Actionable Hallucination Detection: Translating Latent Uncertainty into Agentic Critique

Sanidhya Vijayvargiya[*1] and **Rahul Lokesh**[1]

[1]Samsung Research America
sanidhya.v@samsung.com, rahul.lokesh@samsung.com

Abstract

Large Language Models (LLMs) deployed as AI agents frequently exhibit user specification-grounding failures, executing hallucinated, undesired actions to force a resolution rather than expressing uncertainty. Existing detection methods fail to provide actionable, real-time correction as they either do not localize the hallucinations, or incur prohibitive inference latency. We introduce the Latent Critic, a lightweight low-rank adapter (LoRA) that operates concurrently with a frozen base LLM's generation to actively restructure the transformer's residual stream—amplifying latent grounding signals and translating them into localized, natural language feedback within a single sequence. By refining the base model's native uncertainty signals, this manipulation of the latent space enables reliable, granular detection without the overhead of secondary inference loops. Mechanistic analysis via activation patching and layer-wise probing shows that this rank-invariant behavior restructures pre-existing uncertainty geometry into a linearly separable representation that transfers more reliably than base model representations alone. Using tool-calling as an instantiation of granular hallucinations, we validate the detection and downstream improvements enabled by the Latent Critic architecture across Qwen and Llama-based models. Demonstrating superior real-time efficacy, our approach significantly outperforms equivalent-scale fine-tuned external detectors, semantic entropy baselines, and passive internal probes in isolating hallucinations, achieving 0.966 AUROC and $>80\%$ accuracy in localization (e.g., *ungrounded: date*). When deployed in a closed-loop ReAct environment, the Critic acts as a negligible latency guardrail, intercepting hallucinations before execution to prevent undesired actions while simultaneously leveraging this specific localized feedback to enable efficient agent self-correction.

1 Introduction

Human communication is inherently characterized by omission and ambiguity. When an AI agent encounters an incomplete instruction, the optimal behavior is to halt execution, express uncertainty, and seek clarification [Vijayvargiya et al., 2026b, Kim et al., 2024]. However, Large Language Models (LLMs) are heavily optimized for instruction following and next-token prediction which instill a severe task-completion bias [Bai et al., 2022]. In agentic workflows, this bias manifests as a critical failure mode in the form of user specification-grounding failures [Sun et al., 2025]. Unlike standard factual hallucinations, an action may be logical, but remains a hallucination if it was never explicitly or implicitly requested by the user. Rather than flagging missing specifications, models confidently execute these undesired actions, creating concrete, misaligned changes to the environment. This lack of adaptability to ambiguous input limits the reliability of AI agents for deployment in high-stakes settings.

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

Current detection paradigms fail to satisfy the latency, actionability, and generalization requirements of real-time agents. Fine-tuned external LLM-as-a-judge evaluators [Darwish et al., 2025] and multiple-sampling techniques like semantic entropy [Ji et al., 2023, Kossen et al., 2024] operate on surface text, introducing prohibitive inference delays. Furthermore, because models confidently generate these actions, uncertainty-based sampling methods often yield low entropy for these errors, causing detection to collapse. In contrast, probing techniques show that a model's internal representations encode signals of its own uncertainty [Orgad et al., 2025]. However, passive probing typically trains layer-wise classifiers that yield scalar confidence scores, which lack the actionable localization required to debug an action.

This actionability gap raises a fundamental question: *can we translate internal uncertainty into localized feedback without adding prohibitive latency?*

To answer this, we introduce the *Latent Critic*, a Parameter-Efficient Fine-Tuning (PEFT) architecture designed to operate as a minimal-latency mechanistic guardrail. We scope our design and investigation specifically to tool-calling. Unlike open-ended text generation, where the boundaries of a hallucination are often diffuse and subjective, tool calls are executable actions that require exact, parameter-level specification grounding. The structured domain of tool calling acts as a testbed for isolating grounding errors and evaluating fine-grained uncertainty extraction. By attaching a specialized LoRA adapter to a frozen base model, we preserve its native agentic capabilities while leveraging the LLM backbone to verbalize the exact localized hallucination (e.g., `ungrounded: date`) within the same generation sequence (Figure 1). Through a masked diagnostic objective, the Critic restructures the native representation of specification grounding, amplifying it into a linearly separable geometry that transfers more reliably across distribution shifts than passive probes.

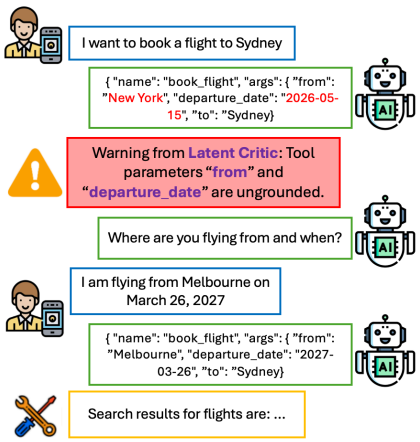


Figure 1: The closed-loop agentic architecture. When the frozen base model generates a hallucinated parameter, the Latent Critic concurrently extracts the specification-grounding failure, blocking execution and feeding the localized error back as a zero-shot environmental observation.

By embedding this detection mechanism directly into the generation process, the Latent Critic bridges the gap between mechanistic representation and practical agent safety. Evaluating this architecture across Qwen and Llama-based models, we demonstrate that exposing and restructuring this latent geometry provides a highly reliable, minimal-latency guardrail that remains the strongest detector across distribution shifts and enables efficient agent self-correction. We structure our investigation around three core Research Questions (RQs):

- **RQ1 (Detection):** Can modifying the residual stream yield superior hallucination localization?
- **RQ2 (Mechanism):** How does the adapter restructure the model's internal geometry to translate uncertainty into actionable feedback?
- **RQ3 (Deployment):** Can agents leverage this specific, localized feedback for zero-shot self-correction, easing the safety-productivity tradeoff?

2 Experimental Setup

To rigorously evaluate real-time hallucination detection, we isolate specification-grounding failures from general capability errors and establish a method for generating and labeling tool-calling trajectories at scale.

2.1 Defining Tool-Calling Hallucinations

Within an agentic tool-calling scope, we evaluate three primary behavioral outcomes:
(1) Correct & Grounded (`ok`)**:** The correct tool is invoked with all parameters grounded in the contextual history.
(2) Wrong Tool (`wrong_tool`)**:** A tool unsuitable for the user's goals is invoked (a policy error).

(3) Ungrounded Parameters (`ungrounded`): The correct tool is invoked, but the model fabricates parameters never specified in the context (a specification-grounding error).

We explicitly exclude capability limitations such as omitted parameters (which occur in $<0.2\%$ of our empirical rollouts) and syntax errors. To illustrate the latter: if a user requests an action for "tomorrow" and the agent generates the literal string `"tomorrow"` instead of the required schema format (e.g., `"YYYY-MM-DD"`), the model possesses the correct contextual grounding but lacks structural compliance.

Crucially, our definition of hallucination in this domain focuses entirely on a lack of support in the user's conversational context. Unlike factual hallucinations, which are evaluated against an external ground truth, specification grounding is evaluated strictly against the dialogue history. A generated parameter can be highly plausible or even factually correct (e.g., guessing a valid date or a commonly used setting), but it remains a hallucination if the user never provided it, explicitly or implicitly.

2.2 Trajectory Generation

To generate data characterized by naturalistic ambiguity, we construct a dynamic environment consisting of a base agent and a simulated user [Patil et al., 2025, Suri et al., 2026].

Datasets: Many tool-calling benchmarks feature either simplified schemas that fail to elicit hallucinations at scale, or multi-hop tasks that are prohibitively complex for the evaluated models. To reliably trigger hallucinations, we select base scenarios from SLM agent literature [Vijayvargiya and Lokesh, 2025], providing tractable user intents with highly detailed tool schemas. We utilize 5,000 scenarios for training and a set of 500 tasks with unseen tools for evaluation (ID). We also utilize 200 scenarios from ToolAlpaca [Tang et al., 2023]—which consists of real-world APIs spanning 50 diverse categories—to evaluate generalization to out-of-distribution (OOD) tasks. The generated tool calls consist of values ranging from single tokens to multiple sentences to evaluate the robustness of the Critic, while the key-value structure allows for precise localization label generation.

Simulated User & Binary Mask: Powered by Qwen3.5-122B [Qwen Team, 2026], the user incrementally reveals an underlying goal through structurally varied, ambiguous requests. To programmatically track this, the user maintains a hidden *Completed Specifications List (CSL)*—a binary mask over the ground truth tool calls that tracks whether the required information for a parameter in the GT tool calls has been provided (`1`) or withheld (`0`).

