(x,\mathcal{T}))} \left[h_1(r, x, \mathcal{T}) = a \right],$$

where $\Pi_{\theta,h}$ measures the probability that the complete model–harness system realizes action a, irrespective of the particular intermediate representation r. For a given harness $h = (h_0, h_1)$, we

propose the following harness-robust tool-unlearning objective, given as

$$\min_{\theta} \quad \mathbb{E}_{(x,a)\sim\mathcal{D}_f}\left[\ell_f\left(\Pi_{\theta,h}(a \mid x,\mathcal{T})\right)\right] + \lambda\mathbb{E}_{(x,a)\sim\mathcal{D}_r}\left[\ell_r\left(\Pi_{\theta,h}(a \mid x,\mathcal{T})\right)\right]. \tag{2}$$

This formulation introduces two key changes relative to conventional sequence-level unlearning: (1) the object of unlearning is the *executable action* rather than a particular sequential output. Multiple intermediate representations could correspond to the same action after harness processing; (2) the harness is treated as part of the deployed agent, so forget and retain tasks are evaluated after both pre-generation and post-generation transformations. Consequently, suppressing the probability of a canonical target sequence is insufficient if another model output remains mapped by the harness to the same undesired action.

Evaluation Metric. The formulation (2) motivates evaluating unlearning at the level of the deployed model–harness system rather than only at the model output. Therefore, we evaluate each unlearned model under both the default interface (no harness) and an assisted harness h on the same instructions, while holding model parameters, tool registries, and decoding settings fixed. For both benchmarks, we use the *accuracy* metric, the official function-call checker. It measures the fraction of trajectories with the correct tool selection, arguments, and call sequence. Higher retain performance and lower forget-set performance are desirable after unlearning.

Accuracy provides an aggregate measure of performance but does not reveal whether the harness recovers behaviors that appear suppressed under the default interface. Therefore, we define the *Harness Recovery Rate* (HRR). Let $s(x,t) \in 0,1$ denote the benchmark correctness indicator for instruction x and predicted tool call t, where $s(x,t)=1$ if t matches the corresponding ground-truth tool call. Let $t_0(x)$ denote the model prediction without the harness and $t_1(x)$ the prediction after applying the harness. We define

$$\mathrm{HRR}(h) = \frac{\sum_{x\in D_f} \mathbf{1}\{s(x,t_0(x))=0,\ s(x,t_1(x))=1\}}{\sum_{x\in D_f} \mathbf{1}\{s(x,t_0(x))=0\}},$$

measuring the fraction of forget examples that are incorrect under the default interface but become correct after harness intervention.

4 HARU: Harness-Aware Robust Unlearning

In this section, we propose **Harness-Aware Robust Unlearning (HARU)**, a general procedure for mitigating tool-use leakage under harnessed setup. HARU approximates the action-level objective in (2) by converting behaviors recovered through the harness into additional training set. We expose an initially unlearned model to the same harness used at deployment, identify responses that still realize an undesired action, and explicitly suppress these harness-recoverable behaviors during refinement.

(1) Initial Unlearning. We start from the pre-trained model θ_0 and optimize the conventional unlearning objective in (1) for t_0 epochs, obtaining an initially unlearned model θ_1. We set $\theta(1) := \theta_1$ and then perform t_1 rounds of harness-aware unlearning.

(2)-a Harness-Guided Data Construction. At each epoch t, we freeze the current model $\theta(t)$ and expose the forget instructions to the deployment harness. We retain model responses whose harness-processed execution still realizes an unwanted action. These responses are treated as *hard negatives*. Although, their surface forms could differ from the original forget targets, the complete model–harness system maps them to the same undesired behavior. We augment the original forget data with the hard negatives.

(2)-b Harness-Aware Unlearning. We update the model using a harness-aware unlearning objective while preserving the original retain set resulting the updated model $\theta(t+1)$.

The specific construction of the hard negatives depends on where the harness acts in the model–tool pipeline. We next instantiate HARU for pre- and post-processing harnesses.

Pre-processing Harness. For each forget example $(x, a) \in \mathcal{D}_f$, we sample K outputs from the frozen prep-rocessing harness. Each output induces a modified model context $c^{(k)}$, from which the frozen model generates a response $c^{(k)} = h_0^{(k)}(x, \mathcal{T})$ and $r^{(k)} \sim p_{\theta(t)}\big(\cdot \mid x, c^{(k)}\big)$ for every $k = 1, \ldots, K$. We keep $(x, c^{(k)}, a)$ if the resulting execution $r^{(k)}$ matches with the ground-truth action a. Combining the resulting hard negatives with the original forget examples produces an augmented forget set $\widehat{D}_f$.

