UNDERSTANDING AND MITIGATING HALLUCINATION ESCAPE IN TOOL-USING LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) increasingly serve as autonomous agents that invoke external tools. However, this capability introduces tool hallucination, selecting incorrect tools or generating invalid calls. Existing mitigation methods report substantial improvements, yet we identify a previously overlooked failure mode that we term **Hallucination Escape**. These methods reduce hallucination on the tool configuration they are tuned on but increase it on other configurations, canceling out the gain. We further investigate this phenomenon and find that hallucination rises sharply when a model's intrinsic tool-use tendencies conflict with the current tool configuration, and that existing methods reinforce rather than suppress these tendencies, which in turn contributes to hallucination escape. Building on these findings, we propose **ESCAPEGUARD**, a training-free inference-time method that combines conflict-aware gating with configuration-derived attention enhancement to mitigate tool hallucination while preventing hallucination escape. Across six benchmarks on various models, ESCAPEGUARD reduces tool-selection hallucination by 9.0 pp and suppresses hallucination escape, lowering the cross-configuration mean by 23.7 pp and achieving an 89.1% net improvement in paired-query evaluation. We hope this work can encourage evaluation beyond a single tool configuration and pave the way for more reliable tool-using LLM agents.

1 INTRODUCTION

Large language models (LLMs) increasingly serve as autonomous agents that call external tools to complete tasks, but this capability introduces a tool hallucination: selecting nonexistent or irrelevant tools (*Tool-selection Hallucination*), or generating invalid arguments (*Tool-usage Hallucination*) (Xu et al., 2025; Zhang et al., 2024; Ke et al., 2026). Unlike text hallucination, which produces plausible but incorrect text, a hallucinated tool call can trigger irreversible operations on real systems, making it a more consequential reliability risk (Zhang et al., 2024). To address this problem, existing methods fall into two families. ❶ Training-based methods reshape model behavior through modified data or objectives: Relign adds hesitation-based deferral (Xu et al., 2025), Gorilla adopts retrieval-aware fine-tuning (Patil et al., 2024), and PALADIN fine-tunes on fault-injected trajectories (Vuddanti et al., 2025); ❷ Steering-based methods instead intervene on internal representations: LinSteer steers tool selection with activation-difference vectors (Wu et al., 2026), and PRISMS applies detector-gated directional interventions (Ke et al., 2026). Both families report substantial improvements (Figure 1a).

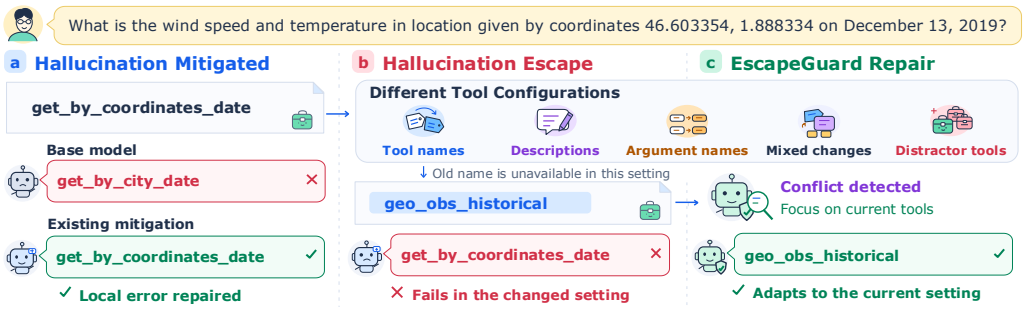

Figure 1: Hallucination escape and repair. (a) Under the standard configuration (C_1), an existing mitigation method produces the correct call. (b) After the tool is renamed (C_2), the same method still calls the original name. (c) ESCAPEGUARD adapts to the renamed configuration and produces the correct call.

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.

Phenomenon (§ 3). These improvements, however, are measured only under the tool configuration each method is tuned on and do not account for performance under other configurations. We evaluate existing methods along two dimensions: varying the tool configuration for a fixed query (Figure 2) and varying the query under a fixed tool configuration (Table B.1). This evaluation reveals a previously overlooked failure mode that we term **Hallucination Escape**, where mitigation reduces hallucination in the tuned setting but increases it in others by enough to outweigh the reduction (Figure 1b). Specifically, ❶ on the configuration side, all five methods reduce hallucination by 7.5 pp on average under their tuned configuration but collectively increase it by 13.0 pp across the remaining five, leaving every method with a higher cross-configuration mean than the base model; ❷ on the query side, $\sim$40% of repaired queries simultaneously introduce new errors on paired queries requiring a different tool. Besides, tool-usage hallucination shows limited sensitivity to all methods These raise a question: what drives hallucination escape, and why do existing methods exacerbate it?

Investigation (§ 4). To answer this, we examine whether models hold intrinsic tool-use tendencies, as the observed behavior suggests models disregard the runtime tool configuration. Concretely, we elicit model outputs without access to the tool configuration and find that models consistently favor specific tools and argument structures (ρ_{sel}=0.65), with hallucination rising sharply when these preferences conflict with the current configuration. We further validate this finding internally through layerwise probes, which detect the conflict with 86.4% accuracy and predict tool hallucination with an AUROC of 0.82. Finally, we analyze existing mitigation methods and find that they reinforce these tendencies (prior strength $+0.08$ on average), with stronger reinforcement associated with larger escape ratios (E_{w} up to 2.42), explaining why they produce and even exacerbate hallucination escape.

Solution (§ 5). Building on these findings, we propose **ESCAPEGUARD**, an inference-time method that mitigates tool hallucination while preventing hallucination escape (Figure 1c). The key idea is to redirect the model's attention back to the runtime tool configuration when a tendency conflict is detected (Figure 5). Specifically, ESCAPEGUARD reuses the frozen conflict probes to identify high-conflict inputs and, only for those inputs, strengthens attention to tool names and descriptions during selection and to the selected tool's schema during argument generation. This design ties the corrective signal to the current configuration rather than a fixed tendency, allowing it to naturally adapt as tools or queries change. Across six benchmarks on various models, ESCAPEGUARD reduces tool-selection hallucination by 9.0 pp and argument-name hallucination by up to 67% relative, while effectively suppressing hallucination escape along both dimensions: the cross-configuration mean drops by 23.7 pp, and paired-query evaluation shows an 89.1% net improvement.

To summarize, our contributions are as follows: ❶ we identify and formally define hallucination escape, a previously unrecognized failure mode in which mitigation reduces hallucination in the tuned configuration but increases it in others (§ 3); ❷ we reveal the mechanism behind it: models hold intrinsic tool-use tendencies that conflict with changed configurations, and existing methods reinforce these tendencies (§ 4); ❸ we propose ESCAPEGUARD, a training-free inference-time method that mitigates tool hallucination while effectively preventing hallucination escape across models (§ 5).

2 BACKGROUND AND RELATED WORK

LLM-based Agents and Tool Use. LLMs increasingly serve as autonomous agents that invoke tools to complete tasks. Toolformer introduces self-supervised tool-call learning (Schick et al., 2023), ReAct interleaves reasoning with action (Yao et al., 2023), and ToolLLM scales to real-world APIs (Qin et al., 2024). As these capabilities expand, so does the risk of erroneous tool invocations.

Tool Hallucination and Mitigation. Tool hallucination occurs when a model's tool invocation is inconsistent with available tools or task requirements (Xu et al., 2025; Patil et al., 2024; Liu et al., 2026; Xiong et al., 2025), including **tool-selection** hallucination (invoking nonexistent or irrelevant tools) and **tool-usage** hallucination (generating undefined parameters or ungrounded values). Existing mitigations includes: ❶ Training-based methods reshape behavior through modified objectives: Relign (Xu et al., 2025) introduces hesitation-based deferral, PALADIN (Vuddanti et al., 2025) improves fault recovery through annotated trajectories, and Gorilla (Patil et al., 2024) reduces API hallucination via retrieval-aware fine-tuning; ❷ Steering-based methods instead intervene on internal activations: LinSteer (Wu et al., 2026) manipulates tool selections via activation steering, and PRISMS (Ke et al., 2026) applies detector-gated interventions for distinct failure modes. However, both evaluate only under the tuned-on tool configuration. Next, we test whether these gains generalize.

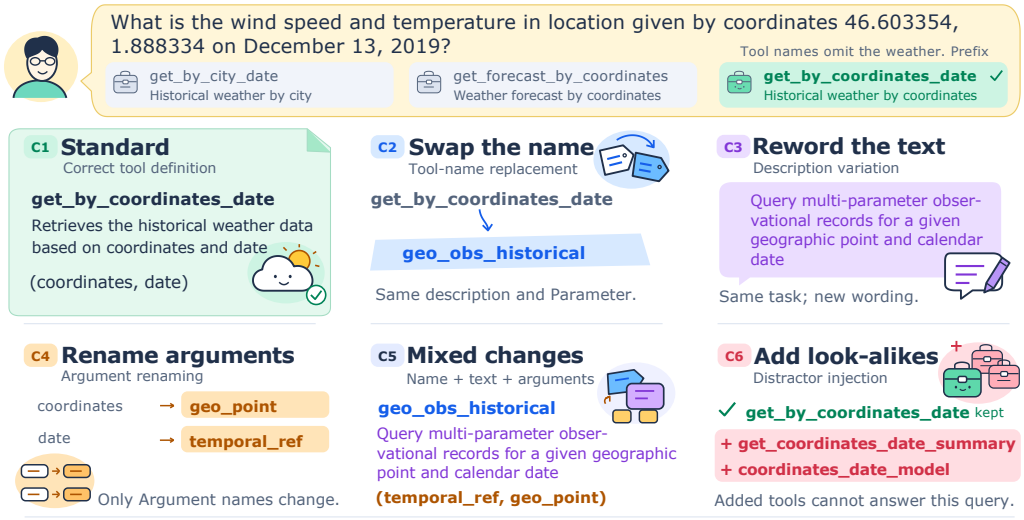

Figure 2: Tool configuration changes (C_1–C_6) illustrated with a weather API example. Each variant modifies one or more aspects of the correct tool while ensuring a valid solution remains available.

3 Phenomenon: Hallucination Escape

Existing mitigation methods report reductions, yet their evaluations keep tool definitions fixed, varying only the number, combination, and availability of tools. In practice, tools serving the same function may differ in naming and argument specifications. We evaluate them under varied tool configurations and queries, revealing a consistent failure mode and defining it as hallucination escape.

3.1 Evaluation Setup

Evaluation Protocol. The cross-configuration evaluation covers two aspects. ❶ Tool Configuration: for each query, the tool list always contains one correct tool whose definition varies systematically. We construct six configurations (Figure 2): C_1 (standard) uses original names, descriptions, and arguments; C_2 (tool replacement) substitutes a functionally equivalent tool with a different name; C_3 (description change) rewrites functional descriptions; C_4 (argument renaming) replaces argument names; C_5 (mixed) simultaneously changes the tool name, description, and argument names; C_6 (distractor injection) adds tools with similar names but mismatched functionality. All configurations were constructed using GPT-5.6-Terra, manually reviewed, and validated to preserve a valid solution (Appendix A). ❷ Query Task: under the same tool list, we construct pairs of related queries requiring different tools to test whether fixing one query introduces errors on another (Appendix B).

Methods, Models, and Data. We reproduce five mitigation methods spanning both categories. For training-based methods, we adopt Relign (Xu et al., 2025), PALADIN (Vuddanti et al., 2025), and Gorilla (Patil et al., 2024); for steering-based methods, we adopt LinSteer (Wu et al., 2026) and PRISMS (Ke et al., 2026). All use the original code and are trained or calibrated on standard-configuration (C_1) data. Experiments use LLaMA-3.1-8B-Instruct (Meta, 2024) and Qwen-3.5-9B (Qwen Team, 2026). Configuration evaluation data are sampled from BFCL V3 and Seal-Tools ($\sim$1,000 examples $\times$ 6 configurations). For query changes, 120 pairs (240 queries) are constructed from BFCL. All are repeated over three random seeds; see Appendix C for dataset selection.

Evaluation Metrics. ❶ For cross-configuration evaluation, we measure hallucination by: Tool-Selection Hallucination Rate (TSHR), the fraction of queries where the model selects an unavailable or incorrect tool; and Tool-Usage Hallucination Rate (TUHR), conditional on correct selection, decomposed into fabricated argument names (ANHR) and incorrect values (AVHR). Since $C_1 - C_3$ and C_6 preserve the argument schema, we report TSHR alone; for C_4 and C_5, which also modify arguments, we report OHR $=$ TSHR $+ (1 -$ TSHR$) \times$ TUHR. Cross-configuration mean $\bar{H}(m)$ and variance $\mathrm{Var}(m)$ aggregate these metrics. ❷ For cross-query evaluation, we report Repair Rate (fraction of errors corrected), Transfer Damage Rate (fraction of repairs introducing new errors on the paired query), and Net Improvement $=$ Repair $\times (1 -$ Damage$)$. Formal definitions in Appendix D.

Table 1: Cross-configuration hallucination rates (%). Subscripts: change from the base (pp).