To illustrate, consider a GT target of `{"name": "book_flight", "args": {"from": "Melbourne", "to": "Sydney", "date": "2027-03-26"}}`. If the user's first turn is "I want to fly to Sydney," the CSL mask initializes as `{"from": 0, "to": 1, "date": 0}`. If the agent then asks for the departure city and the user replies "Melbourne," the mask updates in Turn 2 to `{"from": 1, "to": 1, "date": 0}`.

During the agent's turn, if it outputs a tool call with a parameter corresponding to a `0`, the generation is labeled as an `ungrounded` hallucination. Tools entirely absent from the underlying goal are labeled `wrong_tool`. The simulated user updates the CSL incrementally at each turn, and the initial programmatic labels with the Qwen 3.5 122B model achieve an 89% agreement rate with manual human evaluation. As our setting studies specification grounding—whether information is present in the dialogue—rather than conversational naturalness, simulated users provide controlled, balanced interactions that are logistically prohibitive to collect at scale with human subjects [Vijayvargiya et al., 2026b, Suri et al., 2026]. To ensure rigorous benchmarking, all evaluation trajectories (both ID and OOD) undergo verification by a human annotator, and all reported metrics utilize these corrected, human-verified labels. Failure cases in the programmatic pipeline predominantly included errors in implicit specification tracking or formatting by the simulator model. This pipeline circumvents expensive human labeling for the training set, scales reliably, and accurately localizes exact hallucinated parameters.

Using our pipeline, we execute rollouts with Qwen3-4B and Llama-xLAM-2-8B [Zhang et al., 2024] to ensure our paradigm generalizes across distinct architectures.

3 RQ1: Actionable Hallucination Detection via the Latent Critic

Existing internal probing techniques yield passive scalar outputs that lack the verbalized localization required for agentic self-correction. To address RQ1, we introduce the Latent Critic designed to

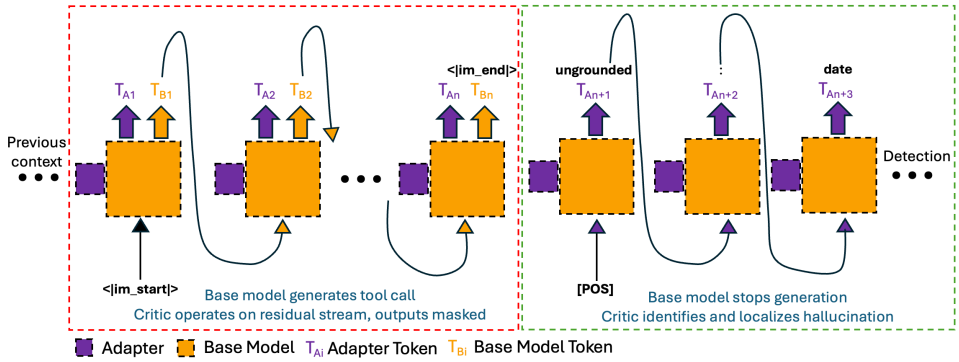


Figure 2: Concurrent extraction architecture. The adapter processes the base model's hidden states during tool call generation and projects the accumulated uncertainty upon generating `[POS]` token.

Method	In-Distribution (ID)			Out-of-Distribution (OOD)		
	F1	AUROC	AUPRC	F1	AUROC	AUPRC
Token Entropy	0.668 ± 0.031	0.674 ± 0.035	0.626 ± 0.050	0.370 ± 0.032	0.485 ± 0.039	0.222 ± 0.031
Semantic Entropy	0.623 ± 0.031	0.639 ± 0.036	0.619 ± 0.049	0.369 ± 0.032	0.460 ± 0.041	0.220 ± 0.032
SEP (Kossen et al.)	0.623 ± 0.031	0.548 ± 0.038	0.482 ± 0.045	0.372 ± 0.038	0.499 ± 0.039	0.235 ± 0.035
Linear Probe	0.697 ± 0.031	0.782 ± 0.024	0.760 ± 0.033	0.383 ± 0.027	0.616 ± 0.029	0.278 ± 0.032
External Judge	0.695 ± 0.041	0.915 ± 0.016	0.864 ± 0.029	0.647 ± 0.021	0.884 ± 0.020	0.741 ± 0.041
Latent Critic (ours)	$\mathbf{0.870 \pm 0.025}$	$\mathbf{0.966 \pm 0.010}$	$\mathbf{0.924 \pm 0.027}$	$\mathbf{0.670 \pm 0.033}$	$\mathbf{0.925 \pm 0.015}$	$\mathbf{0.815 \pm 0.034}$

Table 1: Hallucination Detection Performance on **Qwen3-4B**. Positive class = `ungrounded`. Critic and Judge evaluated one-vs-rest on the full 3-way set (ID N=303: `ok` 41.6%, `ug` 35.0%, `wt` 23.4%; OOD N=522: `ok` 76.2%, `ug` 20.7%, `wt` 3.1%). Probe and Entropy family evaluated strictly on the binary `ok`-vs-`ungrounded` subset. $\pm$ denotes one standard deviation over 10,000 bootstrap resamples. The Latent Critic achieves peak performance across all threshold-independent metrics.

concurrently extract internal uncertainty and output the precise semantic location of a hallucination (e.g., specific keys).

3.1 Latent Critic Architecture

The detection module consists of a Low-Rank Adaptation (LoRA) module [Hu et al., 2021] executed concurrently with the frozen base agent via multi-adapter serving. Because the adapter's weights are initialized at zero, its initial forward pass mirrors the base model's exact hidden states. During the agentic loop, the frozen base model autoregressively generates the tool call syntax. Concurrently, the adapter receives the same input tokens but learns to modify hidden states and amplify uncertainty signals, while its outputs are disregarded in this phase (Figure 2).

Upon completion of the tool call, a trigger token (`[POS]`) is appended, acting as a localized attention sink. This token establishes a structural boundary, signaling that the Critic must now verbalize the detection results by aggregating uncertainty from the full tool-call context and generating a structured classification (e.g., `ok`, `wrong_tool`, or `ungrounded: [parameter_name]`). To ensure the adapter acts strictly as an uncertainty extractor, we modify the Supervised Fine-Tuning (SFT) objective. During loss calculation, the base tool-call generation is masked and the gradients are computed only on the `[POS]` token and the subsequent label. It is important to note that during inference, the adapter actively processes inputs while the base model generates the tool call, however, its outputs at this stage are disregarded. This draws inspiration from pause tokens [Goyal et al., 2024]. Yet, while prior work utilizes dummy tokens to inject extra computational steps during text generation, the Critic's steps during the base model's tool call generation allow for uncertainty accumulation.

3.2 Results

We evaluate the Critic across Qwen3-4B and Llama-xLAM-2-8B against an equivalent-scale External Judge (Qwen 3.5 4B), a passive internal Linear Probe trained on the base model's optimal layer, and multiple uncertainty-based baselines (Token Entropy, Semantic Entropy, and SEP). The External Judge and the Linear Probe are trained with the same data as the Critic, with the Judge using the exact configuration as the Critic (LoRA rank, input prompt, labels, etc.) while the probe and entropy methods are tuned on the ok vs ungrounded classification only (no wrong_tool). We hypothesize that

Method	Llama-xLAM-2-8B	
	ID AUROC	OOD AUROC
Linear Probe	0.833 ± 0.044	0.514 ± 0.030
External Judge	0.903 ± 0.031	0.630 ± 0.030
Latent Critic (ours)	$\mathbf{0.926 \pm 0.029}$	$\mathbf{0.649 \pm 0.028}$

Table 2: Cross-Model Replication on **Llama-xLAM-2-8B**. Positive class = `ungrounded`. The advantage over passive-probing strongly replicates, while the advantage over external evaluators proves model-dependent under shift.

actively restructuring the residual stream allows for granular, robust detection that external text-based evaluators, entropy methods, and passive internal probes miss. Detailed class-wise Precision, Recall, and F1 metrics are available in Appendix C.

In-Distribution Detection: Evaluated on our test set, the Latent Critic demonstrates the strongest overall performance (Table 1). Crucially, these results highlight the fundamental nature of specification-grounding errors. Because models confidently fabricate plausible parameters to complete tasks, standard uncertainty-based sampling methods yield low entropy for these errors. Consequently, methods like Semantic Entropy collapse into a single cluster; when adjusting their AUPRC metrics for baseline prevalence, they sit exactly at the trivial always-positive decision floor, failing to discriminate grounding entirely.

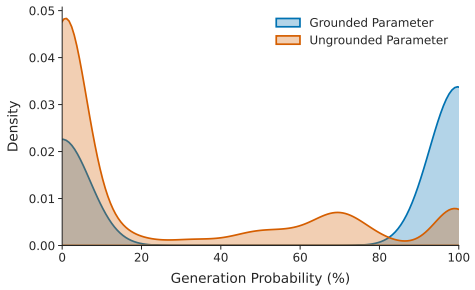
Forced to judge surface-level text, the External Judge struggles with calibration. While it ranks well on Qwen (0.915 ± 0.016 AUROC), it generates poorly (0.695 F1). By directly modifying the hidden states, the Latent Critic isolates the actual geometric signal of specification grounding, achieving 0.966 AUROC on Qwen and localizing errors with $>80\%$ exact parameter-match accuracy. Importantly for deployment where minimizing false blocks is paramount, the Critic successfully catches 38.7% of Qwen's hallucinations at a strict 1% False Positive Rate (FPR), compared to 28.3% for the Judge and just 3.8% for the passive Probe.