We approximate the harness-robust objective in (2) by replacing the original forget distribution with the augmented distribution:

$$\min_{\theta} \mathbb{E}_{(x,c,a)\sim\widehat{D}_f} \left[\ell_f(\theta; x, c, a)\right] + \lambda \mathbb{E}_{(x,a)\sim D_r} \left[\ell_r(\theta; x, a)\right].$$

Only model-response tokens contribute to the forget loss. Thus, HARU suppresses forgotten actions even when they are extracted under alternative contexts supplied by the pre-processing harness.

Post-processing Harness. In tool-use settings, the model could move toward another output realization that still calls the undesired capability. As in our experiment we use the schema harness, for each forget sample $(x, a) \in D_f$, we introduce an explicit non-executable destination $y_s = \texttt{\{"tool_calls":[]\}}$. Given the original forget set D_f, we construct the corresponding safe-target set $\widehat{D}_f = \{(x, y_s) : (x, a_f) \in D_f\}$.

During the refinement stage, each forget instruction is therefore optimized both away from its original tool-use completion and toward the explicit no-tool response. At each epoch, we augment the conventional forget–retain objective with the safe-destination term, given as

$$\min_{\theta} \mathbb{E}_{(x,a)\sim D_f} \left[\ell_f(\theta; x, s)\right] + \gamma \mathbb{E}_{(x,y_s)\sim\widehat{D}_f} \left[\ell_s(\theta; x, y_s)\right] + \lambda \mathbb{E}_{(x,a)\sim D_r} \left[\ell_r(\theta; x, a)\right], \tag{3}$$

where $\ell_s(\cdot)$ denotes the safe-destination loss that encourages the model to generate the non-executable response y_s and γ controls the strength of the safe-destination objective.

5 Experimental Results

5.1 Experimental Setup

Benchmarks. We use ToolAlpaca Tang et al. (2023) for trajectory-based tool-use experiments. For direct function calling, we construct the unlearning training data from the Salesforce `xlam-function-calling-60k` corpus Zhang et al. (2025) and evaluate the resulting models on the Berkeley Function Calling Leaderboard (BFCL-v4; Patil et al. (2025)). We construct the unlearning datasets by partitioning APIs into forget and retain groups and assigning train and evaluation examples according to their APIs.

For ToolAlpaca, we select the forget APIs at random and hold out examples from both groups for evaluation.

For xLAM/BFCL, we designate 18 APIs as target capabilities to be removed (as provided in **Table 8**) and construct the training data from a separate pool of $60,000$ tool-calling examples. Among these, $3,632$ examples invoke at least one target API; $1,612$ of them also invoke a non-target API and are excluded as mixed examples. This leaves $2,020$ *pure-forget* examples containing only target APIs. The remaining $56,368$ examples contain no target API and form the candidate retain set, from which we randomly sample $4,040$ examples for training. Across the corpus, $3,179$ unique APIs are invoked, of which 18 are assigned to the forget set and $3,161$ to the retain set. Evaluation covers four BFCL-v4 categories: `simple_python`, `multiple`, `parallel`, and `parallel_multiple`. The primary forget evaluation set contains only target-capability examples correctly solved by the original xLAM model, while the retain evaluation set contains examples that do not call any target API. **Table 1** summarizes the resulting splits. Construction details are provided in **Appendices A.1** and **A.2**.

Deployment Harnesses. On ToolAlpaca, we consider a preprocessing harness that provides reasoning and planning guidance before tool-call generation. The standalone actor follows ReAct Yao et al. (2023), generating its own reasoning and tool calls throughout the interaction. With harness

Benchmark	APIs		Training		Evaluation	
	Forget	Retain	Forget	Retain	Forget	Retain
ToolAlpaca	82	325	635	2,462	100	100
xLAM/BFCL	18	3,161	2,020	4,040	112	867

Table 1: Summary of API counts and training and evaluation set sizes for the forget and retain partitions of each benchmark.

assistance, we replace the actor's reasoning with reasoning and planning generated by a stronger frozen model, while the final tool action is issued by the actor. On xLAM/BFCL, we use a deterministic schema-based repair. The harness does not identity of the undesired target tool. It has access only to the user query, the model-generated output, and the set of registered tool schemas available to the agent. When the native model output is already a valid tool call, the harness leaves it unchanged. Otherwise, it matches the generated argument structure against the registered schemas, uses query–schema compatibility to resolve ambiguous matches, removes unsupported arguments, and reconstructs a valid executable tool call.