Model	Method	C_1(TSHR)	C_2(TSHR)	C_3(TSHR)	C_4(OHR)	C_5(OHR)	C_6(TSHR)	$\bar{H}(m)\downarrow$	Var$(m)\downarrow$
LLaMA-3.1	Base	22.8	50.4	31.6	46.2	59.6	38.0	41.4	148.1
	Relign	$14.6_{-8.2}$	$55.2_{+4.8}$	$34.1_{+2.5}$	$44.7_{-1.5}$	$64.2_{+4.6}$	$41.3_{+3.3}$	$42.4_{+1.0}$	247.8
	PALADIN	$15.4_{-7.4}$	$54.6_{+4.2}$	$33.8_{+2.2}$	$45.1_{-1.1}$	$63.3_{+3.7}$	$40.8_{+2.8}$	$42.2_{+0.8}$	233.1
	Gorilla	$13.2_{-9.6}$	$58.9_{+8.5}$	$34.5_{+2.9}$	$44.3_{-1.9}$	$66.9_{+7.3}$	$42.3_{+4.3}$	$43.4_{+2.0}$	297.7
	LinSteer	$16.8_{-6.0}$	$54.2_{+3.8}$	$33.7_{+2.1}$	$45.9_{-0.3}$	$63.1_{+3.5}$	$40.7_{+2.7}$	$42.4_{+1.0}$	218.9
	PRISMS	$15.8_{-7.0}$	$53.3_{+2.9}$	$32.9_{+1.3}$	$45.6_{-0.6}$	$62.2_{+2.6}$	$40.1_{+2.1}$	$41.7_{+0.3}$	220.1
Qwen-3.5	Base	19.4	47.6	29.4	43.4	56.8	36.0	38.8	149.2
	Relign	$10.3_{-9.1}$	$51.0_{+3.4}$	$32.1_{+2.7}$	$42.8_{-0.6}$	$62.2_{+5.4}$	$37.9_{+1.9}$	$39.4_{+0.6}$	261.4
	PALADIN	$13.6_{-5.8}$	$52.9_{+5.3}$	$30.6_{+1.2}$	$41.9_{-1.5}$	$59.9_{+3.1}$	$39.4_{+3.4}$	$39.7_{+0.9}$	225.2
	Gorilla	$9.1_{-10.3}$	$53.7_{+6.1}$	$33.2_{+3.8}$	$42.5_{-0.9}$	$62.4_{+5.6}$	$41.3_{+5.3}$	$40.4_{+1.6}$	282.9
	LinSteer	$15.3_{-4.1}$	$52.1_{+4.5}$	$30.4_{+1.0}$	$42.7_{-0.7}$	$59.2_{+2.4}$	$39.7_{+3.7}$	$39.9_{+1.1}$	204.1
	PRISMS	$11.7_{-7.7}$	$49.5_{+1.9}$	$31.8_{+2.4}$	$43.1_{-0.3}$	$60.1_{+3.3}$	$37.4_{+1.4}$	$38.9_{+0.1}$	228.7

3.2 Evaluation Results

We evaluate hallucination rates under six tool configurations (Table 1) and across paired query tasks that share the same tool list but require different tools (Figure 3). We report two findings as follows.

Finding 1: Existing methods reduce the tuned tool-selection hallucination but worsen others. As shown in Table 1, both base models are highly sensitive to configuration changes: TSHR is 19% – 23% under C_1 but surges to 48% – 50% after tool-name replacement (C_2). Existing methods amplify this imbalance: all reduce TSHR on C_1 yet increase it by 2 – 9 pp on C_2, 1 – 6 pp on C_3 and C_6, and 2 – 7 pp in OHR on C_5, while C_4 OHR changes remain marginal (0.3 – 1.9 pp), suggesting that the usage component within OHR is largely unaffected. Overall every method with $\bar{H}$ above the base model and 62.8% higher Var on average. The same pattern holds across queries (Figure 3): methods repair target-query errors but with 37% – 50% probability simultaneously introduce new errors on paired queries, substantially limiting net improvement.

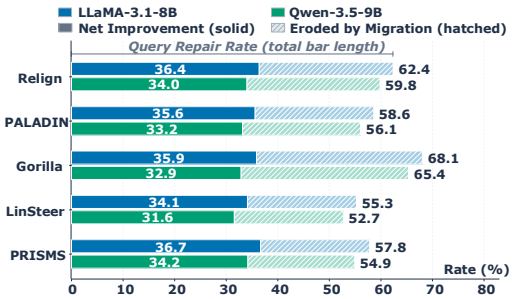

Figure 3: Cross-query hallucination escape (%).

Finding 2: Tool-usage hallucination shows limited sensitivity to all methods. To disentangle the two components of OHR, we compute TUHR on correctly selected samples across all six configurations (Appendix E). All methods change TUHR by no more than 1.5 pp under any configuration. Even though base TUHR reaches 24% – 28% on C_4 and C_5, existing methods leave this component essentially untouched. The marginal C_4 OHR reductions in Table 1 thus stem primarily from small TSHR improvements, while the dominant TUHR component remains unchanged. To control for denominator shifts caused by different correct-selection sets, Appendix E.2 repeats the analysis on the shared subset, confirming the same limited sensitivity (all changes ≤ 0.8 pp).

3.3 Definition of Hallucination Escape

The two findings above reveal a consistent pattern: existing methods shift hallucination across configurations rather than reducing it overall. We formalize this as follows.

Definition 3.1 (Hallucination Escape). Let m be a mitigation method and $\mathcal{C} = \{c_1, \ldots, c_K\}$ the configuration set. Method m exhibits hallucination escape if: (1) $\exists\, \mathcal{C}_\downarrow \subset \mathcal{C}$: $H(m, c_k) < H(\text{base}, c_k)$; (2) $\exists\, \mathcal{C}_\uparrow \subset \mathcal{C} \setminus \mathcal{C}_\downarrow$: $H(m, c_j) > H(\text{base}, c_j)$; (3) $\bar{H}(m) \geq \bar{H}(\text{base})$.

The empirical results show that all five methods satisfy this definition: they reduce hallucination on their tuned configuration yet increase it on others. The pattern manifests across both tool configurations and queries. Meanwhile, tool-usage hallucination remains largely immune to all methods, suggesting a distinct underlying mechanism. These findings raise a natural question: what drives hallucination escape, and why do existing methods exacerbate rather than suppress it?

4 INVESTIGATION: THE MECHANISM OF HALLUCINATION ESCAPE

The previous section shows existing mitigation methods reduce hallucination on their tuned configuration but increase it on others. To understand what drives this, we examine whether models hold intrinsic tool-use tendencies, as the observed behavior suggests models disregard the runtime tool configuration. We first test whether such tendencies exist and correlate with hallucination (§ 4.1), then validate this finding internally through layerwise probes (§ 4.2), and finally show how existing methods reinforce these tendencies, producing and even exacerbating hallucination escape (§ 4.3).

4.1 INTRINSIC TENDENCIES AND HALLUCINATION

In this section, we empirically investigate two aspects: whether models hold intrinsic tool-use tendencies that persist without access to the runtime configuration, and how hallucination relates to conflicts between these tendencies and the current configuration.

Identifying Intrinsic Tool-Use Tendencies. We examine what models generate without the tool configuration: if outputs are consistent across samples, the model holds a strong intrinsic tendency. We evaluate six models on 300 queries randomly selected from BFCL v3, Seal-Tools, and Glaive Function Calling V2 using closed-book elicitation (K=10 samples per query q, temperature 0.7; Appendix C.2). For tool selection, we remove the tool list and measure the prior strength as the frequency of the most commonly generated tool t: $S^{\text{sel}}_{\text{prior}}(q) = \max_t \frac{1}{K}\sum_{j=1}^{K} \mathbf{1}[t_j = t]$. For tool usage, we provide the tool name but omit its schema and compute $S^{\text{use}}_{\text{prior}}(q) = \frac{1}{K}\sum_{i=1}^{K} \text{Jaccard}(P_i, P_{\text{mode}})$, where P_i is the argument-name set in sample i and P_{mode} the most frequent set. Across models, mean selection priors range from 0.61 to 0.78 and usage priors from 0.57 to 0.71 (Table F.1), indicating that models favor specific tools and argument structures even without the runtime configuration.

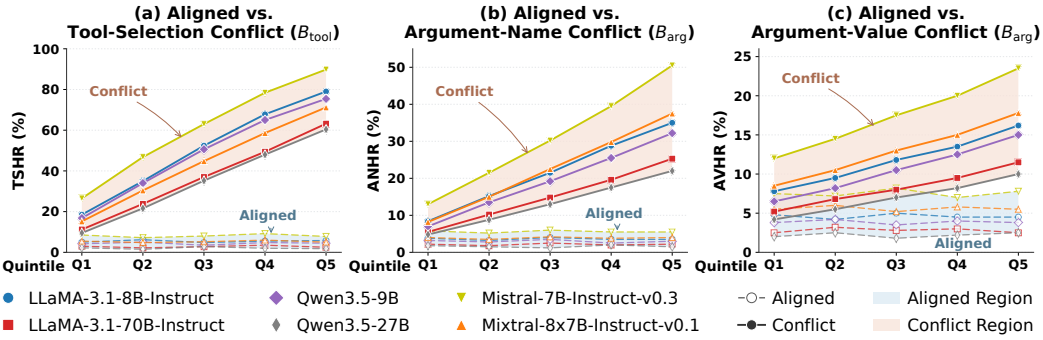

Figure 4: Hallucination rate by prior-strength quintile under aligned and conflict settings. (a) tool-selection conflict under B_{tool}; (b) argument-name and (c) value conflict under B_{arg}. Q1-Q5:weakest to strongest tendency.

Tendency Conflict and Hallucination. We next test whether these tendencies become harmful when they disagree with the current configuration. For each query we construct three matched settings: *aligned* (A), where the correct tool and schema match the elicited tendency; *tool-conflict* (B_{tool}), which replaces the preferred tool with a functionally equivalent alternative under a different name and rewritten description; and *argument-conflict* (B_{arg}), which keeps the tool name but modifies argument names or types. Both conflict settings preserve valid solutions. We compare TSHR between A and B_{tool} (Figure 4a), and TUHR between A and B_{arg} (Figure 4b,c). To examine how prior strength modulates hallucination, we rank queries by prior strength and divide them into five equal-sized quintiles ($Q_1 - Q_5$, weakest to strongest), measuring hallucination within each under both conditions. As shown in Figure 4, under alignment, hallucination remains uniformly low regardless of prior strength, whereas under conflict it increases monotonically from Q_1 to Q_5. This correlation is strongest for tool selection (ρ=0.58 – 0.72) and argument name (ρ=0.50 – 0.64), while argument value shows only a modest effect (ρ=0.30 – 0.38). Per-model breakdowns in Appendix F. These suggest that intrinsic tendencies are benign when aligned with the configuration but are associated with substantially increased hallucination when a conflict arises, with severity scaling with tendency.

The empirical experiments reveal that models exhibit stable intrinsic tendencies at both the selection and usage levels. When these tendencies align with the runtime configuration, hallucination remains uniformly low; when they conflict, hallucination rises sharply and scales with tendency. This evidence is behavioral, and we next seek further validation from the model's internal representations.

4.2 Internal Evidence of Tendency Conflict

To move beyond behavioral evidence, we probe whether the conflict between intrinsic tendencies and the runtime configuration is separable in the model's representation space and whether this internal signal is also associated with the tool hallucination.

Layerwise Conflict Probing. We probe LLaMA-3.1-8B and Qwen-3.5-9B using 1,200 queries (the 300 from § 4.1 plus 900 additional samples). We extract the hidden state $\mathbf{h}^{(l)} \in \mathbb{R}^d$ at the last input token from every decoder layer and train a logistic-regression probe at each layer (Appendix G.1):

$$p_r^{(l)}(x) = \sigma\Big((\mathbf{w}_r^{(l)})^\top \mathbf{h}^{(l)}(x) + b_r^{(l)}\Big), \quad r \in \{\text{sel}, \text{arg}\}, \tag{4.1}$$

where x denotes the serialized input, $\mathbf{w}_r^{(l)}$ and $b_r^{(l)}$ are learned probe parameters, and r indexes the tool-selection and argument-generation stages, respectively. As shown in Table 2, probes achieve 85% – 87% accuracy for tool selection and 77% – 79% for argument name, and control experiments rule out potential confounds (Appendix G.4&G.5). Extended results across all backbones in Appendix G.2.

Predicting hallucination from conflict. Distinguishing A from B alone does not ensure the detected signal relates to hallucination. We therefore test whether it predicts hallucination *within* conflicting inputs. We use the peak-layer probe output as a conflict score $P^r_{\text{conflict}}(x) = p_r^{(l_r^*)}(x)$, divide conflicting inputs into five quintiles, and measure hallucination rates per quintile. As shown in Table 2, for both models, TSHR and ANHR increase monotonically from Q_1 to Q_5, the extreme conflict-score quintiles (e.g., LLaMA-3.1-8B: TSHR 19.6% $\rightarrow$ 81.2%, ANHR 8.4% $\rightarrow$ 35.6%), with AUROCs of 0.81 – 0.84 for tool selection and 0.74 – 0.75 for argument name.

Table 2: Conflict detection and hallucination prediction.

Model	Conflict	Peak	Acc.↑	$Q_1 \rightarrow Q_5$	AUROC↑
LLaMA-3.1	Sel.	20 / 32	87.2%	19.6% $\rightarrow$ 81.2%	0.84
	Arg.N	23 / 32	79.4%	8.4% $\rightarrow$ 35.6%	0.75
Qwen-3.5	Sel.	19 / 32	85.6%	17.8% $\rightarrow$ 79.4%	0.81
	Arg.N	23 / 32	76.8%	7.2% $\rightarrow$ 33.8%	0.74

These results suggest that tendency conflict is not only behavioral but also encoded internally, with stronger conflict associated with higher hallucination. Armed with these probes, we next examine how existing methods affect these tendencies and their link to hallucination escape.

4.3 Analyzing Hallucination Escape in Existing Methods

The previous sections establish that hallucination rises when intrinsic tendencies conflict with the runtime configuration. A natural hypothesis is that existing methods reinforce these tendencies, explaining why they improve aligned configurations at the cost of conflicting ones. We test this by measuring how each method changes tendency strength.

Setup. We repeat closed-book elicitation before and after applying each method, computing the prior-strength change only for queries whose preferred tool remains unchanged: $\Delta S_m = \frac{1}{|\mathcal{Q}_m^{\text{ret}}|} \sum_{q \in \mathcal{Q}_m^{\text{ret}}} \big[S_m(q) - S_{\text{base}}(q)\big]$, where $\mathcal{Q}_m^{\text{ret}}$ denotes queries preserving the original preferred tool. To quantify the severity of escape, we define the weighted Escape Ratio $E_w(m) = O(m)/I(m)$, where $I(m)$ and $O(m)$ aggregate total improvement and deterioration across configurations; values above 1 indicate total deterioration exceeding improvement.

Table 3: Tendency reinforcement and escape.