Out-of-Distribution Generalization: To test robustness under shift, we evaluate these detectors on unseen scenarios from ToolAlpaca (Table 1). The Linear Probe collapses ($0.782 \rightarrow 0.616$ AUROC), demonstrating that un-restructured internal representations of grounding do not transfer reliably. In contrast, the Latent Critic maintains a strong 0.925 ± 0.015 AUROC. The drop in its OOD F1 score ($0.870 \rightarrow 0.670$) is partly driven by base-rate mechanics, as the OOD prevalence of hallucinations drops from 35.0% to 20.7%. The entropy baselines remain non-discriminative under shift.

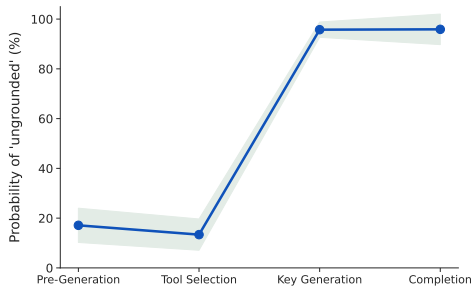
Cross-Model Replication: We also validate this architecture on Llama-xLAM-2-8B (Table 2). The advantage of restructuring over passive probing strongly replicates across both settings (Critic 0.926 AUROC vs. Probe 0.833 ID; Critic 0.649 vs. Probe 0.514 OOD). However, the advantage of internal access over the external judge under shift appears model-dependent. On the xLAM backbone, both the Critic and Judge degrade similarly (0.649 vs. 0.630 AUROC). Ultimately, however, the Latent Critic's improvement over passive internal probing remains statistically significant across both models ($p < 0.001$, measured via McNemar's test [McNemar, 1947]).

Error Analysis: While the metrics represent a significant improvement over existing methods, the Latent Critic's accuracy remains fundamentally bottlenecked by the base model's own capability failures. Because it reads internal states, the Critic is reliable only if the base model internally registers the specification gap. Consequently, detection errors predominantly stem from base model breakdowns that corrupt this geometric signal. For instance, during runaway generation—where the model infinitely loops—the residual stream becomes chaotic, destroying the uncertainty geometry. Similarly, partial grounding dilutes the signal: if a user specifies "2 PM" but the schema requires a full `datetime`, the agent anchors on the grounded time while confidently fabricating the date, occasionally bypassing the Critic.

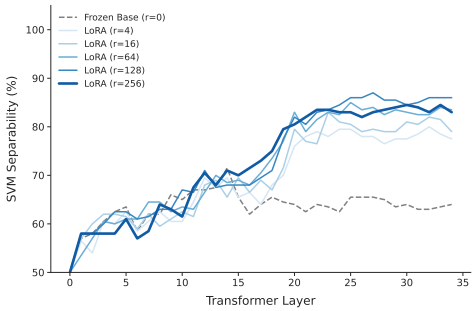
Perceived errors also reveal the Critic's nuanced grounding boundaries. If instructed to "Send an email about X," an agent might create a plausible subject and body. While human judges might accept this, the Critic strictly flags it as `ungrounded`, penalizing unguided, high-stakes hallucination. Yet, for tasks inherently requiring generative expansion (e.g., text summarization), it correctly permits the output. Finally, occasional conflation with the `wrong_tool` class occurs because policy/routing errors can overlap with grounding signals. Overall, misclassifications usually stem from the base



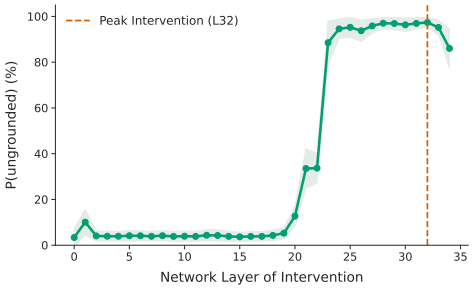
((a)) **Post-Softmax Confidence.** Distribution overlap renders simple logit-thresholding nonviable.



((b)) **Evolution of Uncertainty.** Sharp spike to >95% when the ungrounded key is formulated.



((c)) **Linear Separability.** Latent Critic states undergo sharp, rank-invariant disentanglement around Layer 15.



((d)) **Causal Tracing.** Transplanting a corrupted state in late layers overwrites the network's prediction.

Figure 3: **Mechanistic Analysis of the Latent Critic.** (a) Surface-level output logits fail to reliably separate grounded parameters from fabrications. (b) Latent extraction is reactive, not prescient. (c) The adapter acts as a geometric amplifier, separating entangled base states. (d) Localized activation patching confirms the adapter relies causally on this underlying latent geometry.

model's unreliability in generation rather than extraction flaws, hinting the Critic's real-world utility exceeds its strict metrics.

4 RQ2: Mechanistic Analysis of Latent Uncertainty Extraction

While RQ1 establishes the empirical strengths of the Latent Critic, RQ2 investigates the internal mechanisms enabling this behavior using Qwen3-4B. We analyze how our adapter architecture geometrically maps the uncertainty in the residual stream to the vocabulary space. Specifically, we structure our analysis to establish *why* internal access is necessary (Section 4.1), *when* the specification-grounding signal emerges (Section 4.2), *what* the adapter does to these representations (Section 4.3), and provide *causal evidence* that the adapter relies on this underlying geometry (Section 4.4).

4.1 Confidence in Predictions Does Not Imply Reliability

To establish *why* external judges and entropy methods struggle, we analyze the token-generation probabilities of the base model across 500 scenarios. Standard logit-based confidence scoring fails because models are optimized for task completion. They suffer from poor confidence calibration [Azaria and Mitchell, 2023], regularly outputting confident fabrications (e.g., guessing plausible tokens with >80% confidence) while also generating low-confidence grounded actions. Consequently, the probability density functions of grounded and ungrounded parameters exhibit a strong overlap (Figure 3a), making simple logit-thresholding unreliable. By processing the hidden states directly, the Latent Critic circumvents this constraint. It classifies the ungrounded trajectory with an average confidence of **95.38%**, demonstrating that internal geometry captures significantly more grounding information than is permitted to surface in output logits [Kossen et al., 2024].

Probe Configuration	ID AUROC	OOD AUROC
Linear Probe (Frozen Base States)	0.782	0.616
2-Layer MLP (Frozen Base States)	0.808	0.678
Linear Probe (Latent Critic States)	0.944	0.915
Latent Critic (Full Output)	**0.966**	**0.925**

Table 3: Representation Transfer Analysis. Probes were trained on 16k examples using states extracted from the final generated tool-call token at Layer 21. Training a linear probe on the Latent Critic's restructured states bridges the OOD generalization gap, confirming the adapter exposes a shift-stable geometry rather than simply memorizing a classifier.

4.2 Latent Extraction is Reactive, Not Prescient

To determine exactly *when* the specification-grounding signal arises, we track the Latent Critic's classification confidence across four distinct stages of tool-call construction ($N = 500$). We incrementally append the `[POS]` trigger token: (1) before the tool call, (2) after the tool name was selected, (3) immediately after the ungrounded parameter key was generated, and (4) upon completion of the full payload.

As shown in Figure 3b, the average probability of outputting `ungrounded` remains volatile prior to generating the payload (**19.54%**) and drops slightly when the correct tool name is selected (**16.25%**), as tool selection acts as a coarse-grained commitment that partially resolves the trajectory. However, at the exact sequence step where the ungrounded parameter key is generated, confidence sharply spikes to **95.24%**. This establishes that the internal representation is reactive rather than predictive: the signal denoting a hallucination emerges within the hidden states the instant the model formulates the flawed parameter.

4.3 Rank-Invariant Restructuring of the Residual Stream

To understand *what* the Critic does to these representations, we analyze the linear separability of the hidden states. As established in RQ1, the frozen base model's raw hidden states contain a partial linear representation of grounding In-Distribution, but this direction fails to transfer under shift (0.616 AUROC).

To determine whether this transfer failure is due to limited probe capacity or the underlying representation itself, we trained a 2-layer MLP probe on the frozen states (Table 3). The increased capacity yielded only a modest OOD improvement (0.678 AUROC). However, training a simple linear probe on the *Latent Critic's* adapted hidden states achieved 0.944 ID and 0.915 OOD AUROC. Because the linear probe becomes highly effective once trained on the adapted states, it provides strong evidence that the masked objective explicitly restructures the representation into a shift-stable linear geometry, rather than the adapter memorizing a complex, non-linear classifier from scratch.