Models and Baselines. For ToolAlpaca, we use the fine-tuned model ToolAlpaca-7B Tang et al. (2023) as the actor to be unlearned. For xLAM/BFCL, we use the fine-tuned model `Salesforce/xLAM-1b-fc-r` Zhang et al. (2025). We exploit representative unlearning baselines, including GradDiff Maini et al. (2024), NPO Zhang et al. (2024), and the SFT and DPO variants of ToolDelete Cheng & Amiri (2025). The *Original* actors before unlearning serve as references. For the ToolAlpaca preprocessing harness, we use the frozen ToolAlpaca-13B model to provide reasoning and planning guidance. Detailed baseline implementations and training hyperparameters are provided in **Appendix A**.

5.2 Unlearning performance evaluation

Tables 2 and 4 compare forget and retain tool-call accuracy under the default interface and with harness assistance. We assess robustness using HRR and the assisted forget accuracy, while retain accuracy measures the utility preserved after unlearning.

Method	Unlearning Effectiveness		Harness Robustness		HRR ↓	
	Forget ↓	Retain ↑	Forget ↓	Retain ↑	Rate	Counts
Original	75.0	70.0	85.0	77.0	–	–
GradDiff	65.0	**68.0**	76.0	**80.0**	40.0	14/35
NPO	33.0	53.0	53.0	62.0	32.8	22/67
ToolDelete-SFT	46.0	60.0	72.0	67.0	57.4	31/54
ToolDelete-DPO	45.0	55.0	53.0	64.0	18.2	10/55
HARU	**11.0**	59.0	**14.0**	69.0	**4.5**	4/89

Table 2: Unlearning performance comparison on ToolAlpaca with no harness and with reasoning provided by ToolAlpaca-13B harness. Scores and HRR are percentages; counts denote recovered examples over no-harness failures.

Reasoning assistance recovers forgotten tool use. On ToolAlpaca, assistance increases forget accuracy for all four baselines, with HRR ranging from 18.2% to 57.4%. For example, NPO forget accuracy increases from 33.0% to 53.0%, while retain accuracy improves from 53.0% to 62.0%. The same reasoning assistance that improves retained tool use therefore also restores behavior suppressed under the default interface.

HARU maintains stronger suppression while preserving competitive utility on ToolAlpaca. Under assisted conditions, HARU's forget accuracy increases only marginally, from 11.0% to 14.0%, while its HRR remains at 4.5%, substantially lower than the 32.8% observed for NPO. These

results indicate that HARU is more robust than the other unlearning baselines, maintaining effective suppression of the target tool even in the presence of external assistance. On the retain tasks, HARU preserves comparable general capabilities. Although GradDiff achieves slightly stronger retain performance, HARU delivers substantially better unlearning effectiveness. Overall, these results demonstrate that HARU achieves a more favorable trade-off between target-tool suppression and utility preservation, providing strong and persistent suppression while retaining useful capabilities on non-target tasks.

Stability across repeated evaluations. To assess whether HARU's advantage is due to evaluation randomness, we compare it with NPO, the strongest baseline in standalone forget suppression by multiple experiments. For each fixed checkpoint, we repeat the evaluation four times with different random seeds on the same 100 forget tasks, both without and with the frozen ToolAlpaca-13B harness. As shown in **Table 3**, HARU consistently achieves substantially lower forget accuracy than NPO across all runs, with only minor variation across seeds. This demonstrates that HARU's stronger target-tool suppression is persistent and not attributable to a particular evaluation run.

Method	Forget Accuracy ↓	
	w/o harness	w/ harness
NPO	33.00 ± 1.41	52.75 ± 2.06
HARU	$\mathbf{10.50 \pm 0.58}$	$\mathbf{14.75 \pm 0.50}$

Table 3: Evaluation stability on ToolAlpaca forget tasks over four runs with different random seeds. We report mean ± standard deviation (%), without and with the frozen ToolAlpaca-13B reasoning harness.

Method	Unlearning Effectiveness		Harness Robustness		HRR ↓	
	Forget ↓	Retain ↑	Forget ↓	Retain ↑	Rate	Counts
Original	100.0	83.3	100.0	83.6	N/A	N/A
GradDiff	**1.8**	1.6	**1.8**	2.0	**0.0**	0/115
NPO	42.0	48.7	88.4	81.6	80.0	52/65
ToolDelete-SFT	70.5	69.0	70.5	69.2	**0.0**	0/32
ToolDelete-DPO	98.2	**81.7**	98.2	**82.0**	**0.0**	0/2
HARU	40.2	70.7	41.0	71.2	1.5	1/67

Table 4: Unlearning performance comparison on BFCL-v4 without post-processing and with schema-repair assistance. Scores and HRR are reported as percentages; counts denote recovered examples among failures without post-processing.