Method	LLaMA			Qwen		
	ΔS	**Ret.**	**E_w ↓**	**ΔS**	**Ret.**	**E_w ↓**
Relign	+.11	80	1.57	+.07	84	1.38
PALADIN	+.08	82	1.52	+.08	83	1.78
Gorilla	+.14	76	2.00	+.11	80	1.86
LinSteer	+.07	86	1.92	+.09	86	2.42
PRISMS	+.05	91	1.17	+.04	93	1.13

Results. As shown in Table 3, 76% – 93% of queries retain their preferred tool after mitigation (denoted Ret. in Table 3), with prior strength increasing by 0.04 – 0.14 across all methods. Stronger reinforcement is consistently associated with larger escape ratios (E_w up to 2.42). The same pattern holds across queries: reinforced preferences overgeneralize to similar queries requiring other tools.

Taken together, the findings in §4.1 – §4.3 provide a coherent picture: models hold intrinsic tool-use tendencies that are associated with increased hallucination under conflict, are encoded in internal representations, and are reinforced by existing methods. This motivates a conflict-aware approach that mitigates tool hallucination while preventing escape, which we introduce next.

5 SOLUTION: ESCAPEGUARD

The analysis traces hallucination escape to intrinsic tendencies that existing methods reinforce. This motivates **ESCAPEGUARD**, an inference-time method that detects tendency conflicts and redirects attention to the runtime settings (Figure 5). We describe its design in §5.1 and evaluate it in §5.2&§5.3.

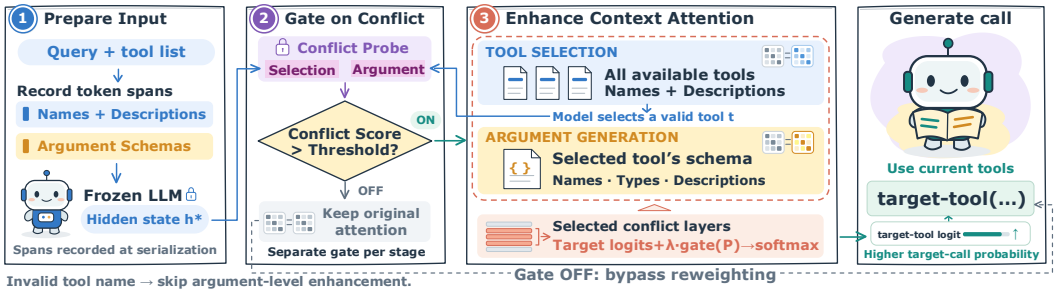

Figure 5: Overview of ESCAPEGUARD. Given a query and tool list, ESCAPEGUARD records tool-related token spans and extracts hidden states from the frozen LLM (❶). Conflict probes then score each input per stage and activate intervention only above a threshold (❷). When activated, attention is enhanced toward tool names and descriptions during selection, or toward the selected tool's schema during argument generation (❸).

5.1 DESIGN OF ESCAPEGUARD

The core idea of ESCAPEGUARD is to detect when a model's intrinsic tendencies conflict with the runtime tool configuration and, only in those cases, strengthen the model's attention to the relevant tool definitions. This requires addressing three questions: **what** to attend to, **when** to intervene, and **how** to enhance attention. Accordingly, ESCAPEGUARD operates in three stages: input preparation, conflict-aware gating, and context-specific attention enhancement, described below.

Stage ❶: Input Preparation. Given a query q and tool configuration C, ESCAPEGUARD first identifies the token spans that will serve as attention targets in later stages. Specifically, for each tool $t \in C$, it records the token spans of each tool's name ($S_{\text{name}}(t)$), description ($S_{\text{desc}}(t)$), and argument schema ($S_{\text{schema}}(t)$) during input serialization. These spans define **what** the model should attend to when intervention is triggered.

Stage ❷: Conflict-Aware Gating. ESCAPEGUARD next determines **when** to intervene. It reuses the frozen conflict probes from §4.2 to compute a conflict score for each generation stage r:

$$P^r_{\text{conflict}}(q, C) = \sigma\left(\mathbf{w}_r^\top \mathbf{h}^{(l_r^*)}(q, C) + b_r\right), \quad r \in \{\text{sel}, \text{arg}\}, \tag{5.1}$$

where l_r^* is the representative conflict layer. Intervention activates only when $P^r_{\text{conflict}}(q, C) > \tau_r$; otherwise the original computation is preserved. This gating mechanism ensures that ESCAPEGUARD modifies only high-conflict inputs, leaving low-conflict ones untouched.

Stage ❸: Context-Specific Attention Enhancement. When the gate activates, ESCAPEGUARD enhances attention toward the current tool configuration, addressing **how** to intervene. During tool selection, the target span covers all tool names and descriptions: $S_{\text{sel}}(C) = \bigcup_{t \in C}(S_{\text{name}}(t) \cup S_{\text{desc}}(t))$. After the model generates a tool t, the target shifts to its schema span $S_{\text{arg}}(C, t) = S_{\text{schema}}(t)$. For an attention head at the intervention layer, the logit $z_{ij} = \mathbf{q}_i^\top \mathbf{k}_j / \sqrt{d}$ from decoding token i to context token j is modified as

$$\tilde{z}_{ij} = z_{ij} + \lambda_r \frac{P^r_{\text{conflict}}(q, C) - \tau_r}{1 - \tau_r} \mathbf{1}[P^r_{\text{conflict}}(q, C) > \tau_r] \, \mathbf{1}[j \in S_r], \tag{5.2}$$

where λ_r is the maximum intervention strength, τ_r is the conflict threshold, and S_r denotes S_{sel} or S_{arg} accordingly. The factor $(P^r_{\text{conflict}} - \tau_r)/(1 - \tau_r)$ rescales the above-threshold conflict score to $[0, 1]$, so the bias increases linearly with conflict strength rather than switching on at a fixed magnitude. Attention is then computed normally via $\tilde{\alpha}_{ij} = \text{softmax}_j(\tilde{z}_{ij})$.

In summary, all three stages derive from the current tool configuration rather than a fixed correction, allowing ESCAPEGUARD to naturally adapt to tool renaming, replacement, and schema changes. This design mitigates tool hallucination by redirecting attention to the runtime configuration while avoiding the tendency reinforcement that causes hallucination escape in existing methods.

5.2 Experimental Setup

Having described the design of EscapeGuard, we now evaluate it across diverse models, benchmarks, and tool configurations.

Datasets. We evaluate on six benchmarks covering a range of tool-use scenarios: BFCL (Patil et al., 2025), Seal-Tools (Wu et al., 2024), RelyToolBench (Xu et al., 2025), API-Bank (Li et al., 2023), MetaTool (Huang et al., 2024), and APIBench. Subset selection follows our single-turn protocol; metric applicability and pool sizes are detailed in Table H.2. All evaluation data is disjoint from the probing corpus and hyperparameter selection to prevent information leakage.

Models, Baselines, and Metrics. We use the same six backbones from §4: LLaMA-3.1-8B/70B-Instruct (Meta, 2024), Qwen-3.5-9B/27B (Qwen Team, 2026), Mistral-7B-Instruct-v0.3 (Mistral AI, 2024a), and Mixtral-8x7B-Instruct-v0.1 (Mistral AI, 2024b). For baseline comparison, we evaluate all five baselines (Relign, PALADIN, Gorilla, LinSteer, PRISMS) alongside EscapeGuard on LLaMA-3.1-8B and Qwen-3.5-9B; the remaining four models are used to examine cross-model generalization. We report TSHR, TUHR (decomposed into ANHR and AVHR), and OHR as defined in §3. Per-benchmark metric applicability is in Table H.2.

Intervention Setup. The conflict threshold and attention bias are set to τ_r=0.6 and λ_r=1.8, selected on held-out data. For architectures with hybrid attention (e.g., Qwen-3.5), intervention is restricted to full-attention layers. Both hyperparameters exhibit a broad stable region around these values (sensitivity analysis in Appendix H.8). Model-specific intervention regions are listed in Appendix H.1.

5.3 Experimental Results

We evaluate EscapeGuard from four perspectives: overall effectiveness, baseline comparison, cross-configuration robustness, and component ablation study.

Overall Effectiveness. As shown in Table 4, EscapeGuard reduces TSHR for every model-benchmark pair, with improvements ranging from 4.3 pp to 14.7 pp. At the usage level (Table 5, the subscripts mean the change from the original model), EscapeGuard consistently lowers TUHR and ANHR across all six models. The improvement in AVHR is substantially smaller, consistent with the weaker association between argument-value hallucination and tendency conflict observed in §4. These gains come at modest cost: inference overhead is 4% – 6% of base latency (Appendix H.9).

Table 4: TSHR (↓%) across six backbones and six benchmarks. Subscripts: change from Base.

Model	BFCL	Seal-Tools	RelyTool.	API-Bank	MetaTool	APIBench
LLaMA-3.1-8B-Instruct	9.4$_{-9.0}$	11.2$_{-12.5}$	17.6$_{-9.7}$	6.3$_{-9.8}$	11.4$_{-10.2}$	9.8$_{-10.5}$
LLaMA-3.1-70B-Instruct	6.2$_{-4.4}$	7.6$_{-6.6}$	12.5$_{-4.3}$	4.5$_{-4.8}$	7.0$_{-6.1}$	6.9$_{-5.5}$
Qwen-3.5-9B	7.3$_{-9.8}$	9.6$_{-12.8}$	15.3$_{-11.2}$	4.6$_{-10.7}$	9.7$_{-11.1}$	8.1$_{-11.5}$
Qwen-3.5-27B	4.7$_{-5.1}$	6.5$_{-7.1}$	10.9$_{-4.5}$	3.4$_{-5.2}$	5.8$_{-6.7}$	5.8$_{-5.4}$
Mistral-7B-Instruct-v0.3	19.8$_{-11.0}$	22.5$_{-14.7}$	28.7$_{-10.9}$	16.7$_{-11.7}$	21.6$_{-13.8}$	20.1$_{-13.5}$
Mixtral-8x7B-Instruct-v0.1	13.8$_{-8.2}$	17.2$_{-10.4}$	20.6$_{-9.8}$	12.3$_{-8.0}$	16.7$_{-9.5}$	15.0$_{-9.6}$

Baseline Comparison. We compare EscapeGuard with all five baselines on LLaMA-3.1-8B using BFCL, Seal-Tools, API-Bank, and APIBench. As shown in Table 6, EscapeGuard achieves the lowest TSHR, TUHR, and ANHR across all four benchmarks, with more stable improvements than existing methods. While baselines reduce TSHR by 3 – 6 pp on average, EscapeGuard achieves 9 – 12 pp reductions. The advantage is most pronounced for ANHR, where EscapeGuard roughly halves the base rate while existing methods show marginal changes. The same pattern holds on Qwen-3.5-9B (Appendix H.5).

Table 5: Usage-level macro averages (%).

Model	TUHR ↓	ANHR ↓	AVHR ↓
LLaMA-3.1-8B-Instruct	8.6$_{-6.6}$	3.5$_{-5.2}$	6.5$_{-0.8}$
LLaMA-3.1-70B-Instruct	6.9$_{-3.9}$	2.6$_{-3.8}$	5.2$_{-0.6}$
Qwen-3.5-9B	6.7$_{-5.8}$	2.4$_{-4.9}$	5.6$_{-0.6}$
Qwen-3.5-27B	5.3$_{-2.8}$	3.0$_{-1.3}$	4.0$_{-0.7}$
Mistral-7B-Instruct-v0.3	19.0$_{-7.2}$	14.3$_{-4.5}$	16.9$_{-0.9}$
Mixtral-8x7B-Instruct-v0.1	12.9$_{-8.4}$	6.7$_{-6.9}$	12.3$_{-0.8}$

Cross-configuration Robustness. To directly test whether EscapeGuard prevents hallucination escape, we evaluate under the same $C_1 - C_6$ configurations from §3. As shown in Table 7, EscapeGuard reduces hallucination across all configurations. Under $C_1 - C_3$, rates converge

Table 6: Comparison with existing methods on LLaMA-3.1-8B-Instruct. Lower is better. Bold = best.

Dataset	Metric	Base	Relign	PALADIN	Gorilla	LinSteer	PRISMS	EscapeGuard
BFCL	TSHR/TUHR	18.4/14.6	14.1/13.7	14.9/14.1	13.5/15.4	15.6/14.7	15.1/14.1	**9.4/8.2**
	ANHR/AVHR	8.3/7.2	7.8/7.2	8.1/7.4	8.6/7.7	8.5/7.6	7.9/7.3	**3.7/6.7**
Seal-Tools	TSHR/TUHR	23.7/17.2	18.9/16.4	19.4/16.9	17.8/17.5	20.3/17.4	19.7/16.4	**11.2/9.2**
	ANHR/AVHR	9.9/8.4	9.2/8.5	9.4/8.6	10.3/9.2	10.1/9.0	9.3/8.4	**4.0/7.5**
API-Bank	TSHR/TUHR	16.1/12.3	12.0/11.4	12.7/12.0	11.5/13.0	13.3/12.6	13.0/11.7	**6.3/5.8**
	ANHR/AVHR	7.1/6.1	6.4/5.8	6.6/6.1	7.2/6.7	7.0/6.3	6.9/6.1	**2.8/5.2**
APIBench	TSHR/TUHR	20.3/16.1	15.8/15.1	16.3/15.6	14.4/16.8	17.2/16.4	16.8/15.5	**9.8/9.1**
	ANHR/AVHR	9.5/7.5	8.5/7.3	8.9/7.4	9.5/8.2	9.2/8.0	8.6/7.6	**3.5/6.6**

to 5.6 – 8.6%, effectively narrowing the performance gap among configurations. Mixed changes (C_5) see OHR reductions of 33.4 – 34.2 pp. The smallest gains appear under C_6 (13.2 – 13.6 pp), plausibly because injected distractors dilute the enhanced attention signal. For LLaMA and Qwen, the cross-configuration mean decreases by 24.2 and 23.3 pp while variance falls by 40.6% and 39.4%. For paired-query evaluation (Table 8), ESCAPEGUARD achieves 87.8% – 90.4% net improvement, maintaining a high repair rate while introducing substantially less transfer damage than baselines.

Table 7: ESCAPEGUARD under C_1 – C_6. Subscripts: change from Table 1. C_4/C_5: OHR; others: TSHR.