Furthermore, as illustrated in Figure 3c, this geometric transformation is rank-invariant. Although indistinguishable from the base model in early syntactic layers (0–14), the adapted representations undergo a large increase in separability starting at Layer 15. Even highly bottlenecked adapters (e.g., $r = 4$) successfully separate the latent space. This implies that the specification-grounding signal occupies a low-dimensional subspace; the adapter only needs to learn a low-rank linear projection to isolate and amplify it.

4.4 Causal Verification via Activation Patching

Finally, to obtain *causal evidence* that the adapter operates strictly on latent geometric representations rather than surface-level text, we perform cross-trajectory activation patching [Meng et al., 2023]. Restricting our intervention to the final `[POS]` token, we execute a forward pass on a corrupted, ungrounded trajectory and cache the hidden states. We then execute a second forward pass on a grounded trajectory. During this second pass, we overwrite the clean `[POS]` token's hidden state at layer L with the corrupted state, allowing the network to complete the remaining forward pass naturally. The context window and prior generated tokens remain grounded. Unlike standard activation patching, which corrupts a single input, we patch across different trajectories to test whether the geometric signal generalizes across contexts.

As shown in Figure 3d, despite the surrounding text indicating a correct tool-call, this localized intervention forces the Latent Critic to output `ungrounded` in **96%** of trajectories. Executing this layer-by-layer reveals a distinct S-curve. Patching early syntactic layers (0–20) yields negligible changes (<5% probability), while patching mid-to-late layers results in a sharp probability inversion peaking at Layer 32. This provides strong causal evidence that the hallucination signal is a geometric feature formed in the later layers, and that the adapter's classification operates independently of surface text. Together, this analysis provides an understanding of how the Latent Critic performs its function, the role of the trigger token, and why it is effective.

5 RQ3: Closed-Loop Agentic Self-Correction

Having established that the Latent Critic accurately extracts parameter-specific errors, we evaluate its utility within an agentic loop. We hypothesize that while simply blocking an action provides safety, projecting that latent uncertainty into targeted natural language (e.g., `ungrounded: date`) functions as an actionable environmental observation, enabling the base agent to self-correct more efficiently than a generic safety block.

5.1 Experimental Setup

To test the real-world applicability of our architecture, we integrate the Latent Critic into a closed-loop ReAct environment. We evaluate 500 in-distribution (ID) trajectories derived from Vijayvargiya and Lokesh [2025], alongside 200 out-of-distribution (OOD) trajectories featuring unseen API schemas and scenarios sampled from ToolAlpaca [Tang et al., 2023]. We compare the base agent loop against two intervention paradigms to isolate the utility of our generative architecture.

1. **Generic Intervention:** Simulates a standard binary classification probe by using the Latent Critic for detection without providing localized feedback. If an error is detected, the environment blocks the tool and returns a generic error: *System Error: Tool execution blocked due to hallucination.* This isolates the marginal utility of the Critic's verbalization.
2. **Specific Intervention:** Our proposed method. The environment blocks the tool and injects the specific, localized verbalization: *System Error: Tool blocked. You hallucinated the parameter: '[PARAM]'.*

To rigorously evaluate the impact of the Latent Critic, we isolate the model's tool-calling capabilities by measuring *Parameter-Level Metrics* across both ID and OOD datasets. Rather than judging a multi-turn trajectory solely as a binary success or failure, we evaluate the exact overlap between the agent's executed tool call arguments and the ground-truth parameters required to fulfill the user's intent. In this formulation, **Precision** acts as a strict proxy for safety: out of all the parameters the agent successfully executed, what percentage were actually grounded and correct? Conversely, **Recall** measures functional task completion: out of all the parameters required by the user's goal, what percentage did the agent successfully execute? The **F1-Score** provides the harmonic mean, representing the agent's overall functional reliability. We also report **Trajectory Success** and the **False Block Rate** (the percentage of correctly grounded calls erroneously blocked). We only consider successfully executed tool calls for parameter analysis; blocked calls are disregarded.

5.2 Results

Because the simulated user deliberately provides ambiguous instructions, the Base Agent exhibits a severe task-completion bias, resulting in ungrounded, low-precision calls attempting to force task completion (Table 4). Applying a **Generic Intervention** acts as a safety boundary, driving ID precision up to 66.1% and OOD precision to 62.3% and blocks these actions. However, because the agent is not given localized feedback, it cannot effectively correct the payload. Due to repeated hallucination blocks, an elevated false-block rate (9.4%), and a lack of actionable feedback, fewer actions can be executed successfully, resulting in a drop in recall and overall parameter F1.

The **Specific Latent Critic** effectively mitigates this tradeoff. On ID tasks, it achieves the highest precision (69.8%) while successfully guiding the agent to recover its recall (54.4%), yielding a peak parameter F1 score of 61.2%. Specific feedback concurrently improves Trajectory Success to 22.1% while reducing false blocks by more than 3× (down to 2.9%) compared to generic feedback.

Under distribution shift (OOD), trajectory success remains close to the frozen base model's ceiling (∼2%) across all methods, making trajectory completion metrics relatively insensitive due to the

Agent Configuration	Parameter-Level Metrics (%)			Trajectory-Level (%)	
	Precision	Recall	F1-Score	Success Rate	False Block
In-Distribution (ID)					
Base Agent (No Intervention)	49.7 ± 2.2	$\mathbf{54.8 \pm 2.6}$	52.1 ± 1.9	16.4 ± 1.8	–
+ Generic Intervention	66.1 ± 3.4	41.9 ± 2.4	51.3 ± 2.2	20.3 ± 1.8	9.4 ± 2.2
+ Specific (Ours)	$\mathbf{69.8 \pm 1.8}$	54.4 ± 2.2	$\mathbf{61.2 \pm 1.8}$	$\mathbf{22.1 \pm 1.9}$	$\mathbf{2.9 \pm 1.0}$
Out-of-Distribution (OOD)					
Base Agent (No Intervention)	52.5 ± 3.4	$\mathbf{26.6 \pm 2.1}$	35.3 ± 2.2	$\mathbf{2.2 \pm 0.8}$	–
+ Generic Intervention	62.3 ± 4.7	21.3 ± 2.8	31.7 ± 2.4	2.0 ± 1.4	15.9 ± 4.1
+ Specific (Ours)	$\mathbf{66.8 \pm 3.0}$	25.3 ± 2.7	$\mathbf{36.7 \pm 1.9}$	2.1 ± 1.5	$\mathbf{13.2 \pm 3.3}$

Table 4: Impact of intervention paradigms on agentic reliability. Generic detection boosts Precision (acting as a safety net) but crashes Recall due to a high false block rate and confused retry loops. The Latent Critic's *Specific* verbalization uniquely cushions this tradeoff, minimizing false blocks and maximizing parameter F1.

base model's broader capability failures on unseen datasets. However, parameter-level recovery still clearly improves (OOD F1: 36.7% vs 35.3%; Precision: 66.8% vs 52.5%), indicating that the Critic continues to identify the correct interventions and enforce safety even when the base agent ultimately lacks the reasoning capacity to fully complete the task.

5.3 Self-Correction

To quantify how specific verbalization affects execution metrics, we isolate the subset of trajectories where an intervention occurred and track the agent's ability to self-correct its policy (e.g., pivoting to ask the user a clarifying question instead of engaging in repeated guessing).

Intervention	In-Distribution (ID)		Out-of-Distribution (OOD)	
	Recovery Rate (%)	Failed Attempts	Recovery Rate (%)	Failed Attempts
Generic	23.9 ± 2.7	1.47 ± 0.14	20.0 ± 5.3	1.41 ± 0.12
Specific (Ours)	$\mathbf{37.0 \pm 1.9}$	$\mathbf{1.06 \pm 0.09}$	$\mathbf{29.3 \pm 4.7}$	$\mathbf{1.00 \pm 0.00}$

Table 5: **Self-Recovery Mechanics.** Providing exact parameter localization improves the agent's recovery rate while significantly reducing wasted inference loops across both distributions.

As detailed in Table 5, actionable specific verbalization improves both the rate of recovery and its efficiency. When faced with a Generic block, the agent often enters a failure loop, repeatedly guessing values for different parameters or trying new hallucinated values for the same parameter (D.1). By injecting the exact diagnostic string, the Latent Critic boosts the successful recovery rate to **37.0%** (ID) and **29.3%** (OOD), representing relative improvements of 54.8% and 46.5%, respectively.

Furthermore, it consistently reduces the average number of failed retry attempts. On novel OOD schemas, the Latent Critic achieves an average of only **1.00 attempt** prior to success. However, it is important to contextualize this efficiency. In many cases, the agent never recovers, signaling a need for dedicated recovery-policy training to fully utilize the diagnostic feedback. Nevertheless, in cases where successful recoveries are possible with the frozen base agent, exact parameter localization significantly reduces the computational overhead and latency associated with retry loops. Finally, as a single-pass extraction module, the Critic itself operates with negligible overhead, adding $<$10 ms per executed tool call in our vLLM deployment environment (and 58 ms in a standalone HuggingFace setup), satisfying the strict real-time requirements of autonomous agents.