HARU is robust to schema-repair post-processing while balancing forgetting and retention. GradDiff, ToolDelete-SFT, ToolDelete-DPO, and HARU all exhibit strong robustness to schema repair, with HRRs of 0.0%, 0.0%, 0.0%, and 1.5%, respectively. However, the other robust methods fail to achieve a desirable forget–retain balance: ToolDelete-SFT and ToolDelete-DPO retain high forget accuracies of 70.5% and 98.2%, indicating insufficient forgetting, while GradDiff reduces forget accuracy to 1.8% at the cost of collapsing retain accuracy to 2.0%. In contrast, HARU maintains low forget accuracy (41.0%) and substantially higher retain accuracy (71.2%), while remaining nearly unaffected by schema repair. NPO achieves higher retained utility, but its forget accuracy rises sharply from 42.0% to 88.4% under repair (80.0% HRR), revealing strong recoverability. Thus, HARU uniquely combines effective forgetting, retained utility, and robustness to post-processing.

5.3 HARU Generalization across Harness

We further study whether HARU generalizes beyond the harness used during unlearning. We compare HARU (unlearned using ToolAlpaca-13B as the harness) and the strongest baseline NPO across four different harnesses: Qwen2.5-7B, Mistral-7B, Llama-3.1-8B, and Gemma-4-E2B. These models belong to different model families but are all fine-tuned on the ToolAlpaca dataset, allowing them to provide distinct reasoning trajectories.

Figure 4 compares the HRR of HARU and NPO across reasoning providers. HARU consistently achieves an HRR below 8%, whereas NPO remains above 24% under every harness setting. For example, with Gemma-4-E2B, NPO and HARU attain HRRs of 34.3% and 7.8%, respectively, while with Llama-3.1-8B the corresponding values are 24.3% and 4.4%, representing the two extremes among the evaluated harnesses. These results show that, across reasoning providers of varying capability, HARU consistently exhibits substantially lower recovery rates than NPO. We further plot the $y = x$ line to facilitate direct comparison between the two methods; all harness results, including the original ToolAlpaca-13B setting, fall below this line. Overall, the consistently lower HRR of HARU across reasoning providers indicates that its robustness generalizes beyond the ToolAlpaca-13B harness used during unlearning.

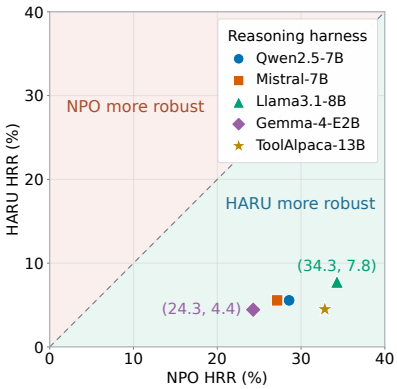

Figure 4: Generalization experiment.

5.4 Harness aibility ablation

We test whether the harness can induce tool-use capability in a model that has not undergone tool-use fine-tuning. We evaluate Vicuna-7B, the base model prior to ToolAlpaca fine-tuning, with and without the harness on the ToolAlpaca forget and retain splits. We additionally evaluate BBH-Hard Suzgun et al. (2023) and MMLU Hendrycks et al. (2021) to assess whether the harness affects general model capability in the absence of tool-use training.

As shown in **Table 5**, the harness improves general-task performance, increasing BBH-Hard accuracy from 32% to 36% and MMLU accuracy from 34% to 44%. In contrast, tool-use accuracy remains at 0–1% on both ToolAlpaca splits, with no measurable improvement from the harness. This indicates that the harness can enhance general model capability but does not itself confer effective tool-use ability on a model without prior tool-use knowledge. The results therefore suggest that tool-use knowledge is still retained in the unlearned agent, and that ordinary harness helps elicit this remaining capability.

Evaluation	**Vicuna-7B**	
	w/o harness	w/ harness
Forget	1/100	0/100
Retain	1/100	1/100
BBH-Hard	32/100	36/100
MMLU	34/100	44/100

Table 5: Ablation study on Vicuna-7B performance with and without harness assistance.