Model	C_1(TSHR)	C_2(TSHR)	C_3(TSHR)	C_4(OHR)	C_5(OHR)	C_6(TSHR)	$\bar{H}(m)\downarrow$	Var$(m)\downarrow$
LLaMA-3.1-8B-Instruct	$7.4_{-15.4}$	$8.6_{-41.8}$	$8.0_{-23.6}$	$29.6_{-16.6}$	$25.4_{-34.2}$	$24.4_{-13.6}$	$17.2_{-24.2}$	87.9
Qwen-3.5-9B	$5.6_{-13.8}$	$6.8_{-40.8}$	$6.2_{-23.2}$	$28.4_{-15.0}$	$23.4_{-33.4}$	$22.8_{-13.2}$	$15.5_{-23.3}$	90.4

Ablation Study. We ablate four components on LLaMA-3.1-8B and Qwen-3.5-9B (Appendix H.10, Table H.13). Each contributes: the conflict gate avoids unnecessary reweighting; stage-specific enhancement outperforms random context token enhancement; the correct conflict layers are essential; and input-dependent enhancement outperforms fixed steering vectors.

Table 8: Paired-query performance (%).

Metric	LLaMA-8B	Qwen-9B
Repair Rate ↑	91.6%	93.8%
Transfer Damage ↓	4.2%	3.6%
Net Improvement ↑	87.8%	90.4%

The results demonstrate that ESCAPEGUARD consistently reduces tool hallucination across various backbones and benchmarks, achieving the lowest TSHR, TUHR, and ANHR in baselines comparison with all five existing methods. Crucially, unlike existing methods that exhibit hallucination escape, ESCAPEGUARD reduces hallucination across all six tool configurations, lowering the cross-configuration mean by over 23.7 pp while achieving 87.8% – 90.4% net improvement in paired-query evaluation. These results show that conflict-aware, configuration-grounded intervention can mitigate tool hallucination without the escape effect that undermines existing approaches.

6 DISCUSSION

ESCAPEGUARD consistently reduces tool hallucination and prevents hallucination escape across six backbones as an inference-time method. That said, the six tool configurations and architectures studied, though representative, are not exhaustive. The tendency-conflict mechanism accounts weakly for argument-value errors, and ESCAPEGUARD's AVHR improvements are correspondingly modest. Gains are also smallest under distractor injection (C_6), likely because confusable tools dilute the enhanced attention signal. Addressing argument-value errors, improving distractor robustness, and extending to broader architectures remain promising directions for future work.

7 CONCLUSION

We identify hallucination escape, a failure mode in which mitigation methods reduce tool hallucination on their tuned configuration but increase it on others. We trace this to intrinsic tool-use tendencies that existing methods reinforce, and propose ESCAPEGUARD, a training-free method that detects tendency conflicts and redirects attention to the runtime configuration. Across various models and benchmarks, ESCAPEGUARD reduces tool-selection hallucination by 9.0 pp while effectively suppressing hallucination escape. We hope this work encourages evaluation beyond a single tool configuration and contributes to more reliable tool-using LLM agents.

AI USE STATEMENT

We used generative AI tools to assist with English-language editing, LaTeX integration, and formatting of this manuscript. The experimental protocol also uses GPT-5.6-Terra to construct tool configuration variants (Appendix A). All AI-assisted revisions were reviewed and edited by the authors to preserve the intended scientific meaning and ensure consistency with the methods, experimental results, and conclusions. We take full responsibility for the final content of this work, including all text and artifacts produced with the assistance of generative AI.

ETHICS STATEMENT

This work studies and mitigates tool hallucination in LLM agents using publicly available models and benchmarks. ESCAPEGUARD operates as a training-free inference-time intervention that does not modify model weights or introduce new training data. All tool configurations do not involve real user data or API calls. We do not foresee negative societal impacts specific to this work; on the contrary, reducing tool hallucination contributes to more reliable deployment of tool-using LLM agents.

REPRODUCIBILITY STATEMENT

Appendices A – D document configuration construction, query pairs, data scope, and metrics. Appendices E – H provide additional results, behavioral and probing protocols, intervention regions, sensitivity analyses, and efficiency measurements. Code and data will be released upon publication.

REFERENCES

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and Lichao Sun. Metatool benchmark for large language models: Deciding whether to use tools and which to use. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

Yutong Ke, Ming Yin, and et al. A few neurons reveal when llms misuse tools: Sparse detection and selective steering for reliable tool use. *CoRR*, abs/2608.00218, 2026. doi: 10.48550/ARXIV.2608.00218. URL https://doi.org/10.48550/arXiv.2608.00218.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 3102–3116. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.187. URL https://doi.org/10.18653/v1/2023.emnlp-main.187.

Xuannan Liu, Xiao Yang, Zekun Li, Pei-Pei Li, and Ran He. Agenthallu: Benchmarking automated hallucination attribution of llm-based agents. *CoRR*, abs/2601.06818, 2026. doi: 10.48550/ARXIV.2601.06818. URL https://doi.org/10.48550/arXiv.2601.06818.

Meta. Llama 3.1 official model cards: 8B-Instruct and 70B-Instruct, 2024. URL https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct. The 70B checkpoint is available under the same organization as Llama-3.1-70B-Instruct.

Mistral AI. Mistral-7B-Instruct-v0.3 model card, 2024a. URL https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3.

Mistral AI. mistralai/Mixtral-8x7B-Instruct-v0.1 model card, 2024b. URL https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): from tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Qwen Team. Qwen3.5 model cards, 2026. URL https://huggingface.co/collections/Qwen/qwen35. Checkpoints: Qwen3.5-9B and Qwen3.5-27B.

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. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

Sri Vatsa Vuddanti, Aarav Shah, Satwik Kumar Chittiprolu, Tony Song, Sunishchal Dev, Kevin Zhu, and Maheep Chaudhary. PALADIN: self-correcting language model agents to cure tool-failure cases. *CoRR*, abs/2509.25238, 2025. doi: 10.48550/ARXIV.2509.25238. URL https://doi.org/10.48550/arXiv.2509.25238.

Mengsong Wu, Tong Zhu, Han Han, Chuanyuan Tan, Xiang Zhang, and Wenliang Chen. Seal-tools: Self-instruct tool learning dataset for agent tuning and detailed benchmark. In *Natural Language Processing and Chinese Computing - 13th National CCF Conference, NLPCC 2024, Hangzhou, China, November 1-3, 2024, Proceedings, Part II*, volume 15360 of *Lecture Notes in Computer Science*, pp. 372–384. Springer, 2024. doi: 10.1007/978-981-97-9434-8_29. URL https://doi.org/10.1007/978-981-97-9434-8_29.

Zekun Wu, Ze Wang, Seonglae Cho, Yufei Yang, Adriano S. Koshiyama, Sahan Bulathwela, and María Pérez-Ortiz. Tool calling is linearly readable and steerable in language models. *CoRR*, abs/2605.07990, 2026. doi: 10.48550/ARXIV.2605.07990. URL https://doi.org/10.48550/arXiv.2605.07990.

Qian Xiong, Yuekai Huang, Ziyou Jiang, Zhiyuan Chang, Yu Zheng, Tianhao Li, and Mingyang Li. Butterfly effects in toolchains: A comprehensive analysis of failed parameter filling in LLM tool-agent systems. In *Findings of the Association for Computational Linguistics: EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 16712–16729. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-EMNLP.907. URL https://doi.org/10.18653/v1/2025.findings-emnlp.907.

Hongshen Xu, Zichen Zhu, Lei Pan, Zihan Wang, Su Zhu, Da Ma, Ruisheng Cao, Lu Chen, and Kai Yu. Reducing tool hallucination via reliability alignment. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/xu25ap.html.

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, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Yuxiang Zhang, Jing Chen, and et al. Toolbehonest: A multi-level hallucination diagnostic benchmark for tool-augmented large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pp. 11388–11422. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.EMNLP-MAIN.637.

A TOOL CONFIGURATION CONSTRUCTION

A.1 CONFIGURATION CONSTRUCTION RULES

Tool Replacement Rules (C_2). The replacement tool name uses different lexical forms to reduce direct matching to the original. For example, `currency_conversion.get_rate` → `fx_historical_lookup` (rather than `get_currency_rate`, which retains the keywords "currency" and "rate," allowing the model to infer functionality). The functional description remains complete and accurate, ensuring that the correct answer can be obtained by reading the description.

Description Change Rules (C_3). The tool name remains unchanged while the functional description's wording and emphasis are altered, reflecting the versatility of the same tool across different scenarios. For example, the description of `currency_conversion.get_rate` is "Get the currency conversion rate from one currency to another at a specified date," which is changed to "Retrieve the foreign exchange mid-market reference between two ISO 4217 currency codes for a given valuation date." The correct tool-query mapping remains intact.

Argument Renaming Rules (C_4). Argument names are replaced with alternative surface forms to reduce reliance on familiar names. For example, `from_currency` → `source_unit`, `to_currency` → `target_unit`, `date` → `temporal_ref`. The functional descriptions of arguments remain accurate, while their surface forms change.

Mixed Changes Rules (C_5). The tool name, functional description , and argument names are all simultaneously replaced, with argument order randomly shuffled. This represents the most comprehensive change scenario closest to real-world deployments.

Distractor Tool Construction Rules (C_6). Injected distractor tools satisfy the following conditions: (1) the tool name is highly similar to high-frequency tool names in the model's prior (e.g., `currency_conversion.get_rate_forecast` vs. `currency_conversion.get_rate`), maximizing naming confusion; (2) the description partially overlaps with but does not fully match the correct tool (e.g., "Get the forecasted future conversion rate trend between two currencies" vs. "Get the currency conversion rate from one currency to another at a specified date"), testing the model's fine-grained discrimination; (3) 2–3 distractor tools are injected per query, ensuring the total tool list length does not exceed 10 (to avoid confounding factors from excessive context length).

Configuration Style Note. All mitigation methods are trained/calibrated under the standard naming style C_1, the default in each original paper. C_2–C_6 constitute out-of-distribution test scenarios.

A.2 CONFIGURATION CONSTRUCTION EXAMPLES

The following five examples are drawn from the BFCL V3 single-turn portion, Glaive Function Calling V2, and Seal-Tools, covering five domains: finance, communication, translation, meteorology, and travel, to illustrate the systematic changes in tool configuration under C_1–C_6 for the same query.

EXAMPLE 1: EXCHANGE RATE QUERY

Query. "Find the conversion rate from Euro to Dollar at January 1, 2022"

Original tool list (3 tools; correct tool is `currency_conversion.get_rate`):

Tool and description	Original argument schema
`currency_conversion.convert` Converts a specified amount of money from one currency to another at the latest rate.	`from_currency` (string): The currency that you want to convert from. `to_currency` (string): The currency that you want to convert to. `amount` (float): The amount of money that you want to convert. *Required:* `from_currency`, `to_currency`, `amount`.
`currency_conversion.get_latest_rate` Get the latest currency conversion rate from one currency to another.	`from_currency` (string): The currency that you want to convert from. `to_currency` (string): The currency that you want to convert to. *Required:* `from_currency`, `to_currency`.
`currency_conversion.get_rate` Get the currency conversion rate from one currency to another at a specified date.	`from_currency` (string): The currency that you want to convert from. `to_currency` (string): The currency that you want to convert to. `date` (string): The date at which the conversion rate applies. Default is the current date. *Required:* `from_currency`, `to_currency`.

C_1–C_6 transformation results (showing only changes to the correct tool `currency_conversion.get_rate`; other tools remain unchanged):

Table A.2: Configuration for example 1. Only the correct tool changes in C_1–C_5; C_6 adds distractors.

C	Tool name	Description	Arguments
C_1	`currency_conversion.get_rate`	Get the currency conversion rate from one currency to another at a specified date	`from_currency`, `to_currency`, `date`
C_2	`fx_historical_lookup`	Get the currency conversion rate from one currency to another at a specified date	`from_currency`, `to_currency`, `date`
C_3	`currency_conversion.get_rate`	Retrieve the foreign exchange mid-market reference rate between two ISO 4217 currency codes for a given valuation date	`from_currency`, `to_currency`, `date`
C_4	`currency_conversion.get_rate`	Get the currency conversion rate from one currency to another at a specified date	`source_unit`, `target_unit`, `temporal_ref`
C_5	`fx_historical_lookup`	Retrieve the foreign exchange mid-market reference rate between two monetary units for a given valuation date	`source_unit`, `target_unit`, `temporal_ref`
C_6	`currency_conversion.get_rate_forecast`, `conversion_rate_history`	Get the currency conversion rate from one currency to another at a specified date	`from_currency`, `to_currency`, `date`

EXAMPLE 2: EMAIL SENDING

Query. "I need to send an email to my boss. The recipient's email is boss@company.com. The subject is 'Project Update'. And the message is 'Dear Boss, I have completed the project as per the given deadline.' "

Original tool list (2 tools; correct tool is `send_email`):

Tool and description	Original argument schema
`send_email` Send an email to a recipient	`recipient` (string): The email address of the recipient `subject` (string): The subject of the email `message` (string): The content of the email *Required:* `recipient`, `subject`, `message`.
`search_recipes` Search for recipes based on ingredients	`ingredients` (array of string): The ingredients to search for *Required:* `ingredients`.

C_1–C_6 transformation results (showing only changes to the correct tool `send_email`; other tools remain unchanged):

Table A.4: Configuration for example 2. Only the correct tool changes in C_1–C_5; C_6 adds distractors.

C	Tool name	Description	Arguments
C_1	`send_email`	Send an email to a recipient	`recipient`, `subject`, `message`
C_2	`msg_dispatch`	Send an email to a recipient	`recipient`, `subject`, `message`
C_3	`send_email`	Push a formatted notification via SMTP protocol to the target mailbox	`recipient`, `subject`, `message`
C_4	`send_email`	Send an email to a recipient	`target_addr`, `msg_title`, `content_payload`
C_5	`notify_relay`	Deliver a message to a specified receiver through a communication channel	`target_addr`, `msg_title`, `content_payload`
C_6	`send_email_template`, `email_draft_manager`	Send an email to a recipient	`recipient`, `subject`, `message`

EXAMPLE 3: TRANSLATION

Query. 'Tell me how to say "Today is a beautiful day" in Amharic.'