6 Related Work

Agentic Tool-Calling and Grounding. Existing evaluation frameworks for AI agents [Li et al., 2023, Qin et al., 2023, Patil et al., 2025] and constrained decoding pipelines [Beurer-Kellner et al., 2024] predominantly emphasize functional correctness and syntactic validity. However, when faced with underspecified instructions, LLMs succumb to a task-completion bias, fabricating missing parameters rather than seeking clarification [Zhang and Choi, 2025, Chi et al., 2024, Vijayvargiya et al., 2026a]. To mitigate these specification-grounding failures, the dominant paradigm relies on post-hoc action correction [Shinn et al., 2023, Gou et al., 2024]. Yet, because these methods operate on surface text, they incur secondary inference latency and are susceptible to hallucinations. Crucially, they

require an erroneous action to be executed and observed to fail before correction begins, whereas our approach intercepts the action pre-execution.

Real-Time Hallucination Detection. Detection methods span from output-level sampling [Manakul et al., 2023, Kuhn et al., 2023]—which is robust but incurs prohibitive latency for real-time deployment—to single-pass internal probes [Kadavath et al., 2022, Azaria and Mitchell, 2023, Su et al., 2024, Kossen et al., 2024]. While probing resolves latency bottlenecks by training lightweight classifiers directly on hidden states, it functions as a passive observer, outputting scalars that are non-actionable for granular self-correction. Furthermore, existing works primarily evaluate factual correctness. In agentic workflows, the challenge shifts to *user-specification alignment* where a guessed parameter (e.g., a valid date) may be factually plausible but remains a hallucination if not grounded in the user's context. While recent work by Healy et al. [2026] demonstrates that internal states can detect policy tool-selection errors, the Latent Critic targets specification-grounding failures, translating latent uncertainty into localized, natural language feedback.

Representation Engineering. A foundational mechanistic result is that transformer residual streams encode geometrically separable representations of truth and uncertainty [Burns et al., 2024, Marks and Tegmark, 2024, Ferrando et al., 2025]. Although interventions like Inference-Time Intervention (ITI) [Li et al., 2024] and activation patching [Meng et al., 2023] prove that these latent directions are causally influential, their implementation typically requires sweeping layers or attention heads. Furthermore, generic PEFT does not reliably improve this linear separability and can disrupt the pre-existing geometric structure [Hu et al., 2026]. Consequently, recent real-time LoRA probes [Obeso et al., 2026] utilize KL-regularization to explicitly prevent the adapter from altering base dynamics, while adversarial approaches [Min et al., 2026] actively collapse separability. The Latent Critic exploits this bidirectional malleability. Drawing structural inspiration from pause tokens [Goyal et al., 2024] to accumulate these signals, we design a masked diagnostic objective that restructures the representation space, amplifying the geometric separation of specification grounding to enable an actionable, end-to-end architecture.

7 Conclusion

Our work bridges the gap between mechanistic interpretability and AI agent safety by introducing the Latent Critic. Rather than relying on surface-level text or passive scalar probes, our architecture actively restructures the transformer's residual stream to expose specification-grounding failures in real time. Mechanistically, we demonstrate via causal activation patching and probe analyses that a masked diagnostic objective restructures brittle internal state representations into a more stable, linearly separable subspace. By projecting this amplified geometry into targeted natural language, the Critic provides granular localization with negligible, single-pass latency, outperforming fine-tuned external evaluators, semantic entropy baselines, and passive internal probes. Deploying this architecture in a closed-loop environment proves that agents can utilize their own localized uncertainty as specific, actionable feedback for zero-shot self-correction, mitigating the safety-productivity tradeoff.

Despite these results, several limitations scope our current work. First, training the Critic relies on programmatic labels derived from a simulated environment, and our evaluation sets were verified by a single human annotator. While the programmatic mask provides reliable ground-truth localization for structured tool-calling, extending span-level localization to open-ended generation remains an open challenge. Second, the reliance on the base model's internal state means that not only is downstream self-correction upper-bounded by the frozen model's capabilities, but the internal detection signal itself weakens under severe distribution shift when the base agent's generation quality collapses. Relatedly, we observed that the performance advantage of internal access over external judges under shift is model-dependent. Ultimately, our finding that a masked diagnostic objective restructures latent representations of grounding to improve real-time detection and localization suggests a promising training principle to render internal uncertainty mechanically accessible across transformer architectures through PEFT.

References

Amos Azaria and Tom Mitchell. The internal state of an llm knows when it's lying, 2023. URL https://arxiv.org/abs/2304.13734.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. Training a helpful and harmless assistant with reinforcement learning from human feedback, 2022. URL https://arxiv.org/abs/2204.05862.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Guiding llms the right way: Fast, non-invasive constrained generation, 2024. URL https://arxiv.org/abs/2403.06988.

Collin Burns, Haotian Ye, Dan Klein, and Jacob Steinhardt. Discovering latent knowledge in language models without supervision, 2024. URL https://arxiv.org/abs/2212.03827.

Yizhou Chi, Jessy Lin, Kevin Lin, and Dan Klein. Clarinet: Augmenting language models to ask clarification questions for retrieval, 2024. URL https://arxiv.org/abs/2405.15784.

Ahmed M. Darwish, Essam A. Rashed, and Ghada Khoriba. Mitigating llm hallucinations using a multi-agent framework. *Information*, 16(7), 2025. ISSN 2078-2489. doi: 10.3390/info16070517. URL https://www.mdpi.com/2078-2489/16/7/517.

Javier Ferrando, Oscar Obeso, Senthooran Rajamanoharan, and Neel Nanda. Do i know this entity? knowledge awareness and hallucinations in language models, 2025. URL https://arxiv.org/abs/2411.14257.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing, 2024. URL https://arxiv.org/abs/2305.11738.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens, 2024. URL https://arxiv.org/abs/2310.02226.

Kait Healy, Bharathi Srinivasan, Visakh Madathil, and Jing Wu. Internal representations as indicators of hallucinations in agent tool selection. *arXiv preprint arXiv:2601.05214*, 2026.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Xu Hu, Yifan Zhang, Songtao Wei, Chen Zhao, Qiannan Li, Bingzhe Li, and Feng Chen. Small updates, big doubts: Does parameter-efficient fine-tuning enhance hallucination detection ?, 2026. URL https://arxiv.org/abs/2602.11166.

Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. Towards mitigating LLM hallucination via self reflection. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 1827–1843, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.123. URL https://aclanthology.org/2023.findings-emnlp.123/.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El-Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. Language models (mostly) know what they know, 2022. URL https://arxiv.org/abs/2207.05221.

Hyuhng Joon Kim, Youna Kim, Cheonbok Park, Junyeob Kim, Choonghyun Park, Kang Min Yoo, Sang goo Lee, and Taeuk Kim. Aligning language models to explicitly handle ambiguity, 2024. URL https://arxiv.org/abs/2404.11972.

Jannik Kossen, Jiatong Han, Muhammed Razzak, Lisa Schut, Shreshth Malik, and Yarin Gal. Semantic entropy probes: Robust and cheap hallucination detection in llms, 2024. URL https://arxiv.org/abs/2406.15927.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation, 2023. URL https://arxiv.org/abs/2302.09664.

Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model, 2024. URL https://arxiv.org/abs/2306.03341.

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, 2023. URL https://arxiv.org/abs/2304.08244.

Potsawee Manakul, Adian Liusie, and Mark J. F. Gales. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models, 2023. URL https://arxiv.org/abs/2303.08896.

Samuel Marks and Max Tegmark. The geometry of truth: Emergent linear structure in large language model representations of true/false datasets, 2024. URL https://arxiv.org/abs/2310.06824.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947. doi: 10.1007/BF02295996.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt, 2023. URL https://arxiv.org/abs/2202.05262.

Nay Myat Min, Long H. Pham, Hongyu Zhang, and Jun Sun. Corvus: Red-teaming hallucination detectors via internal signal camouflage in large language models, 2026. URL https://arxiv.org/abs/2601.14310.

Oscar Obeso, Andy Arditi, Javier Ferrando, Joshua Freeman, Cameron Holmes, and Neel Nanda. Real-time detection of hallucinated entities in long-form generation, 2026. URL https://arxiv.org/abs/2509.03531.

Hadas Orgad, Michael Toker, Zorik Gekhman, Roi Reichart, Idan Szpektor, Hadas Kotek, and Yonatan Belinkov. Llms know more than they show: On the intrinsic representation of llm hallucinations, 2025. URL https://arxiv.org/abs/2410.02707.