6 Conclusion

We demonstrate that ordinary harness assistance can undermine unlearning by restoring apparently suppressed tool-use capabilities. Tool-unlearning experiments on ToolAlpaca and BFCL show that reasoning guidance before generation and schema repair after generation can recover target-tool use without changing model parameters. We formalize robustness at the level of the complete model–harness system, introduce HRR to quantify harness-mediated recovery, and propose HARU to address this vulnerability. Across both benchmarks, HARU strengthens robustness to harness assistance and achieves strong forgetting while preserving substantial retained-tool utility.

7 Broader Implications

Unlearning robustness and, more broadly, agent security remain insufficiently studied under harness assistance. Our findings expose a critical deployment gap: helpful harness can weaken unlearning implemented through model updates. Effective forgetting therefore requires robustness to the surrounding harness. These findings suggest that aligning the model alone is insufficient to ensure agent security, because deployed behavior also depends on the harness and its interactions with the environment. More generally, security assessments must account for how harnesses and environments affect the persistence of model-level protections in deployed agents. A key direction for future security research is therefore to develop system-level approaches, like HARU that jointly consider model alignment, harness design, tool access, and the execution environment.

AI USE STATEMENT

We used generative AI tools to polish the language and presentation of the manuscript. In our experiments, we also used LLMs as API simulators and as judges to evaluate agent outputs. The authors are responsible for reviewing AI-assisted content and verifying the experimental findings, and take full responsibility for the final manuscript, including all claims and artifacts produced with the aid of generative AI.

REFERENCES

Yajing Bai, Jinhao Duan, Jie Peng, Xianfeng Wu, Sijia Liu, Song Wang, and Tianlong Chen. HarnessRisk: A lifecycle-oriented benchmark for agent harness safety. *arXiv preprint arXiv:2608.17597*, 2026.

Baicheng Chen, Zheyuan Liu, Jingyu Zhang, Kaize Ding, Ningshan Ma, Yue Huang, and Meng Jiang. Forgotten in weights, recovered by tools: Agentic tool unlearning for LLM agents. *arXiv preprint arXiv:2608.21544*, 2026.

Jiali Cheng and Hadi Amiri. Tool unlearning for tool-augmented LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 9947–9963. PMLR, 2025.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-Act: Improving planning of agents for long-horizon tasks. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 15419–15462. PMLR, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

Joel Jang, Dongkeun Yoon, Sohee Yang, Sungmin Cha, Moontae Lee, Lajanugen Logeswaran, and Minjoon Seo. Knowledge unlearning for mitigating privacy risks in language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14389–14408. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.805.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 24370–24391. PMLR, 2024.

LangChain. LangChain Core: parse_partial_json. Software documentation, 2026. Python API reference, accessed September 26, 2026.

Sydney Lewis. Same model, different harness: Different coding-agent results. *arXiv preprint arXiv:2608.26218*, 2026.

Jakub Łucki, Boyi Wei, Yangsibo Huang, Peter Henderson, Florian Tramèr, and Javier Rando. An adversarial perspective on machine unlearning for AI safety. *Transactions on Machine Learning Research*, 2025.

Aengus Lynch, Phillip Guo, Aidan Ewart, Stephen Casper, and Dylan Hadfield-Menell. Eight methods to evaluate robust unlearning in LLMs. *arXiv preprint arXiv:2402.16835*, 2024.

Pratyush Maini, Zhili Feng, Avi Schwarzschild, Zachary C. Lipton, and J. Zico Kolter. TOFU: A task of fictitious unlearning for LLMs. In *Proceedings of the First Conference on Language Modeling*, 2024.

Qianyu Meng, Yanan Wang, Liyi Chen, Yihang Li, Wei Wu, Wenyuan Jiang, Qimeng Wang, Chengqiang Lu, Yan Gao, Yi Wu, and Yao Hu. Agent harness for large language model agents: A survey. Preprints.org, 2026. Preprint, version 3.

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

Weijia Shi, Jaechan Lee, Yangsibo Huang, Sadhika Malladi, Jieyu Zhao, Ari Holtzman, Daogao Liu, Luke Zettlemoyer, Noah A. Smith, and Chiyuan Zhang. MUSE: Machine unlearning six-way evaluation for language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-acl.824.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. ToolAlpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*, 2023.

Vercel. AI SDK Core: Tool calling. Software documentation, 2026. Tool call repair documentation, accessed September 26, 2026.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *Proceedings of the First Conference on Language Modeling*, 2024.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Yilun Yao, Xinyu Tan, Chao-Hsuan Liu, Yaoming Li, Zhengyang Wang, Wenhan Yu, Zhewen Tan, Yuxuan Tian, Guangxiang Zhao, Lin Sun, Xiangzheng Zhang, and Tong Yang. HarnessBench: Measuring harness effects across models in realistic agent workflows. *arXiv preprint arXiv:2605.27922*, 2026.