Original tool list (5 tools; correct tool is `translateAmharic`):

Tool and description	Original argument schema
`translateAmharic` Translate text to Amharic language *Field:* Foreign Languages/Amharic	`text` (str): The text to be translated to Amharic *Required:* `text`.
`translateToHebrew` Translate a text to Hebrew *Field:* Foreign Languages/Hebrew	`text` (str): The text to be translated `language` (str): The source language of the text (e.g., English, Spanish, French) *Required:* `text`.
`translateHausa` Translate text from Hausa to English or vice versa *Field:* Foreign Languages/Hausa	`text` (str): The text to be translated *Required:* `text`.
`pronounceXhosaPhrase` Get the pronunciation of a given Xhosa phrase *Field:* Foreign Languages/Xhosa	`phrase` (str): The Xhosa phrase for which you want to get the pronunciation *Required:* `phrase`.
`translateIgbo` Translate a word or phrase from English to Igbo *Field:* Foreign Languages/Igbo	`word` (str): The word or phrase to be translated from English to Igbo *Required:* `word`.

C_1–C_6 transformation results (showing only changes to the correct tool `translateAmharic`; other tools remain unchanged):

Table A.6: Configuration for example 3. Only the correct tool changes in C_1–C_5; C_6 adds distractors.

C	Tool name	Description	Arguments
C_1	`translateAmharic`	Translate text to Amharic language	`text`
C_2	`lang_convert_amh`	Translate text to Amharic language	`text`
C_3	`translateAmharic`	Convert input string into the Amharic script using a neural translation model	`text`
C_4	`translateAmharic`	Translate text to Amharic language	`input_sequence`
C_5	`script_render_amh`	Transcode the source string into the target Ge'ez-script language	`input_sequence`
C_6	`translateAmharicAudio`, `amharicDictionary`	Translate text to Amharic language	`text`

EXAMPLE 4: METEOROLOGICAL DATA

Query. "What is the wind speed and temperature in location given by coordinates 46.603354, 1.888334 on December 13, 2019?"

Original tool list (3 tools; correct tool is `weather.get_by_coordinates_date`):

Tool and description	Original argument schema
`weather.get_by_city_date` Retrieves the historical weather data based on city and date.	`city` (string): The city for which to retrieve the weather. `date` (string): The date for which to retrieve the historical weather data in the format YYYY-MM-DD. *Required:* `city`, `date`.
`weather.get_forecast_by_coordinates` Get the weather forecast for a specific geographical coordinates.	`coordinates` (tuple): The geographical coordinates (latitude, longitude). `days_ahead` (integer): Number of days to forecast from current date (default is 7). *Required:* `coordinates`.
`weather.get_by_coordinates_date` Retrieves the historical weather data based on coordinates and date.	`coordinates` (tuple): The geographical coordinates (latitude, longitude). `date` (string): The date for which to retrieve the historical weather data in the format YYYY-MM-DD. *Required:* `coordinates`, `date`.

C_1–C_6 transformation results (showing only changes to the correct tool `weather.get_by_coordinates_date`; other tools remain unchanged):

Table A.8: Configuration for example 4. Only the correct tool changes in C_1–C_5; C_6 adds distractors.

C	Tool name	Description	Arguments
C_1	`weather.get_by_coordinates_date`	Retrieves the historical weather data based on coordinates and date	`coordinates`, `date`
C_2	`geo_obs_historical`	Retrieves the historical weather data based on coordinates and date	`coordinates`, `date`
C_3	`weather.get_by_coordinates_date`	Query multi-parameter observations for a given geographic point and calendar date	`coordinates`, `date`
C_4	`weather.get_by_coordinates_date`	Retrieves the historical weather data based on coordinates and date	`geo_point`, `temporal_ref`
C_5	`geo_obs_historical`	Query multi-parameter observations for a given geographic point and calendar date	`geo_point`, `temporal_ref`
C_6	`weather.get_coordinates_date_summary`, `weather.coordinates_date_model`	Retrieves the historical weather data based on coordinates and date	`coordinates`, `date`

EXAMPLE 5: FLIGHT BOOKING

Query. "I need to book a flight to Paris. I want to leave on the 15th of next month and return on the 30th. We are a group of 5."

Original tool list (2 tools; correct tool is `book_flight`):

Tool and description	Original argument schema
`book_flight` Book a flight for a specific destination	`destination` (string): The desired destination `departure_date` (string): The date of departure `return_date` (string): The date of return `passenger_count` (integer): The number of passengers flying *Required:* `destination`, `departure_date`, `return_date`, `passenger_count`.
`create_calendar_event` Create a new calendar event	`event_title` (string): The title of the event `start_time` (string): The start time of the event `end_time` (string): The end time of the event *Required:* `event_title`, `start_time`, `end_time`.

C_1–C_6 transformation results (showing only changes to the correct tool `book_flight`; other tools remain unchanged):

Table A.10: Configuration for example 5. Only the correct tool changes in C_1–C_5; C_6 adds distractors.

C	Tool name	Description	Arguments
C_1	`book_flight`	Book a flight for a specific destination	`destination`, `departure_date`, `return_date`, `passenger_count`
C_2	`trip_reservation`	Book a flight for a specific destination	`destination`, `departure_date`, `return_date`, `passenger_count`
C_3	`book_flight`	Reserve air transport seats on a scheduled carrier route for specified travel dates	`destination`, `departure_date`, `return_date`, `passenger_count`
C_4	`book_flight`	Book a flight for a specific destination	`target_city`, `depart_ref`, `return_ref`, `pax_count`
C_5	`air_route_reserve`	Secure passage on a commercial air carrier for the designated itinerary	`target_city`, `depart_ref`, `return_ref`, `pax_count`
C_6	`book_flight` + distractors: `book_flight_insurance`, `flight_price_alert`	Book a flight for a specific destination	`destination`, `departure_date`, `return_date`, `passenger_count`

A.3 Configuration Quality Control

All configurations ($C_2 - C_6$) were initially generated with LLM assistance and then independently verified and refined by two authors for consistency. The construction process followed three stages.

LLM-assisted generation and independent verification. Configuration variants were initially generated by prompting an LLM with the rules in Section A.1. The full set of variants was then independently reviewed and refined through manual verification, correcting cases where the generated output violated a construction rule or introduced unintended cues. Reviewers worked from the same generated candidates without sharing intermediate edits.

Reconciliation. The two sets were compared at the tool-name, description, argument-name, and distractor levels. Discrepancies (averaging 8.3% of entries across configurations) were resolved by discussion, preferring for the variant that more completely satisfied the stated construction rules.

Automated validation. After reconciliation, we ran automated checks to verify that (1) every configuration variant preserves a valid reference solution (i.e., the correct tool and its required arguments remain present and invocable), (2) C_2 tool names share no content words with the original, (3) C_3 descriptions preserve the original tool's functional scope, (4) C_4 argument names differ from the originals in surface form while the schema remains structurally valid, and (5) C_6 distractor tools do not duplicate any existing tool in the list. All queries passed these checks. The validation scripts will be released with the supplementary code.

B Query Pair Construction Rules

Query pairs satisfy: (1) they share the same tool list and each has a unique correct tool; (2) they are semantically related but involve different tasks; (3) the correct tools' functional descriptions are sufficient to distinguish them. Pairs are constructed from the BFCL V3 single-turn multi-tool portion, with 30 pairs per functional domain across four domains. Below are representative examples.

Table B.1: Query pair examples from the BFCL V3 multiple subset.

Target query / correct tool	Related query / correct tool
What is the capital of Brazil? `country_info.capital`	What is the largest city in Brazil? `country_info.largest_city`
Calculate the displacement of a car moving at 20 m/s with 10 m/s^2 acceleration for 5 seconds `kinematics.calculate_displacement`	Calculate the final speed of a car moving at 20 m/s with 10 m/s^2 acceleration for 5 seconds `kinematics.calculate_final_speed`
What is the wind speed in location 46.6, 1.9 on December 13, 2019? `weather.get_by_coordinates_date`	What is the 7-day weather forecast for coordinates 46.6, 1.9? `weather.get_forecast_by_coordinates`
Convert 100 Euros to US Dollars at the latest rate `currency_conversion.convert`	Find the conversion rate from Euro to Dollar on January 1, 2022 `currency_conversion.get_rate`

C Dataset Selection and Data Construction

This appendix describes the data used for controlled evaluation in § 3, behavioral analysis in § 4.1, and internal representation analysis in § 4.2.

C.1 Controlled and Paired-Query Evaluation

The controlled evaluation uses the single-turn portions of BFCL V3 and Seal-Tools, whose explicit tool definitions and reference calls support controlled configuration changes while preserving a valid solution. Approximately 1,000 source queries yield approximately 6,000 evaluations across C_1–C_6, with three random seeds, per configuration to account for sampling variance.

The paired-query evaluation uses the BFCL V3 single-turn multi-tool subset: 30 pairs in each of four domains, totaling 120 pairs (240 queries). Each pair shares one tool list but requires different correct tools (Appendix B), isolating query-level effects.

C.2 Behavioral Analysis Set

§ 4.1 uses a behavioral set of 300 queries sampled equally from BFCL, Seal-Tools, and Glaive Function Calling V2. We select 100 queries from each benchmark to reduce dataset-specific bias while retaining diverse tool names, descriptions, and argument schemas.

All selected examples satisfy three criteria: the correct tool is explicitly identifiable, the runtime tool configuration contains a structured argument schema, and the example supports the construction of both tool-selection and argument-level conflicts. For Seal-Tools, we retain single-tool examples with complete tool definitions and valid argument schemas. For Glaive Function Calling V2, we retain examples with unambiguous reference calls and parsable structured arguments.

For each query, we construct three conditions: the aligned condition A, the tool-selection conflict condition B_{tool}, and the argument-level conflict condition B_{arg}. In A, the runtime tool and argument schema are consistent with the model's elicited preference. In B_{tool}, the preferred tool is replaced by a functionally equivalent alternative with a different name and a meaning-preserving description rewrite, while the argument schema remains unchanged. In B_{arg}, the tool name and description remain unchanged, but argument names or types are modified while preserving a valid solution.

For closed-book elicitation, we sample $K = 10$ outputs per query at temperature 0.7. Contextual evaluation under A, B_{tool}, and B_{arg} uses greedy decoding.

C.3 Probing Corpus and Data Splits

The internal representation analysis in § 4.2 expands the behavioral set into a corpus of 1,200 source queries. All 300 behavioral queries are retained, allowing the internal analysis to directly cover the observations established in § 4.1. We add 900 queries from the same three sources to increase sample diversity and improve the stability of layerwise localization: 300 from BFCL, 200 from Seal-Tools, and 400 from Glaive Function Calling V2. Table C.1 summarizes the corpus composition.

Table C.1: Composition of the 1,200-query probing corpus. Counts refer to source queries before constructing condition variants.

Dataset	Behavioral	Added	Total
BFCL	100	300	400
Seal-Tools	100	200	300
Glaive Function Calling V2	100	400	500
Total	**300**	**900**	**1,200**

For each source query, we attempt to construct A, B_{tool}, and the argument-name subset of B_{arg}, following Appendix C.2. Not every source query yields a valid variant for every condition (e.g., some queries lack suitable alternative tool names for B_{tool}), resulting in slight class imbalance across conditions (52.1%–53.4% aligned); see Appendix G.4 for its effect on probe baselines. All variants derived from the same source query are assigned to the same split to prevent leakage between aligned and conflicting conditions.

We reserve 240 source queries (20% of the corpus) as a fixed held-out test set. The remaining 960 queries are divided into 720 training and 240 validation queries.

We repeat probe fitting over five stratified random splits of the training and validation portion while keeping the test set fixed. Dataset source and conflict type are stratified across splits. All split sizes refer to source queries rather than their constructed condition variants.

D Mathematical Definitions of Evaluation Metrics

D.1 Basic Metrics

Let $\mathcal{Q}$ be the query set, c_k a configuration, and m a method; $m = \text{base}$ is the unmodified model. The output for q_i is $\hat{y}_i = (\hat{t}_i, \hat{p}_i)$, with $\hat{p}_i = \{(\hat{n}_j, \hat{v}_j)\}$. The reference is $y_i^* = (t_i^*, p_i^*)$. Let $\mathcal{T}(c_k)$ contain the available tools and $\mathcal{N}(t_i^*, c_k)$ their valid argument names.

Tool selection. Unavailable and functionally incorrect tools count as hallucinations:

$$I_{\mathrm{TS}}(q_i, c_k, m) = \begin{cases} 1, & \hat{t}_i \notin \mathcal{T}(c_k), \\ 1, & \hat{t}_i \in \mathcal{T}(c_k) \text{ and } \hat{t}_i \neq t_i^*, \\ 0, & \text{otherwise.} \end{cases} \tag{D.1}$$

The cases represent fabrication, misselection, and correct selection, respectively.

$$\mathrm{TSHR}(m, c_k) = \frac{1}{|\mathcal{Q}|} \sum_{q_i \in \mathcal{Q}} I_{\mathrm{TS}}(q_i, c_k, m). \tag{D.2}$$

Tool usage. Define $\mathcal{Q}_{\mathrm{correct}}(m, c_k) = \{q_i : \hat{t}_i = t_i^*\}$. Conditional on correct selection,

$$I_{\mathrm{AN}} = \mathbf{1}[\exists(\hat{n}_j, \hat{v}_j) \in \hat{p}_i : \hat{n}_j \notin \mathcal{N}(t_i^*, c_k)], \tag{D.3}$$

$$I_{\mathrm{AV}} = \mathbf{1}[\exists(\hat{n}_j, \hat{v}_j) \in \hat{p}_i : \hat{n}_j \in \mathcal{N}(t_i^*, c_k),\ \hat{v}_j \neq v_j^*], \tag{D.4}$$

$$I_{\mathrm{TU}} = \max\{I_{\mathrm{AN}}, I_{\mathrm{AV}}\}. \tag{D.5}$$

For $R \in \{\mathrm{TU}, \mathrm{AN}, \mathrm{AV}\}$, the corresponding TUHR, ANHR, or AVHR is

$$\mathrm{RHR}(m, c_k) = \frac{1}{|\mathcal{Q}_{\mathrm{correct}}(m, c_k)|} \sum_{q_i \in \mathcal{Q}_{\mathrm{correct}}(m, c_k)} I_R(q_i, c_k, m). \tag{D.6}$$

A call may contain both subtypes, so ANHR and AVHR need not sum to TUHR. Usage rates are undefined when the conditional denominator is zero.