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*, 2025. URL https://openreview.net/forum?id=2GmDdhBdDk.

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, 2023. URL https://arxiv.org/abs/2307.16789.

Qwen Team. Qwen3.5. https://qwen.ai/blog?id=qwen3.5, 2026. Accessed: 2026-04-13.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Weihang Su, Changyue Wang, Qingyao Ai, Yiran HU, Zhijing Wu, Yujia Zhou, and Yiqun Liu. Unsupervised real-time hallucination detection based on the internal states of large language models, 2024. URL https://arxiv.org/abs/2403.06448.

Yiyou Sun, Yu Gai, Lijie Chen, Abhilasha Ravichander, Yejin Choi, and Dawn Song. Why and how llms hallucinate: Connecting the dots with subsequence associations, 2025. URL https://arxiv.org/abs/2504.12691.

Manan Suri, Puneet Mathur, Nedim Lipka, Franck Dernoncourt, Ryan A. Rossi, and Dinesh Manocha. Structured uncertainty guided clarification for llm agents, 2026. URL https://arxiv.org/abs/2511.08798.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases, 2023. URL https://arxiv.org/abs/2306.05301.

Sanidhya Vijayvargiya and Rahul Lokesh. Efficient on-device agents via adaptive context management, 2025. URL https://arxiv.org/abs/2511.03728.

Sanidhya Vijayvargiya, Vijay Viswanathan, and Graham Neubig. Asking what matters: Reward-driven clarification for software engineering tasks, 2026a. URL https://arxiv.org/abs/2604.14624.

Sanidhya Vijayvargiya, Xuhui Zhou, Akhila Yerukola, Maarten Sap, and Graham Neubig. Ambigswe: Interactive agents to overcome underspecificity in software engineering, 2026b. URL https://arxiv.org/abs/2502.13069.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, Zhiwei Liu, Yihao Feng, Tulika Awalgaonkar, Rithesh Murthy, Eric Hu, Zeyuan Chen, Ran Xu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. xlam: A family of large action models to empower ai agent systems, 2024. URL https://arxiv.org/abs/2409.03215.

Michael JQ Zhang and Eunsol Choi. Clarify when necessary: Resolving ambiguity through interaction with LMs. In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 5541–5558, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.306. URL https://aclanthology.org/2025.findings-naacl.306/.

A Data Generation and Annotation Pipeline

To train the Latent Critic, we built an automated, highly scalable pipeline to generate ambiguous tool-calling scenarios and accurately label specification-grounding failures. This avoids the prohibitive cost of human annotation for the large training corpus while maintaining high-fidelity ground truth.

A.1 Prompt Templates

We utilize two primary prompts in our pipeline: the Simulated User (which generates the programmatic ground truth mask) and the 122B Oracle Judge (which acts as a final denoising step for complex implicit references).

Simulated User Prompt

```
You are a user trying to complete tasks. Do not break character.
Persona: {scenario['user_persona']}
Goal: {scenario['scenario_description']}

These are the underlying GROUND TRUTH facts of your request (what you
eventually want the agent to execute):
{gt_tools}

Based on the agent's responses, behave naturally. If the agent asks for
missing information, provide it from the Ground Truth facts. Do not instruct
the agent to call tools or mention the tools. Just respond as a user would,
providing information when asked.

CRITICAL INSTRUCTIONS:
You MUST output ONLY a valid JSON object. No conversational filler outside
the JSON. The JSON must contain exactly two keys: "user_message" and
"completed_specifications".

Your "completed_specifications" must STRICTLY mirror the structure of the
Ground Truth tools. However, replace the actual values with binary integers
(0 or 1) representing whether you have PROVIDED that information to the
agent yet.
1 = You HAVE explicitly or implicitly provided this specific parameter in
  the conversation.
0 = You HAVE NOT provided it yet.

In your <think> tags, explicitly justify why you are setting each parameter
to 0 or 1 based on the conversation history. If the agent successfully
executes the tools and your goal is complete, output exactly: <FINISH>

EXAMPLE OUTPUT FORMAT:
{
 "user_message": "I want to fly to Paris, please.",
 "completed_specifications": [
 {
  "name": "book_flight",
  "arguments": {
  "destination": 1,
  "date": 0,
  "class": 0
  }
 }
 ]
}
```

122B Oracle Judge Prompt (Denoising)

```
You are an expert Data Quality Validator for an AI training pipeline.
The agent just made a tool call. Your job is to read the ACTUAL user
conversation history and determine if the agent hallucinated.

USER'S CONVERSATION HISTORY:
{history}

AGENT'S TOOL CALL:
{tool_calls}

INSTRUCTIONS:
1. Look at the arguments in the Agent's Tool Call.
2. Did the User EXPLICITLY state or imply those values in the Conversation
  History? (e.g., User says "tomorrow", Agent writes "2024-05-16" -> This
  is grounded).
3. If the agent guessed, assumed, or hallucinated a value the user NEVER
  provided, the label is 'ungrounded'.
4. If ALL arguments were provided by the user, the label is 'ok'.
5. If arguments can be inferred or are provided implicitly, consider them
  specified. Specification can be explicit or implicit.

Output ONLY valid JSON:
{
 "thought_process": "Briefly explain.",
 "final_label": "ok" | "ungrounded",
 "specific_errors": "List hallucinated params here, else empty."
}
```

A.2 Annotation Process Walkthrough

The classification of a generated tool call into `ok`, `wrong_tool`, or `ungrounded` is determined through a multi-stage filtering process:

1. **Programmatic Mask Generation (Simulated User):** During the dialogue, the Simulated User maintains a hidden schema matching the underlying API, assigning a binary integer to each parameter: `1` if the user has provided the information, `0` if it remains withheld.
2. **Rule-Based Mask Evaluation:** When the base agent generates a tool call, the pipeline first cross-references the API name. If the tool is entirely incorrect for the user's goal, it is labeled `wrong_tool`. If the tool is correct, the pipeline checks the generated parameters against the user's binary mask. If the agent outputs a parameter that corresponds to a `0` in the mask, it is immediately flagged as a potential `ungrounded` hallucination.
3. **Heuristic Fallback Check:** Because LLMs frequently paraphrase (e.g., the user says "New York" but the mask expected "JFK"), a simple heuristic acts as a rapid fail-safe. If the hallucinated parameter's exact string value is present anywhere in the raw user conversation history, the flag is overridden and corrected to `ok`.
4. **122B Oracle Judge Refinement:** For all remaining parameters flagged as `ungrounded`, the trajectory is passed to an offline 122B LLM Judge (Qwen3.5-122B). The judge performs a deep semantic read to verify if the parameter could be logically inferred or implicitly derived from the context. If it cannot be inferred, the trajectory is definitively labeled `ungrounded:[parameter_name]`.

B Experimental and Baseline Details

B.1 Linear Probe (SVM) Baseline Details

To establish the internal probing baseline compared in Section 3, we trained layer-wise Support Vector Machines (SVMs) on the hidden states of the base models. For each trajectory in the training

set, we extracted the native hidden states of the frozen base model at the final generation token (prior to any adapter intervention).

An SVM with a linear kernel was trained at each layer to classify the states into grounded (`ok`) versus hallucinated (`ungrounded`) tool calls. Through this layer-wise evaluation, we empirically determined that the latent specification-grounding signal peaks in the mid-to-late layers of the network. Specifically, for the **Qwen3-4B** model, the optimal linear separability was observed at **Layer 21**. For **Llama-xLAM-2-8B**, it was observed at **Layer 31**. All empirical results and baseline metrics reported for the Linear Probes utilize the classification outputs from these optimal layers.

B.2 External Judge Training Details

To ensure a rigorous and strictly controlled comparison between internal representations and external text evaluation, the External Judge baseline is a fine-tuned Qwen3.5-4B model. It was trained on the exact same 16,000 ID examples as the Latent Critic. Furthermore, it utilized identical LoRA hyperparameters (Rank=64, Alpha=128), learning rates, and epochs. Crucially, the External Judge was trained to output the exact same diagnostic classification string format (e.g., `ungrounded: [parameter_name]`) based on the CSL-derived labels, allowing us to isolate the impact of *where* the model reads the signal (surface text vs. residual stream) rather than differing model capacities or label distributions.

B.3 Training Hyperparameters & Compute

To ensure reproducibility, the training hyperparameters for the Latent Critic (Parameter-Efficient Fine-Tuning via LoRA) are detailed in Table 6. The base model's weights remain entirely frozen, and gradients are only calculated with respect to the generated `[POS]` token and its corresponding classification output.