Yuanshun Yao, Xiaojun Xu, and Yang Liu. Large language model unlearning. In *Advances in Neural Information Processing Systems*, volume 37, pp. 105425–105475. Curran Associates, Inc., 2024. doi: 10.52202/079017-3346.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Zhiwei Liu, Yihao Feng, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. 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)*, pp. 11583–11597. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.578.

Ruiqi Zhang, Licong Lin, Yu Bai, and Song Mei. Negative preference optimization: From catastrophic collapse to effective unlearning. In *Proceedings of the First Conference on Language Modeling*, 2024.

A Benchmark Construction and Implementation Details

We detail the construction of the unlearning datasets summarized in **Table 1** and the settings for the evaluated checkpoints. Evaluation metrics are defined in Section 3.

A.1 ToolAlpaca

Model and Harness. The actor is `TangQiaoYu/ToolAlpaca-7B`, and the frozen reasoning provider is `TangQiaoYu/ToolAlpaca-13B`, both fine-tuned from Vicuna Tang et al. (2023). The default actor generates both Thought and Action. With assistance, the provider supplies only the Thought and the actor generates every executable call. The provider sees the task, tool descriptions, and interaction history, but no reference answers or forget-set labels.

Target APIs and Training Construction. After removing entries without function descriptions, the ToolAlpaca training corpus contains 447 distinct API names. Using seed 42, we randomly assign 89 APIs (20%) to forgetting and 358 to retention; all examples sharing an API name receive the same assignment. We remove trajectories with empty queries, no tool calls, more than five tool-call steps, or calls exclusively to `getDetails`. Within each API entry, we shuffle examples using the API name and seed 42 and hold out approximately 25% for evaluation, keeping at least one example in each split when possible; singletons remain in training. This yields 635 forget and 2,462 retain training trajectories spanning 82 and 325 APIs, respectively (**Table 1**). The other designated APIs have no usable training examples.

Evaluation Construction. The held-out pools contain 173 forget and 667 retain tasks. For each pool, we shuffle API entries with seed 42 and select exactly 100 tasks, truncating the final entry as needed. These evaluation sets are disjoint from training and fixed across methods and harnesses. Routing suffixes distinguish duplicate API names with different schemas without changing their forget/retain assignment.

HARU Augmentation. The 13B generator proposes ten trajectories per forget instruction. Of 6,350 proposals, 5,680 have judge verdicts. We keep trajectories judged correct in both process and response, remove duplicates, and exclude unjudged proposals. The augmented set contains 3,794 trajectories: 635 originals and 3,159 additions.

Unlearning and GPU Setup. All methods update the full actor on eight NVIDIA A100-SXM4 GPUs (40 GB each), using BF16, DeepSpeed ZeRO-3, and gradient checkpointing. **Table 6** gives shared hyperparameters; **Table 7** specifies each objective. Each update uses 64 forget examples and, when applicable, 64 uniformly sampled retain examples (preference pairs for DPO). Only assistant tokens receive loss; prompt, tool-observation, and padding tokens are masked. NPO and DPO use cached log-likelihoods from the frozen original actor as their reference.

ToolDelete uses `lmsys/vicuna-7b-v1.1` as the tool-free teacher, with greedy decoding and a 256-token limit. Both variants omit the subsequent general-capability adjustment of Cheng & Amiri (2025). The reported HARU checkpoint uses NPO on the fixed augmented set, initialized from the original actor, with no separate retain stream. Its 295 updates versus 50 for each baseline mean that the comparison does not isolate augmentation from additional optimization.

Setting	Value
Optimizer	Adafactor
Learning rate / schedule	10^{-5} / constant
Weight decay / warmup steps	0 / 0
Maximum gradient norm	1.0
Epochs / seed	5 / 42
Batch per GPU / accumulation steps	1 / 8
Effective batch size per stream	64
Context limit	2,048 tokens

Table 6: Shared ToolAlpaca unlearning hyperparameters for all five methods.

Method	Objective	Coefficients
GradDiff	Forget gradient ascent + retain cross-entropy	Weights 1
NPO	Forget NPO + retain cross-entropy Shi et al. (2025)	$\beta = 0.1$, weights 1
ToolDelete-SFT	Teacher targets on forget; original targets on retain	Weights 1
ToolDelete-DPO	Prefer teacher to original on forget; reverse on retain	$\beta = 0.1$, weights 1
HARU	NPO on 3,794 augmented forget trajectories	$\beta = 0.1$, retain weight 1

Table 7: ToolAlpaca method-specific settings. Weights denote forget/retain loss weights. All methods use five epochs; baselines use 635 original forget trajectories.