Overall hallucination. A query is counted as hallucinated if either tool selection is incorrect or tool selection is correct but argument usage is hallucinated. With rates expressed as proportions,

$$\mathrm{OHR} = \mathrm{TSHR} + (1 - \mathrm{TSHR})\mathrm{TUHR}. \tag{D.7}$$

For percentages, use $\mathrm{OHR}_{\%} = \mathrm{TSHR}_{\%} + (1 - \mathrm{TSHR}_{\%}/100)\mathrm{TUHR}_{\%}$. Configurations C_4 and C_5 report OHR because argument usage is also affected.

D.2 Cross-Scenario Aggregation

Let $\mathrm{HR}(m, c_k)$ be the reported metric: TSHR for C_1–C_3 and C_6, and OHR for C_4/C_5. With $K = 6$,

$$\bar{H}(m) = K^{-1} \sum_{k=1}^{K} \mathrm{HR}(m, c_k), \tag{D.8}$$

$$\Delta\mathrm{HR}(m, c_k) = \mathrm{HR}(m, c_k) - \mathrm{HR}(\mathrm{base}, c_k), \tag{D.9}$$

$$\mathrm{Var}(m) = K^{-1} \sum_{k=1}^{K} [\mathrm{HR}(m, c_k) - \bar{H}(m)]^2. \tag{D.10}$$

This mean summarizes the designated configuration-specific metrics, rather than pooled OHR. Variance uses squared percentage points when rates are percentages. A method that leaves the mean unchanged but increases the variance produces a more uneven hallucination distribution—improving on some configurations while worsening on others.

D.3 Query Transfer Metrics

Let $\mathcal{P} = \{(q_i, q_i')\}_{i=1}^{N}$ contain target/related pairs sharing a tool list. Define the initially incorrect targets, repaired targets, and repairs introducing new related-query errors as

$$\mathcal{B} = \{i : I_{\mathrm{TS}}(q_i, \mathrm{base}) = 1\}, \tag{D.11}$$

$$\mathcal{R}_m = \{i \in \mathcal{B} : I_{\mathrm{TS}}(q_i, m) = 0\}, \tag{D.12}$$

$$\mathcal{D}_m = \{i \in \mathcal{R}_m : I_{\mathrm{TS}}(q_i', \mathrm{base}) = 0,\ I_{\mathrm{TS}}(q_i', m) = 1\}. \tag{D.13}$$

Then

$$\text{Repair}(m) = |\mathcal{R}_m|/|\mathcal{B}|, \tag{D.14}$$

$$\text{TransferDamage}(m) = |\mathcal{D}_m|/|\mathcal{R}_m|, \tag{D.15}$$

$$\text{Net}(m) = \text{Repair}(m)[1 - \text{TransferDamage}(m)]. \tag{D.16}$$

The shared configuration argument is omitted. Net Improvement discounts the repair rate by the fraction of repairs that introduce collateral errors, thus reflecting the actual gain of a method after accounting for transfer damage. These formulas use proportions; multiply by 100 for percentages.

D.4 TSHR-Only Cross-Scenario Aggregation

The main text aggregates the designated per-configuration metric (TSHR for $C_1 - C_3$ and C_6; OHR for C_4/C_5) because C_4 and C_5 affect argument usage. To verify that this heterogeneous aggregation does not distort the conclusions, we repeat the cross-scenario analysis using TSHR uniformly across all six configurations. For C_4 and C_5, TSHR is back-calculated from OHR (Table 1) and the per-configuration TUHR (Tables E.1 and E.2) via $\text{TSHR} = (\text{OHR} - \text{TUHR})/(1 - \text{TUHR}/100)$. For EscapeGuard, C_4/C_5 TSHR is similarly derived from the OHR values in Table 7 and the corresponding EscapeGuard TUHR.

Table D.1: TSHR-only cross-scenario aggregation on LLaMA-3.1-8B-Instruct and Qwen-3.5-9B (%). $\bar{H}_{\text{TS}}$ is the mean TSHR across $C_1 - C_6$; Var_{TS} is the corresponding variance.

	LLaMA-3.1-8B				Qwen-3.5-9B			
Method	$\bar{H}_{\text{TS}}$	$\Delta\bar{H}_{\text{TS}}$	Var_{TS}	ΔVar	$\bar{H}_{\text{TS}}$	$\Delta\bar{H}_{\text{TS}}$	Var_{TS}	ΔVar
Base	36.0	—	101.3	—	32.5	—	107.4	—
Relign	37.3	+1.3	208.9	+106.2%	33.6	+1.1	213.0	+98.3%
PALADIN	37.0	+1.0	191.6	+89.1%	33.7	+1.2	196.4	+82.9%
Gorilla	38.4	+2.4	269.0	+165.5%	34.5	+2.0	250.5	+133.2%
LinSteer	37.1	+1.1	175.4	+73.1%	33.7	+1.2	174.7	+62.7%
PRISMS	36.4	+0.4	172.4	+70.2%	32.9	+0.4	179.2	+66.9%
EscapeGuard	10.3	−25.7	40.0	−60.5%	8.3	−24.2	43.2	−59.8%

The TSHR-only aggregation confirms hallucination escape under a homogeneous metric: every existing method yields $\bar{H}_{\text{TS}}$ above the base model (by 0.4 – 2.4 pp for LLaMA and 0.4 – 2.0 pp for Qwen), satisfying Definition 3.3, while variance increases by 62.7% – 165.5%. EscapeGuard substantially reduces both mean (by 25.7/24.2 pp) and variance (by ≈60%), with all conclusions unchanged relative to the designated-metric aggregation.

E Supplementary Controlled Evaluation Results

E.1 Tool-Usage Hallucination under Configuration Changes

Finding 2 in § 3 reports that the usage component within OHR is largely unaffected by existing methods. Tables E.1 and E.2 provide the full per-configuration TUHR breakdown for LLaMA-3.1-8B-Instruct and Qwen-3.5-9B.

In both models, while base TUHR rises noticeably under C_4 and C_5 due to argument renaming, every mitigation method changes TUHR by no more than 1.5 percentage points relative to the corresponding base value in any configuration. This confirms that the marginal C_4 OHR changes observed in Table 1 stem primarily from small TSHR improvements, while the dominant TUHR component remains largely stable across all methods (though not perfectly constant—e.g., Relign reduces C_4 TUHR by 0.9 pp on LLaMA).

Table E.1: TUHR (%) across C_1 – C_6 for LLaMA-3.1-8B-Instruct. Changes are minimal across configurations.

Method	C_1	C_2	C_3	C_4	C_5	C_6	Mean
Base	12.4	14.1	13.8	26.8	24.3	15.7	17.9
Relign	11.8	13.6	13.2	25.9	23.7	15.1	17.2
PALADIN	12.0	13.8	13.5	26.3	24.0	15.4	17.5
Gorilla	11.2	13.9	13.4	26.2	23.5	15.4	17.3
LinSteer	12.1	14.3	14.0	27.1	24.6	15.9	18.0
PRISMS	11.9	14.0	13.6	26.5	24.1	15.6	17.6

Table E.2: TUHR (%) across C_1 – C_6 for Qwen-3.5-9B. The same near-immunity pattern holds.

Method	C_1	C_2	C_3	C_4	C_5	C_6	Mean
Base	11.6	13.2	12.9	28.4	25.7	14.3	17.7
Relign	11.1	12.8	12.4	27.6	25.1	13.9	17.2
PALADIN	11.3	13.0	12.7	28.0	25.4	14.1	17.4
Gorilla	10.8	13.0	12.6	28.1	25.3	14.1	17.3
LinSteer	11.4	13.5	13.1	28.7	26.0	14.5	17.9
PRISMS	11.2	13.1	12.8	28.2	25.5	14.2	17.5

E.2 Shared-Correct Subset Analysis

Because TUHR conditions on correct tool selection, different methods yield different denominator sets $\mathcal{Q}_{\text{correct}}(m, c_k)$. To verify that the limited sensitivity is not an artifact of denominator shift, we restrict the analysis to the shared-correct subset $\mathcal{Q}_{\text{shared}}(m, c_k) = \mathcal{Q}_{\text{correct}}(\text{base}, c_k) \cap \mathcal{Q}_{\text{correct}}(m, c_k)$ and recompute TUHR on this fixed population.

Table E.3: TUHR (%) on the shared-correct subset for LLaMA-3.1-8B-it. Δ is the change from Base.

	C_1		C_2		C_3		C_4		C_5		C_6	
Method	TUHR	Δ	TUHR	Δ	TUHR	Δ	TUHR	Δ	TUHR	Δ	TUHR	Δ
Base	12.4	—	13.1	—	12.8	—	24.6	—	26.2	—	13.0	—
Relign	12.6	+0.2	13.4	+0.3	13.0	+0.2	24.8	+0.2	26.6	+0.4	13.2	+0.2
PALADIN	12.2	−0.2	12.9	−0.2	12.6	−0.2	24.4	−0.2	25.8	−0.4	12.8	−0.2
Gorilla	12.8	+0.4	13.6	+0.5	13.4	+0.6	25.2	+0.6	27.0	+0.8	13.4	+0.4
LinSteer	12.6	+0.2	13.2	+0.1	13.0	+0.2	24.8	+0.2	26.4	+0.2	13.0	0.0
PRISMS	12.4	0.0	13.0	−0.1	12.6	−0.2	24.4	−0.2	26.0	−0.2	12.8	−0.2

Table E.4: TUHR (%) on the shared-correct subset for Qwen-3.5-9B Δ is the change from Base.

	C_1		C_2		C_3		C_4		C_5		C_6	
Method	TUHR	Δ	TUHR	Δ	TUHR	Δ	TUHR	Δ	TUHR	Δ	TUHR	Δ
Base	11.8	—	12.6	—	12.2	—	25.0	—	27.4	—	12.4	—
Relign	12.0	+0.2	12.8	+0.2	12.4	+0.2	25.4	+0.4	27.6	+0.2	12.6	+0.2
PALADIN	11.6	−0.2	12.4	−0.2	12.0	−0.2	24.8	−0.2	27.0	−0.4	12.2	−0.2
Gorilla	12.2	+0.4	13.2	+0.6	12.8	+0.6	25.6	+0.6	28.0	+0.6	12.8	+0.4
LinSteer	11.8	0.0	12.6	0.0	12.4	+0.2	25.2	+0.2	27.4	0.0	12.4	0.0
PRISMS	11.6	−0.2	12.4	−0.2	12.0	−0.2	24.8	−0.2	27.0	−0.4	12.2	−0.2

On the shared-correct subset, all TUHR changes remain within ± 0.8 pp, confirming that the limited sensitivity is not driven by denominator shift.

F Additional Behavioral Analysis

F.1 Intrinsic Prior Strength

For tool selection, the runtime tool list is removed and the model is sampled ten times. The selection prior strength is defined as the frequency of the most frequently generated tool:

$$S^{\text{sel}}_{\text{prior}}(q) = \max_t \frac{1}{K} \sum_{j=1}^{K} \mathbf{1}[t_j = t]. \tag{F.1}$$

For tool usage, the tool name is provided while its schema is omitted. Let P_i denote the argument-name

Table F.1: Mean intrinsic prior strengths on the 300-query behavioral set.

Model	Selection prior	Usage prior
LLaMA-8B	0.69	0.63
LLaMA-70B	0.75	0.68
Qwen-9B	0.71	0.65
Qwen-27B	0.78	0.71
Mistral-7B	0.61	0.57
Mixtral-8x7B	0.66	0.60

Table F.2: TSHR (%) under selection conflict across prior-strength quintiles.

Model	Q_1	Q_2	Q_3	Q_4	Q_5
LLaMA-8B	18.3	35.0	52.4	67.8	79.0
LLaMA-70B	11.2	23.8	37.0	49.4	63.2
Qwen-9B	16.8	34.0	50.6	65.0	75.4
Qwen-27B	9.5	21.6	35.2	48.0	60.4
Mistral-7B	26.5	46.8	63.0	78.4	89.8
Mixtral-8x7B	15.2	30.4	44.8	58.6	71.2

Table F.3: TSHR (%) under aligned condition across selection-prior quintiles.

Model	Q_1	Q_2	Q_3	Q_4	Q_5
LLaMA-8B	5.0	6.2	4.8	5.5	5.8
LLaMA-70B	3.0	2.2	2.8	3.5	2.5
Qwen-9B	4.2	5.0	3.5	4.8	4.5
Qwen-27B	2.2	1.5	2.8	2.0	1.8
Mistral-7B	8.5	7.2	8.0	9.2	7.8
Mixtral-8x7B	5.5	4.8	5.2	6.0	5.0

Table F.5: ANHR (%) under argument-name conflict across usage-prior quintiles.

Model	Q_1	Q_2	Q_3	Q_4	Q_5
LLaMA-8B	8.4	15.2	21.5	28.8	35.0
LLaMA-70B	5.5	10.2	14.8	19.6	25.3
Qwen-9B	7.0	13.5	19.2	25.5	32.2
Qwen-27B	4.8	8.8	13.0	17.5	22.0
Mistral-7B	13.0	21.5	30.2	39.5	50.5
Mixtral-8x7B	8.2	15.0	22.5	29.8	37.5

Table F.6: ANHR (%) under aligned condition across usage-prior quintiles.

Model	Q_1	Q_2	Q_3	Q_4	Q_5
LLaMA-8B	3.8	3.2	4.0	3.5	3.5
LLaMA-70B	2.2	1.8	2.5	2.0	2.2
Qwen-9B	3.2	2.8	3.5	2.5	3.0
Qwen-27B	1.8	1.5	1.2	2.0	1.5
Mistral-7B	5.8	5.2	6.0	5.5	5.5
Mixtral-8x7B	4.0	3.5	4.2	3.8	4.0

set generated in the i-th sample and P_{mode} the most frequently generated argument-name set; we compute

$$S^{\text{use}}_{\text{prior}}(q) = \frac{1}{K}\sum_{i=1}^{K} \text{Jaccard}(P_i, P_{\text{mode}}). \tag{F.2}$$

Across the six models, the mean selection prior ranges from 0.61 to 0.78 and the mean usage prior from 0.57 to 0.71, showing that models repeatedly favor similar tools and argument structures even without the corresponding runtime configuration.