Table 6: Latent Critic LoRA Training Hyperparameters

Hyperparameter	**Value**
LoRA Rank (r)	64
LoRA Alpha (α)	128
Target Modules	`q_proj, k_proj, v_proj, o_proj`
Learning Rate	2×10^{-5}
Batch Size	32
Epochs	3
Optimizer	AdamW
Learning Rate Scheduler	Cosine with Warmup
Warmup Ratio	0.05
Trigger Token	`[POS]`
Loss Masking	Enabled (All context & JSON syntax masked)

Training Compute: Training the Latent Critic and the External Judge involves updating a highly restricted subset of parameters (LoRA adapters). For the Qwen3-4B and Llama-xLAM-2-8B base models, adapter training was performed on a single machine utilizing $1 \times$ NVIDIA A100 (80GB) GPUs. Due to the aggressive loss masking (calculating gradients strictly for the `[POS]` token and diagnostic label), training converged rapidly, requiring approximately 3 to 5 GPU-hours per model.

Inference Latency Overhead: A primary contribution of the Latent Critic is its ability to operate without the secondary inference latency typically associated with LLM-as-a-judge frameworks. Because the adapter's forward passes occur concurrently during the base model's standard autoregressive generation, the only added computational overhead is the generation of the final `[POS]` trigger token and the short diagnostic string (typically 2 to 5 tokens). In standalone HuggingFace testing on an RTX 3090, the Critic added a mean latency of just **58 ms**, compared to 884 ms for the 4B External Judge and over 12 seconds for 10-sample Semantic Entropy. In our local vLLM deployment environment utilized for closed-loop multi-turn agentic rollouts, this localized single-pass generation added $<$**10 milliseconds** of latency per executed tool call, safely satisfying the strict real-time requirements of autonomous agents.

B.4 Architectural Validation (Ablations)

We systematically ablated core components of the architecture to ensure the SFT objective drives maximal gains.

Masked Loss Objective: Removing the tool-call mask and forcing the adapter to optimize for both JSON syntax generation and the diagnostic label resulted in a noticeable 8% drop in `ungrounded` detection accuracy. Forcing the adapter to participate in syntactic generation actively dilutes its representation capacity for semantic evaluation and slows convergence.

The `[POS]` Trigger Token: Removing the trigger token from the training data and forcing the model to seamlessly append the diagnostic label directly to the end of the JSON array caused a severe 22% drop in accuracy. The adapter cannot effectively evaluate a sequence while simultaneously participating in its construction. The structural boundary of the trigger token is strictly required to act as an attention sink, providing an isolated computational step and acting as a dedicated trigger for presenting its classification.

C Detailed Detection Metrics & Robustness Analyses

Note on Metric Thresholds: In the main text and following tables, the classification F1 scores for the Latent Critic and External Judge are decision-based (i.e., evaluated directly from greedy decoding outputs). In contrast, the F1 scores for the continuous baselines (Linear Probe, Token/Semantic Entropy) are evaluated at the optimal threshold swept over the evaluation split. Thus, the continuous baseline F1 scores are selection-optimistic, emphasizing the necessity of relying on the threshold-independent AUROC and AUPRC metrics reported in the main text for rigorous comparison.

C.1 Detailed Class-Wise Performance (In-Distribution)

Table 7 provides the granular Precision, Recall, and F1-Scores across all three classes (`ok`, `ungrounded`, `wrong_tool`) for the primary models evaluated in-distribution.

Base Model	Evaluator	Class	Precision	Recall	F1-Score	Param EM (%)
Qwen-4B	Latent Critic	ok	**0.874**	0.937	**0.904**	**83.83**
		ungrounded	**0.891**	**0.849**	**0.870**	
		wrong_tool	**0.985**	**0.930**	**0.957**	
	External Judge	ok	0.706	**0.952**	0.811	70.63
		ungrounded	0.857	0.585	0.695	
		wrong_tool	0.985	0.901	0.941	
	Linear Probe	ok	0.872	0.885	0.879	N/A
		ungrounded	0.711	0.684	0.697	
		wrong_tool	-	-	-	
Llama-xLAM 8B	Latent Critic	ok	**0.966**	0.875	**0.918**	**80.73**
		ungrounded	**0.893**	**0.833**	**0.862**	
		wrong_tool	**0.885**	**0.979**	**0.929**	
	External Judge	ok	0.962	0.781	0.862	77.06
		ungrounded	0.686	0.800	0.739	
		wrong_tool	0.875	0.894	0.884	
	Linear Probe	ok	0.904	**0.931**	0.917	N/A
		ungrounded	0.781	0.714	0.746	
		wrong_tool	-	-	-	

Table 7: Detailed Class-Wise Performance (In-Distribution). $\pm$ uncertainty omitted for layout clarity. The Latent Critic achieves dominant detection across all classes, driving a significant improvement in exact parameter-level match accuracy.

C.2 Robustness to Generation Length

To ensure performance differences are not simply artifacts of length confounding, we evaluated detection performance across equal-count generation-length quartiles (character count), plus the extreme upper decile ($\geq$ P90). The task evaluates binary grounded vs. hallucinated. As detailed in Table 8, the Latent Critic consistently outperforms the External Judge across every length stratum.

Furthermore, we computed the correlation between detection errors and output features. None of the examined features strongly predict Critic errors (all $|r| \leq 0.091$), whereas the External Judge's errors correlate much more strongly with argument length (Pearson $r = 0.173$, Spearman $\rho = 0.113$).

Length Bin (chars)	Base Rate	Latent Critic AUROC	External Judge AUROC
$[51, 90)$	0.447	**0.966 ± 0.018**	0.894 ± 0.042
$[90, 128)$	0.608	**0.965 ± 0.023**	0.926 ± 0.029
$[128, 174)$	0.584	**0.980 ± 0.016**	0.968 ± 0.019
$[174, 854)$	0.697	**0.955 ± 0.025**	0.903 ± 0.040
≥ 292 (P90)	0.645	**0.865 ± 0.080**	0.803 ± 0.106

Table 8: Detection performance stratified by output length quartiles. The Critic's advantage holds across all lengths, proving it is not confounded by short vs. long payloads. (Point estimates favor the Critic at P90, though unresolvable at $n = 31$).

C.3 Robustness to Task Difficulty

We utilized two proxies for task difficulty: the number of required tool parameters and the depth of the conversational trajectory (number of turns). As shown in Table 9, the Latent Critic remains the strongest detector across all levels of task complexity, and its advantage over the External Judge does not diminish under a higher cognitive grounding burden.

Difficulty Proxy	Stratum	Latent Critic AUROC	External Judge AUROC
# Required Params	1 parameter	**0.926**	0.896
	2 parameters	**0.971**	0.880
	3+ parameters	**0.978**	0.960
# Conversational Turns	Depth 1–3	**0.959**	0.897
	Depth 3–5	**0.963**	0.918
	Depth 5–27	**0.982**	0.954

Table 9: Detection performance stratified by task difficulty proxies. Uncertainty margins omitted for layout clarity.

C.4 Operating Points and Additional OOD Metrics

To evaluate performance under strict safety constraints, we analyzed the True Positive Rate (TPR) at fixed False Positive Rate (FPR) budgets for the `ungrounded` class. As shown in Table 10, the Latent Critic significantly outperforms all baselines in the low-FPR regime. Remarkably, at the strictest 1% FPR operating point, the Critic's True Positive Rate remains essentially unchanged under distribution shift (38.7% ID vs. 38.9% OOD).

Additionally, while our primary evaluation focuses on specification grounding, we also tracked performance on the `wrong_tool` class under distribution shift. The Latent Critic achieves an OOD F1 of 0.360 ± 0.120 compared to the External Judge's 0.087 ± 0.193. We interpret this advantage cautiously due to the very small sample size of OOD `wrong_tool` instances ($n \approx 16$), but it is consistent with the Latent Critic's overall robustness.

Method	TPR @ FPR (%) (ID)			
	1% FPR	5% FPR	10% FPR	20% FPR
Token Entropy	1.9	17.0	21.7	40.6
Semantic Entropy	4.7	13.2	32.1	41.5
SEP (Kossen et al.)	0.0	0.0	10.4	22.6
Linear Probe	3.8	20.8	41.5	66.0
External Judge	28.3	63.2	70.8	83.0
Latent Critic (ID)	**38.7**	**75.5**	**97.2**	**98.1**
Latent Critic (OOD)	**38.9**	**68.5**	**80.6**	**88.0**

Table 10: Operating points (True Positive Rate at specific False Positive Rate budgets) for the `ungrounded` class on Qwen3-4B. The Latent Critic identifies the most hallucinations under strict safety budgets, and its performance at the strictest 1% threshold survives distribution shift.

C.5 Lexical Decidability Analysis

A critical consideration is whether our definition of "specification grounding" is simply reducible to verbatim string matching. To test this, we analyzed the presence of parameter string values within the user's conversation history.

We found that 73.2% (ID) and 81.3% (OOD) of grounded parameter values appear verbatim (or after simple normalization) in the dialogue. However, 51.1% (ID) and 56.7% (OOD) of *hallucinated* values also appear somewhere in the context. Thus, lexical presence is neither sufficient (as models frequently fabricate parameters by stitching together unrelated context fragments, meaning lexical presence does not equate to being grounded for that specific parameter) nor necessary (as many grounded values, such as enums, booleans, or parsed dates, are implicitly derived rather than copied).