Inference and Evaluation. Actor and provider inference use FP16, greedy decoding, and a 256-token limit per generation. Following ToolAlpaca Tang et al. (2023), we use `gpt-3.5-turbo` for API simulation (temperature 0.5) and `gpt-4-0613` for judging (temperature 0.2), with the original `Simulator.txt` and `Evaluation.txt` prompts. Only the judge receives the reference solution. Process correctness is the primary metric; empty trajectories are incorrect and infrastructure failures are rerun. Tool responses are simulated rather than executed on live APIs.

Vicuna-7B Control. The actor-capability ablation in Section 5.4 evaluates Vicuna-7B before tool-use fine-tuning with and without harness assistance. It uses the fixed 100-example forget and retain splits and 100 examples each from BBH-Hard and MMLU. Results are reported as correct predictions out of 100 for each evaluation set.

A.2 xLAM/BFCL

Model and Harness. The actor is `Salesforce/xLAM-1b-fc-r` Zhang et al. (2025), a model fine-tuned for structured function calling and evaluated on BFCL. The deterministic harness leaves valid calls unchanged and repairs malformed calls using argument keys, query–schema compatibility, and the registered tool schemas. It processes the same raw response without an additional model call.

Target APIs and Training Construction. We assign the 18 APIs provided in **Table 8** to the forget group and all remaining APIs to the retain group. Unlearning training uses the Salesforce `xlam-function-calling-60k` corpus of $60,000$ tool-calling examples. Of these, $3,632$ contain at least one target API, including $1,612$ that also invoke a non-target API. Excluding these mixed trajectories yields $2,020$ pure-forget examples, whose reference calls invoke only target APIs. The $56,368$ examples with no target API form the candidate retain pool, from which we randomly sample $4,040$ examples.

Forget APIs		
`light_travel_time`	`least_common_multiple`	`greatest_common_divisor`
`cagr`	`compound_interest`	`solve_quadratic`
`predict_evolution_rate`	`future_investment_value`	`circle_area`
`calculate_quarterly_dividend`	`displacement`	`calculate_electric_field`
`neuronal_activity_rate`	`triangle_area`	`density`
`structural_analysis`	`cosine_similarity`	`final_velocity`

Table 8: Target forget APIs used in the xLAM/BFCL unlearning setup.

Evaluation Construction. Evaluation covers four BFCL-v4 categories: `simple_python`, `multiple`, `parallel`, and `parallel_multiple`, comprising $1,000$ source examples. The primary forget set contains 112 examples whose reference calls involve a target API and for which the original xLAM-1B model produces a BFCL-valid complete reference tool call under the native evaluation interface. The retain set contains 867 examples whose reference calls invoke no target API. We keep these evaluation splits fixed across all methods and use them for the main results reported in **Table 4**.

Training. All five methods use a learning rate of 10^{-5}, weight decay 0.01, a 3,072-token context limit, seed 42, and 32 gradient-accumulation steps, giving effective batches of 32 forget and

64 retain examples (pairs for DPO). Forget and retain loss weights are both 1. GradDiff and both ToolDelete variants use a 10% warmup ratio and gradient clipping at 1.0; HARU also clips gradients at 1.0. **Table 9** summarizes the remaining method-specific settings. ToolDelete uses `deepseek-ai/deepseek-coder-1.3b-instruct` as the tool-free teacher, matching the base model of xLAM-1B. Both ToolDelete variants disable task arithmetic ($\alpha = 0$), since positive values degraded performance in this adaptation. HARU adds cross-entropy toward the no-tool response `{"tool_calls":[]}` during refinement.

Method	Epochs	Objective and coefficients
GradDiff	1	Forget gradient ascent + retain cross-entropy
NPO	1	Forget NPO + retain cross-entropy, $\beta = 0.1$, $\lambda = 1$
ToolDelete-SFT	1	Teacher targets on forget; original targets on retain; $\alpha = 0$
ToolDelete-DPO	3	Teacher/original preference pairs; $\beta = 0.1$, $\alpha = 0$
HARU	$1 + 2$	One NPO epoch, then two with the safe-target loss; $\beta = 0.1$, $\lambda = 1$, $\gamma = 0.5$

Table 9: xLAM/BFCL unlearning settings. λ weights retention, γ weights the safe-target loss, and α controls ToolDelete task arithmetic.