F.2 Tendencies and Hallucination

To further examine the relationship between intrinsic tendency and hallucination, we divide queries into five equal-sized groups by selection-prior strength and measure TSHR under B_{tool}.

Table F.4: Spearman correlations between prior strength and hallucination under conflict.

Model	ρ_{sel}	ρ_{use}	ρ_{AN}	ρ_{AV}
LLaMA-8B	0.68	0.52	0.58	0.36
LLaMA-70B	0.61	0.44	0.52	0.32
Qwen-9B	0.70	0.53	0.60	0.37
Qwen-27B	0.58	0.42	0.50	0.30
Mistral-7B	0.72	0.56	0.64	0.38
Mixtral-8x7B	0.64	0.47	0.55	0.33

As shown in Table F.2, the hallucination rate increases monotonically with prior strength across all six models, further supporting the relationship between stronger intrinsic tendencies and increased hallucination under conflicting runtime evidence. For comparison, Table F.3 reports TSHR under the aligned condition (A): hallucination rates remain uniformly low regardless of prior-strength quintile, confirming that the monotonic increase is specific to the conflict condition. Note that with 300 queries divided into five quintiles ($n{\approx}60$ per group), individual quintile rates carry a per-seed granularity of roughly 1.7 pp; averaging over three random seeds stabilizes these estimates. The Spearman correlations in Table F.4 are computed at the individual-query level, correlating each query's prior strength with its binary hallucination indicator across the full 300-query set (averaged over three seeds), rather than over the five quintile-level rates. Tied ranks are handled with the average-rank convention.

We additionally decompose usage-level hallucination into argument-name and argument-value errors.

As shown in Table F.4, argument-name hallucination shows a substantially stronger association with the intrinsic usage prior than argument-value hallucination. This motivates the subsequent internal analysis to focus on tool-selection and argument-name conflicts.

Table F.7: AVHR (%) under argument-value conflict across usage-prior quintiles.

Model	Q_1	Q_2	Q_3	Q_4	Q_5
LLaMA-8B	7.8	9.5	11.8	13.5	16.2
LLaMA-70B	5.2	6.8	8.0	9.5	11.5
Qwen-9B	6.5	8.2	10.5	12.5	15.0
Qwen-27B	4.2	5.5	7.0	8.2	10.0
Mistral-7B	12.0	14.5	17.5	20.0	23.5
Mixtral-8x7B	8.5	10.5	13.0	15.0	17.8

Table F.8: AVHR (%) under aligned condition across usage-prior quintiles.

Model	Q_1	Q_2	Q_3	Q_4	Q_5
LLaMA-8B	4.8	4.2	5.0	4.5	4.5
LLaMA-70B	2.5	3.2	2.8	3.0	2.5
Qwen-9B	3.8	4.2	3.5	4.0	3.8
Qwen-27B	2.0	2.5	1.8	2.2	2.5
Mistral-7B	7.5	7.2	8.2	7.0	7.8
Mixtral-8x7B	5.5	6.0	5.2	5.8	5.5

Tables F.5 – F.8 provide the corresponding quintile breakdowns for argument-name and argument-value hallucination. Under conflict ($B_{\arg}$), ANHR increases monotonically with usage-prior strength (Table F.5), while AVHR shows a weaker but still positive trend (Table F.7). Under the aligned condition, both ANHR and AVHR remain flat across quintiles (Tables F.6 and F.8), mirroring the pattern observed at the selection level.

G Additional Internal Representation Analysis

G.1 Probe Training and Layer Selection

For each decoder layer l, we extract the hidden state of the last input token,

$$\mathbf{h}^{(l)}(x) \in \mathbb{R}^d,$$

and train an L_2-regularized logistic-regression probe:

$$p_r^{(l)}(x) = \sigma\left((\mathbf{w}_r^{(l)})^\top \mathbf{h}^{(l)}(x) + b_r^{(l)}\right), \qquad r \in \{\text{sel}, \arg\}. \tag{G.1}$$

The regularization coefficient is selected on the validation split from

$$C \in \{10^{-2}, 10^{-1}, 1, 10\}.$$

For each seed, we identify the layer with the highest validation accuracy. When multiple layers fall within 0.2 percentage points of the best accuracy, we select the earliest layer to avoid unnecessarily late intervention. The representative layer l_r^* is the median selected layer across the five seeds, while the observed range reports the corresponding variation across splits.

G.2 Extended Layerwise Probing Results

Across model families and scales, selection conflict becomes most distinguishable in the middle-to-late layers, while argument-name conflict generally peaks later. Selection probes also consistently achieve higher accuracy and hallucination-prediction AUROC than argument-name probes.

G.3 Predicting Hallucination within Conflicting Inputs

To ensure that the probes do not merely distinguish different input templates, we evaluate whether the probe score predicts hallucination within conflicting inputs alone.

For each conflict type,

$$P^r_{\text{conflict}}(x) = p_r^{(l_r^*)}(x) \tag{G.2}$$

serves as the conflict score, by which conflicting examples are divided into five equally sized groups.

The monotonic increase from Q_1 to Q_5 in Table G.1 shows that examples with stronger internal conflict signals are substantially more likely to hallucinate. The effect is observed for every evaluated backbone and is consistently stronger for tool selection than for argument naming.

The corresponding AUROC is computed by using $P^r_{\text{conflict}}(x)$ to rank hallucinating and non-hallucinating examples within the conflicting condition. The score is therefore interpreted as an internal conflict signal rather than a calibrated hallucination probability.

Table G.1: Layerwise probing across six backbones. Accuracy is mean ± standard deviation over five splits; prediction is evaluated within conflicting inputs.

Model	Conflict	Peak Layer (range)	Acc.↑	$Q_1 \rightarrow Q_5$	AUROC ↑
LLaMA-3.1-8B	Selection	20/32 (19–21)	87.2 ± 0.6	19.6 → 81.2	0.84
	Arg. name	23/32 (22–24)	79.4 ± 0.8	8.4 → 35.6	0.75
LLaMA-3.1-70B	Selection	51/80 (50–52)	88.8 ± 0.5	11.8 → 63.2	0.88
	Arg. name	58/80 (57–59)	80.6 ± 0.6	4.2 → 28.4	0.78
Qwen-3.5-9B	Selection	19/32 (17–21)	85.6 ± 0.7	17.8 → 79.4	0.81
	Arg. name	23/32 (22–25)	76.8 ± 0.9	7.2 → 33.8	0.74
Qwen-3.5-27B	Selection	38/64 (37–40)	87.2 ± 0.5	10.2 → 61.6	0.85
	Arg. name	42/64 (41–44)	79.2 ± 0.7	3.8 → 26.4	0.75
Mistral-7B	Selection	21/32 (20–22)	86.0 ± 0.8	29.4 → 91.8	0.80
	Arg. name	23/32 (22–24)	77.6 ± 0.9	16.2 → 46.4	0.73
Mixtral-8x7B	Selection	20/32 (19–21)	86.8 ± 0.7	15.4 → 73.6	0.84
	Arg. name	22/32 (21–23)	78.2 ± 0.8	6.2 → 30.4	0.71

G.4 Control Experiments

We perform two controls to test whether the observed probe accuracy can be explained by trivial dataset or template differences.

First, conflict labels are randomly permuted while preserving the train/validation/test split. Probe accuracy falls to approximately chance level.

Second, we evaluate a query-only representation in which the runtime tool configuration is removed. Without the runtime evidence required to establish alignment or conflict, probe accuracy drops to 51.0%–53.2%—near chance (the residual deviation from 50% reflects slight class imbalance from incomplete query pairing, as not every source query yields a valid variant for every condition).

Table G.2: Probe controls (% accuracy) on the two main backbones.

Model	Conflict type	Full input	Random labels	Query only
LLaMA-3.1-8B	Selection	87.2	50.8	53.2
	Arg. name	79.4	49.6	51.0
Qwen-3.5-9B	Selection	85.6	50.4	52.8
	Arg. name	76.8	50.1	51.5

These controls indicate that the conflict probes rely on the interaction between intrinsic tendency and runtime tool information rather than on query identity alone.

G.5 Disentangling Conflict from Configuration Novelty

Section 4.2 notes that the B_{tool} condition necessarily introduces renamed tools and rewritten descriptions, raising the possibility that probes detect configuration novelty (out-of-distribution surface forms) rather than tendency conflict per se. We conduct two controls to disentangle these factors.

Novelty-matched control. We construct a third condition, B_{novel}, in which tool names and descriptions are rewritten to the same degree as B_{tool} but the replacement *aligns* with the model's elicited preference, so no tendency conflict exists despite equivalent surface-form novelty. Concretely, for a weather query whose elicited preference (per § 4.1) is `fetch_weather_info`, condition A presents the tool under this preferred name. B_{tool} renames the tool to a semantically unrelated name such as `geo_obs_historical` (novel and conflicting), whereas B_{novel} renames it to a semantically equivalent paraphrase such as `retrieve_weather_data` (equally novel in surface form, but still recognizable as a weather-data tool, so the model's tendency is satisfied without conflict). Both B_{tool} and B_{novel} are matched on surface-form edit distance from A. If probes primarily detect novelty, B_{novel} should receive conflict scores comparable to B_{tool}.

As shown in Table G.3, B_{novel} receives substantially lower conflict scores than B_{tool} (0.41 vs. 0.81 for LLaMA; 0.38 vs. 0.79 for Qwen), and classification accuracy drops from 87%/86% to 62%/60%,

Table G.3: Novelty-matched probe control. Mean conflict scores (0 – 1) and classification accuracy (%) for A vs. B_{tool} (conflict detection) and A vs. B_{novel} (novelty detection without conflict).

Condition	LLaMA-3.1-8B			Qwen-3.5-9B		
	Score(A)	Score(B)	Acc.	Score(A)	Score(B)	Acc.
A vs. B_{tool}	0.24	0.81	87.2	0.22	0.79	85.6
A vs. B_{novel}	0.24	0.41	61.8	0.22	0.38	59.6

substantially above chance but far below the full-conflict condition. This indicates that surface-form novelty contributes to but does not dominate the probe signal.

Hallucination prediction within novelty-matched subsets. We further partition the B_{tool} examples into novelty-matched terciles (based on surface-form edit distance between original and renamed tools) and measure whether conflict scores still predict hallucination within each tercile. The monotonic relationship between conflict score and hallucination rate is preserved within all terciles (Spearman ρ=0.51 – 0.67), confirming that the predictive signal is not reducible to novelty.

While these controls substantially narrow the alternative explanation, we acknowledge that fully ruling out all forms of distributional shift as a confound would require intervening on tendency strength while holding the configuration fixed, which is not straightforward in a pretrained model.

G.6 Probe Freezing and § 5 Data Isolation

After layer selection, the final probe for each backbone is fitted using the combined training and validation portion of the probing corpus while keeping the selected representative layer fixed. These probes are then frozen before § 5 experiments.

The representative conflict regions are listed in Table H.1.

All § 5 evaluation examples are kept disjoint from the 1,200-query probing corpus. For Glaive Function Calling V2, examples used for § 4 probing are explicitly excluded from subsequent evaluation. For BFCL and Seal-Tools, overlap is removed using source identifiers together with normalized query and tool-schema matching to ensure no leakage.

Consequently, the conflict probes, intervention regions, and hyperparameter selection are fixed before evaluating EscapeGuard, preventing § 5 test examples from influencing the internal conflict detector.

H Additional Experiments for **EscapeGuard**

H.1 Model-Specific Probes and Intervention Regions

EscapeGuard uses the frozen conflict probes obtained with the § 4 procedure. The representative intervention regions are centered on the model-specific conflict layers identified by layerwise probing.

The probes are frozen before § 5 evaluation. Complete probe accuracy, peak-layer ranges, conflict-quintile analysis, and hallucination-prediction AUROC are reported in Appendix G.

Table H.1: Frozen model-specific intervention regions (decoder layers).

Model	Selection Region	Argument Region
LLaMA-8B	19–21	22–24
LLaMA-70B	50–52	57–59
Qwen-9B	17–21	22–25
Qwen-27B	37–40	41–44
Mistral-7B	20–22	22–24
Mixtral-8x7B	19–21	21–23

H.2 Metric Applicability and Data Isolation

We apply the hallucination metrics from § 3–4 according to the annotations available in each benchmark. BFCL, API-Bank, Seal-Tools, and APIBench provide complete reference tool calls and argument schemas, supporting all four metrics. RelyToolBench supports selection- and usage-level evaluation but does not distinguish argument-name from argument-value errors, so we report TSHR and TUHR only. MetaTool focuses on tool

selection and is evaluated with TSHR only. Table H.2 lists the evaluation pool size after applying our single-turn protocol and removing all overlap with the § 4 probing corpus; actual test-set sizes drawn from each pool are reported alongside the corresponding results tables.

All § 5 test examples are disjoint from the data used for probe training, layer localization, and hyperparameter selection. Glaive evaluation excludes all examples used in the § 4 probing corpus. For BFCL and Seal-Tools, overlap is removed at the query and normalized tool-schema levels.

Table H.2: Metric applicability and evaluation role.

Dataset	Eval. pool	TSHR	TUHR	ANHR	AVHR	Role
BFCL V3	∼3,241	✓	✓	✓	✓	Main
Seal-Tools	∼3,776	✓	✓	✓	✓	Main
RelyToolBench	∼958	✓	✓	–	–	Main
API-Bank	∼214	✓	✓	✓	✓	Main
MetaTool	∼20,630	✓	–	–	–	Main
APIBench	∼2,505	✓	✓	✓	✓	Main
Glaive V2	∼3,500	✓	✓	✓	✓	Suppl.

H.3 Full Usage-Level Results

Table H.3: TUHR (%) ↓ by benchmark, Base → EscapeGuard.

Model	BFCL	Seal-Tools	RelyToolBench	API-Bank	APIBench	Avg.
LLaMA-8B	14.6 → 8.2	17.2 → 9.2	15.8 → 10.7	12.3 → 5.8	16.1 → 9.1	**15.2 → 8.6**
LLaMA-70B	10.4 → 6.8	12.0 → 7.5	11.3 → 7.6	8.8 → 5.4	11.5 → 7.2	**10.8 → 6.9**
Qwen-9B	11.9 → 6.3	14.1 → 7.2	13.1 → 7.9	10.1 → 4.8	13.3 → 7.4	**12.5 → 6.7**
Qwen-27B	7.7 → 4.9	9.2 → 5.7	8.5 → 6.0	6.5 → 4.2	8.6 → 5.7	**8.1 → 5.3**
Mistral-7B	25.8 → 18.4	28.6 → 19.6	26.4 → 21.0	23.2 → 16.2	27.0 → 19.8	**26.2 → 19.0**
Mixtral-8x7B	20.4 → 12.6	23.7 → 14.3	21.6 → 13.4	18.6 → 10.6	22.2 → 13.6	**21.3 → 12.9**

Table H.4: ANHR (%) ↓ by benchmark, Base → EscapeGuard.

Model	BFCL	Seal-Tools	API-Bank	APIBench	Avg.
LLaMA-8B	8.3 → 3.7	9.9 → 4.0	7.1 → 2.8	9.5 → 3.5	**8.7 → 3.5**
LLaMA-70B	6.0 → 2.4	7.3 → 2.9	5.2 → 2.1	7.1 → 3.0	**6.4 → 2.6**
Qwen-9B	7.0 → 2.5	8.2 → 2.8	6.0 → 2.0	8.0 → 2.3	**7.3 → 2.4**
Qwen-27B	4.0 → 2.7	4.8 → 3.4	3.5 → 2.3	4.9 → 3.6	**4.3 → 3.0**
Mistral-7B	18.4 → 14.4	20.2 → 14.8	17.0 → 13.4	19.6 → 14.6	**18.8 → 14.3**
Mixtral-8x7B	13.2 → 6.8	14.9 → 7.4	12.0 → 5.6	14.3 → 7.0	**13.6 → 6.7**

Table H.5: AVHR (%) ↓ by benchmark, Base → EscapeGuard.

Model	BFCL	Seal-Tools	API-Bank	APIBench	Avg.
LLaMA-8B	7.2 → 6.7	8.4 → 7.5	6.1 → 5.2	7.5 → 6.6	**7.3 → 6.5**
LLaMA-70B	5.6 → 5.1	6.5 → 5.8	4.9 → 4.4	6.2 → 5.5	**5.8 → 5.2**
Qwen-9B	6.1 → 5.5	7.1 → 6.4	5.2 → 4.7	6.4 → 5.8	**6.2 → 5.6**
Qwen-27B	4.6 → 3.9	5.4 → 4.6	4.0 → 3.4	4.8 → 4.1	**4.7 → 4.0**
Mistral-7B	17.5 → 16.7	19.1 → 18.1	16.3 → 15.5	18.3 → 17.3	**17.8 → 16.9**
Mixtral-8x7B	12.8 → 12.0	14.6 → 13.7	11.4 → 10.6	13.6 → 12.9	**13.1 → 12.3**

H.4 Supplementary Generalization on Glaive

Glaive Function Calling V2 is used only as supplementary evaluation because a separate subset contributes to the § 4 probing corpus. We use a held-out subset with no query or normalized schema overlap with probe-training examples.

These results follow the shared evaluation protocol and provide supplementary evidence for the findings in the main text.

H.5 Full Baseline Comparison on Qwen-3.5-9B

To verify that the comparison in Section 5.3 is not specific to LLaMA-3.1-8B, we repeat the same evaluation on Qwen-3.5-9B.

Table H.6: Supplementary Glaive results (%), Base → EscapeGuard.

Model	TSHR ↓	TUHR ↓	ANHR ↓	AVHR ↓
LLaMA-8B	20.4 → 11.2	15.6 → 8.8	9.1 → 3.7	7.4 → 6.8
LLaMA-70B	11.6 → 6.4	10.6 → 7.0	6.3 → 4.6	5.5 → 4.3
Qwen-9B	17.8 → 8.6	13.1 → 6.5	7.4 → 2.6	6.9 → 6.2
Qwen-27B	9.5 → 5.4	8.4 → 5.1	4.6 → 2.3	3.9 → 3.4
Mistral-7B	31.6 → 21.4	26.4 → 19.2	18.8 → 14.2	18.0 → 17.2
Mixtral-8x7B	24.8 → 14.6	20.2 → 12.4	13.6 → 6.4	12.8 → 11.6

The Qwen results exhibit the same pattern as LLaMA-3.1-8B: EscapeGuard consistently improves TSHR, TUHR, and ANHR, while its advantage on AVHR remains limited.

Table H.7: Comparison with existing methods on Qwen-3.5-9B. Lower is better. **Bold** = best.

Dataset	Metric	Base	Relign	PALADIN	Gorilla	LinSteer	PRISMS	EscapeGuard
BFCL	TSHR/TUHR	17.1/11.9	13.0/10.3	13.8/10.9	12.2/12.5	14.3/11.3	13.5/10.4	**7.3/6.3**
	ANHR/AVHR	7.0/6.1	6.1/5.6	6.9/5.7	7.2/6.3	6.7/6.0	6.4/5.8	**2.5/5.5**
Seal-Tools	TSHR/TUHR	22.4/14.1	17.6/15.3	18.5/14.7	16.7/16.3	19.0/16.0	18.1/15.6	**9.6/7.2**
	ANHR/AVHR	8.2/7.1	9.4/6.5	9.0/6.7	9.6/7.6	10.1/6.9	9.8/6.7	**2.8/6.4**
API-Bank	TSHR/TUHR	15.3/10.1	11.4/11.0	12.1/11.2	10.7/12.3	12.5/11.9	11.8/11.4	**4.6/4.8**
	ANHR/AVHR	6.0/5.2	6.3/4.7	6.4/5.1	7.1/5.3	7.0/5.2	6.5/5.0	**2.0/4.7**
APIBench	TSHR/TUHR	19.6/13.3	15.1/14.6	16.0/14.8	14.4/16.0	16.4/15.6	15.6/15.0	**8.1/7.4**
	ANHR/AVHR	8.0/6.4	8.5/6.1	8.6/6.4	9.5/7.0	9.3/6.5	8.8/6.2	**2.3/5.8**

H.6 Cross-Configuration Performance Evaluation Details

For the C_1–C_6 experiment, all configuration variants originating from the same query are kept together during evaluation and statistical resampling. Table 7 directly reuses the § 3 evaluation set so that the changes in parentheses are comparable with Table 1.

To verify cross-model generality, we extend the evaluation to four additional backbones (LLaMA-3.1-70B, Qwen-3.5-27B, Mistral-7B, Mixtral-8x7B) using C_1, C_2, and C_5 as representative standard, tool-replacement, and mixed-change configurations.

Table H.8: Cross-model configuration performance (%), Base → EscapeGuard. C_1/C_2: TSHR; C_5: OHR.

Model	C_1	C_2	C_5 (OHR)
LLaMA-70B	12.4 → 4.8	36.8 → 5.6	48.6 → 26.4
Qwen-27B	10.8 → 3.6	34.4 → 4.4	46.2 → 24.6
Mistral-7B	34.6 → 16.4	62.8 → 17.8	72.4 → 34.2
Mixtral-8x7B	24.2 → 12.6	50.6 → 13.4	60.8 → 28.6

Across all four models, EscapeGuard reduces hallucination under every configuration and brings C_2 to a level comparable with C_1.

For paired-query evaluation, both members of each pair are treated as one resampling unit. Net improvement follows the § 3 definition:

$$\text{Net} = \text{Repair} \times (1 - \text{TransferDamage}).$$

Table H.10: Sensitivity to the conflict threshold (%); default $\tau = 0.6$.

τ	TSHR ↓	TUHR ↓	ANHR ↓
0.40	10.0	8.4	3.8
0.50	8.9	7.7	3.2
0.60	**8.3**	**7.3**	**3.0**
0.70	8.8	8.0	3.4
0.80	11.0	9.8	4.7

Table H.11: Sensitivity to attention bias (%); default $\lambda = 1.8$.

λ	TSHR ↓	TUHR ↓	ANHR ↓
0.60	12.0	9.7	4.8
1.00	9.8	8.1	3.7
1.40	8.9	7.6	3.4
1.80	**8.3**	**7.3**	**3.0**
2.40	8.7	7.5	3.2
3.00	9.5	8.0	3.6

H.7 Mechanism Validation

We further examine whether EscapeGuard increases reliance on the current tool configuration in the direction predicted by § 4. The prior elicited without tool configuration input should remain unchanged because intervention is gated on the conflict score, which falls below the activation threshold τ when tool configuration is absent; in contrast, unconditional steering methods such as LinSteer apply representation shifts regardless of whether tool configuration is present, and consequently alter the prior even outside the conflict condition. PRISMS uses a threshold-based activation mechanism but does not condition on the presence of conflicting tool-configuration input, so it may still shift representations outside the target scenario. Meanwhile, attention to tool-configuration spans and the probability margin of outputs consistent with the current setting should increase for high-conflict inputs.

These results distinguish reweighting toward the current tool configuration from tendency reinforcement: EscapeGuard leaves the intrinsic prior unchanged while increasing reliance on the current tool configuration during generation.

Table H.9: Mechanism validation under conflict. Attention and output margins report Base → EscapeGuard.

Model	Stage	ΔS_{prior}	Context attn.	Output margin
LLaMA-3.1-8B	Selection	+0.004	0.18 → 0.41	0.09 → 0.34
	Arg. name	+0.007	0.14 → 0.26	0.08 → 0.19
Qwen-3.5-9B	Selection	-0.002	0.21 → 0.37	0.12 → 0.28
	Arg. name	+0.006	0.09 → 0.29	0.06 → 0.23

H.8 Hyperparameter Sensitivity

Hyperparameters were selected on held-out development data. We then vary one hyperparameter at a time on the test set while keeping all other hyperparameters fixed, to examine sensitivity around the selected operating point.

$$\tau \in \{0.4, 0.5, 0.6, 0.7, 0.8\}, \qquad \lambda \in \{0.6, 1.0, 1.4, 1.8, 2.4, 3.0\}.$$

We report macro averages across the two main backbones (LLaMA-3.1-8B-Instruct and Qwen-3.5-9B). Both hyperparameters exhibit a broad stable region around the selected values rather than a sharply tuned optimum: for τ, the range 0.5 – 0.7 yields TSHR within 1 pp of the optimum on both models; for λ, the range 1.4 – 2.4 similarly shows stable performance.

H.9 Efficiency and Probe Retraining

Inference overhead. EscapeGuard introduces two sources of overhead relative to standard inference: (1) a forward pass through the frozen conflict probes to compute P^r_{conflict}, and (2) attention logit modification at intervention layers for high-conflict inputs. We measure wall-clock latency with batch size 1, averaging over 500 queries drawn from the evaluation benchmarks (mean input length $\approx$480 tokens, mean generated output $\approx$45 tokens). All 7B/8B models are measured on a single A100-80GB GPU in FP16 precision; LLaMA-3.1-70B and Qwen-3.5-27B use 2×A100-80GB with tensor parallelism. Latency is reported per output token, averaged from input tokenization through output decoding.

The overhead ranges from 4.1% to 5.9% of base inference latency. The probe forward pass accounts for approximately 1.2 – 1.8 ms; the remainder comes from attention modification, which applies only

Table H.12: Inference latency (ms per token) and relative overhead of EscapeGuard.

	7B/9B models			Other models†		
	Base	**+EscapeGuard**	**Overhead**	**Base**	**+EscapeGuard**	**Overhead**
LLaMA	42.3	44.8	+5.9%	156.7	163.2	+4.1%
Qwen	40.1	42.4	+5.7%	148.3	154.6	+4.2%
Mistral	41.6	44.0	+5.8%	—	—	—
Mixtral†	—	—	—	89.4	93.7	+4.8%

†Mixtral-8x7B is a MoE model (46.7B total, 12.9B active params); grouped with 70B/27B models by latency tier. Other models use 2×A100-80GB with tensor parallelism; 7B/8B models use a single A100-80GB.

when the conflict gate activates (approximately 35 – 45% of queries in practice). No additional GPU memory is required beyond the probe parameters (fewer than 0.01% of total model parameters).

Probe retraining. The conflict probes are L_2-regularized logistic-regression models fitted on hidden-state features, requiring no gradient backpropagation through the LLM. Retraining a probe for a new model checkpoint takes approximately 15 – 25 minutes on a single GPU (including feature extraction from the 1,200-query probing corpus and fitting across five splits). This cost is modest relative to the model training or fine-tuning it follows, but it does mean that each model update requires a probe refresh. We note that the probing corpus and split construction (Appendix C.3) can be reused across checkpoints of the same architecture, so only the feature extraction and fitting need to be repeated.

H.10 Ablation Study

We ablate EscapeGuard by removing or replacing one component at a time, averaging TSHR and TUHR over BFCL, Seal-Tools, API-Bank, and APIBench on LLaMA-3.1-8B-Instruct and Qwen-3.5-9B.

Table H.13: Ablation averages over the two main backbones (%). Lower is better.

Variant	TSHR ↓	TUHR ↓	ANHR ↓	$\bar{\mathbf{H}}_{\text{config}}$↓
EscapeGuard	**8.3**	**7.3**	**3.0**	**16.4**
w/o conflict gate	11.2	9.4	4.3	19.6
random context tokens	14.8	11.8	6.2	23.9
non-conflict layers	14.2	11.2	5.8	23.1
fixed steering vector	12.2	9.8	4.7	22.3

H.11 Error-Type Breakdown under C_4/C_5

Section 5.3 notes residual elevated OHR under C_4 and C_5 after applying EscapeGuard. Table H.14 decomposes these OHR values into their TSHR, TUHR, ANHR, and AVHR components to identify the dominant error type.

Table H.14: Error-type decomposition (%) of EscapeGuard OHR under C_4 and C_5.

	LLaMA-3.1-8B					Qwen-3.5-9B				
Config	**OHR**	**TSHR**	**TUHR**	**ANHR**	**AVHR**	**OHR**	**TSHR**	**TUHR**	**ANHR**	**AVHR**
C_4	29.6	7.4	24.0	5.2	20.6	28.4	4.8	24.8	4.8	21.4
C_5	25.4	6.3	20.4	4.6	17.2	23.4	3.5	20.6	4.0	18.0

AVHR accounts for the majority of residual TUHR in both configurations and both models, confirming that the residual OHR is dominated by argument-value errors—the error type least associated with tendency conflict (§ 4).