Inference and Evaluation. Outputs are scored by the official BFCL function-call checker. There is no LLM simulator or LLM judge. Default and repaired predictions are paired on the same examples to compute HRR. Checker correctness is distinct from successful live tool execution.

B Related Work

B.1 LLM and Tool Unlearning

LLM unlearning aims to remove selected knowledge or behaviors from a trained model while preserving its remaining capabilities. Established approaches modify model parameters to suppress outputs associated with the forget set. Gradient ascent reduces the likelihood of target sequences Jang et al. (2023), while other objectives suppress unwanted generations and limit degradation on retained data Yao et al. (2024). Negative Preference Optimization (NPO) suppresses target responses through an objective that compares their likelihoods with those assigned by a frozen reference model Zhang et al. (2024). Benchmarks such as TOFU and MUSE evaluate both forgetting and retained utility, with MUSE additionally examining privacy leakage, scalability, and repeated unlearning Maini et al. (2024); Shi et al. (2025). These studies provide the objectives and evaluation principles underlying our unlearning baselines.

Tool unlearning applies selective forgetting to the ability to invoke designated tools. Cheng & Amiri (2025) formalize this problem and propose ToolDelete, which combines tool-knowledge deletion, retained-tool preservation, and general-capability retention. Their evaluation also uses diverse prompts to probe remaining tool knowledge. Our work examines robustness to changes in the deployment harness: whether a fixed unlearned actor can recover target-tool use when assisted by external reasoning or output repair. HARU incorporates examples of these recoverable behaviors into subsequent unlearning.

B.2 Agent Harnesses

An agent harness manages the context, interactions, and execution interfaces surrounding a language model. ReAct interleaves reasoning, actions, and observations Yao et al. (2023); AutoGen coordinates communication and tool execution among agents Wu et al. (2024); and Reflexion uses feedback and stored reflections to guide subsequent attempts Shinn et al. (2023). These systems illustrate how surrounding software can influence an agent's behavior through both the information supplied to the model and the processing of its outputs.

Planning assistance supplies intermediate guidance before an actor generates an action. LLMCompiler separates function-call planning from scheduling and execution Kim et al. (2024), while Plan-and-Act uses a planner to produce high-level plans and an executor to translate them into environment actions Erdogan et al. (2025). Our preprocessing harness follows this separation: a stronger

model provides reasoning and planning, while the unlearned actor selects the tool and generates its arguments.

Output processing supports execution through parsing, validation, and repair. For example, LangChain provides partial-JSON parsing LangChain (2026), and the Vercel AI SDK supports repair of invalid tool calls Vercel (2026). Our postprocessing harness implements deterministic schema-based repair using the model output, user query, and registered tool schemas. Together, reasoning assistance and output repair allow us to examine recovery at two stages of agent execution while keeping the actor's parameters fixed.

B.3 Robustness and Recovery after Unlearning

Robustness evaluations examine whether suppressed information or capabilities remain accessible under alternative inference conditions. Lynch et al. (2024) identify limitations of narrow unlearning evaluations, and Łucki et al. (2025) show that adapted jailbreak prompts can recover capabilities suppressed by unlearning. Such adversarial prompting deliberately modifies the model's input to elicit forgotten behavior. Our work examines recovery caused by assistance designed to improve ordinary task execution. Although our reasoning harness also changes the actor's context, it supplies task-level guidance without prompts optimized to circumvent unlearning; the actor remains responsible for generating the tool call. Our postprocessing harness applies fixed schema-repair rules to the actor's output. We therefore study whether useful, non-adversarial assistance can unintentionally restore suppressed tool-use capabilities while model parameters remain fixed.

Recent concurrent work, Agentic Tool Unlearning (ATU; Chen et al. (2026)), studies recovery of forgotten knowledge through web search, retrieval, and local databases. ATU combines parametric unlearning with trajectory-level reinforcement learning to discourage retrieval of target information and answer leakage. Like our work, it shows that assistance outside the model can undermine parameter-level unlearning. Our study differs in two respects. First, ATU focuses on recovery only through information-access tools, whereas we examine harness assistance from a harness perspective, ranging from reasoning before generation to schema repair after generation. Second, retrieved observations in ATU can directly supply the forgotten content, providing an external route to recovering the answer. Our harnesses operate under more restricted access to target information: they do not retrieve external answers, and the reasoning provider receives neither reference answers nor forget-set labels. Instead, they supply reasoning hints or repair calls using the existing response, query, and tool schemas. Recovery in our setting thus shows that even ordinary assistance, without external retrieval of the forgotten content, can still restore suppressed tool-use capabilities.