A simple lexical-match baseline achieves an AUROC of 0.670 [0.604–0.731] In-Distribution and 0.731 [0.677–0.783] Out-of-Distribution. While this outperforms the sampling-based uncertainty methods (which collapse entirely), it falls dramatically short of the Latent Critic's 0.966 / 0.925 AUROC. This confirms that accurate detection requires a genuinely semantic, contextual understanding of grounding, rather than simple lexical presence.

C.6 Out-of-Distribution Failure Taxonomy

In Section 5.2, we noted that Trajectory Success on the OOD ToolAlpaca schemas remained close to the frozen base model's ceiling ($\sim$2%) regardless of the safety intervention applied. The ToolAlpaca dataset is substantially more difficult than our ID benchmark. Evaluation trajectories average 12.4 turns and extend up to 32 turns.

We found that only 27% of failed evaluation trajectories contain a parameter-level `ungrounded` or `wrong_tool` error. The majority of OOD failures arise from broader capability limitations outside the scope of specification grounding, including malformed tool calls, incorrect tool selection (inventing tools not present in the system prompt), refusal to execute required calls, and schema type errors (e.g., passing strings instead of integers). Often, these failures compound across the long multi-turn trajectories.

Thus, our contribution under distribution shift is intentionally scoped: when specification-grounding failures *do* occur, the Critic detects them accurately, prevents harmful execution, and enables recovery where possible. However, the ultimate trajectory success rate remains bounded by the base agent's underlying planning and functional tool-calling capabilities.

D RQ3: Autonomous Recovery Walkthroughs

To demonstrate the real-world utility of the Latent Critic, we provide a step-by-step qualitative walkthrough of a closed-loop ReAct deployment. This example illustrates how the base model's task-completion bias forces an unsafe hallucination, how the Latent Critic concurrently intercepts the error, and how the injected environmental feedback seamlessly triggers the agent's conversational recovery policy.

Scenario Background: The user requests device management actions (checking battery, enabling power saving, and silencing notifications). The `manage_battery` tool requires a `power_saving_mode` parameter (`medium`, `maximum`, or `adaptive`), which the user omits.

1. User Request:
"Check my battery level, turn on power saving mode, and silence notifications for the next hour."

2. Base Agent Generation (Task-Completion Bias Triggered):
The agent recognizes the correct tool but is missing the required mode. Rather than halting to ask, it confidently guesses the missing variable.
`[{"name": "manage_battery", "arguments": {"action": "enable_power_saving", "power_saving_mode": "medium"}}, ...]`

3. Latent Critic Concurrent Extraction (Execution Blocked):
Triggered upon tool call completion, the adapter projects the internal uncertainty directly into the output space without secondary evaluation latency.
Latent Critic Output: `[POS] ungrounded: ["power_saving_mode"]`

4. Environmental Feedback Injection:
The execution framework intercepts the tool call based on the Latent Critic's Specific flag and returns a synthesized error to the agent's context window.
Observation: `{"status": "error", "message": "Parameter mismatch. Expected explicit specification based on user intent, but received 'medium'. Please update the power_saving_mode parameter."}`

5. Agent Conversational Recovery (Self-Correction):
Reading the specific environmental feedback, the base model naturally pivots its policy from autonomous execution to human-in-the-loop clarification.
Agent: *"...Power saving mode was set to "medium" instead of what you might have intended. Please update the power saving mode to "maximum" or "adaptive" if you want it set to a specific level."*

6. User Clarification & Safe Execution:
User: *"Yes, please set the power saving mode to maximum."*
Agent: `[{"name": "manage_battery", "arguments": {"action": "enable_power_saving", "power_saving_mode": "maximum"}}]`
Latent Critic: `[POS] ok` *(Execution proceeds safely).*

This trajectory highlights a crucial paradigm shift: the autonomous safety boundary is not enforced by a secondary "judge" model reading surface-level text, but by translating the primary agent's own grounding boundaries into hard environmental constraints. By doing so, we successfully utilize the base model's strong conversational abilities without succumbing to its unsafe execution biases.

D.1 Failure Mode Walkthroughs

To empirically ground the limitations discussed in Section 5.3 and Section 7, we provide two qualitative walkthroughs illustrating trajectories where the Latent Critic successfully intercepts a hallucination, but the base agent fails to achieve ultimate task recovery. These examples highlight why downstream recovery is fundamentally upper-bounded by the base model's native conversational policy.

D.1.1 Failure Mode 1: Partial Parameter Recovery (Task Abandonment)

Scenario Background: The user asks to schedule a meeting, but omits the date, time, and duration. The agent guesses all three. Upon receiving environmental feedback, the agent corrects some parameters but fails to clarify the others, resulting in a continuous block because partial recovery is not rewarded in strict schema execution.

1. User Request:
"Schedule a project sync with Sarah."

2. Base Agent Generation (Task-Completion Bias):
The agent guesses the missing date, time, and duration to force execution.
`[{"name": "schedule_meeting", "arguments": {"contact_name": "Sarah", "date": "2026-04-28", "time": "14:00", "duration_minutes": 30}}]`

3. Latent Critic Concurrent Extraction (Execution Blocked):
Latent Critic Output: `[POS] ungrounded: ["date", "time", "duration_minutes"]`

4. Environmental Feedback Injection:
Observation: `System Error: Tool blocked. You hallucinated the parameters: "date", "time", "duration_minutes".`

5. Agent Conversational Pivot (Partial Clarification):
The agent pivots to ask the user, but its native policy forgets to ask for all missing variables.
Agent: "What day and time would you like to schedule the project sync with Sarah?"

6. User Response:
User: "Let's do tomorrow at 10 AM."

7. Base Agent Generation (Incomplete Fix):
The agent fills in the date and time, but *still guesses* the duration.
`[{"name": "schedule_meeting", "arguments": {"contact_name": "Sarah", "date": "2026-04-28", "time": "10:00", "duration_minutes": 30}}]`

8. Latent Critic Concurrent Extraction (Execution Blocked Again):
Latent Critic Output: `[POS] ungrounded: ["duration_minutes"]`

9. Final Agent Failure (Task Abandonment):
Rather than asking for the final parameter, the agent's conversational logic degrades.
Agent: "I'm sorry, I am having trouble accessing the calendar system right now. Please try scheduling the meeting again later."

D.1.2 Failure Mode 2: Stubborn Guessing Loops

Scenario Background: The user asks to set an alarm without specifying the time. Instead of routing the error to human-in-the-loop clarification, the agent attempts to blindly guess different parameters to bypass the environmental block.

1. User Request:
"Set an alarm for my morning workout."

2. Base Agent Generation (Task-Completion Bias):
`[{"name": "set_alarm", "arguments": {"time": "06:00", "label": "morning workout", "repeat_days": ["Monday", "Wednesday", "Friday"]}}]`

3. Latent Critic Concurrent Extraction (Execution Blocked):
Latent Critic Output: `[POS] ungrounded: ["time", "repeat_days"]`

4. Environmental Feedback Injection (Generic):
Observation: `System Error: Tool blocked. You hallucinated the parameters: "time", "repeat_days".`

5. Base Agent Generation (Blind Retry):
Ignoring the implicit instruction to ask the user, the agent immediately tries another plausible guess.
`[{"name": "set_alarm", "arguments": {"time": "07:00", "label": "morning workout", "repeat_days": ["Everyday"]}}]`

6. Environmental Feedback Injection (Generic):
Observation: `System Error: Tool blocked. You hallucinated the parameters: "time", "repeat_days".`

7. Final Agent Failure:
Agent: :System error. I cannot set the alarm at this time. What else can I help you with?"

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction clearly outline both the capabilities and the scope of the Latent Critic. These claims are empirically validated in Section 3 (detection accuracy), mechanistically verified in Section 4 (latent geometry), and evaluated for real-world agentic deployment in Section 5 (safety vs. recall trade-offs).

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Limitations regarding the base model's inherent conversational recovery capabilities and the requirement of structured schema environments are explicitly discussed in Section 5.2 and Section 7. The scope limitation is also presented in Section 1 and highlighted again in Section 7. Furthermore, Appendix D provides detailed qualitative walkthroughs of specific failure modes where the agent fails to recover despite accurate feedback.

   Guidelines:

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

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

   Justification: The paper relies on empirical and architectural contributions rather than formal mathematical proofs or theorems.

   Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: We fully disclose all information necessary for independent reproduction. This includes the exact public datasets used to derive scenarios (Tang et al. [2023], Vijayvargiya and Lokesh [2025]), the exact simulated user and oracle judge prompt templates (Appendix A.1), the programmatic mask logic (Section 2.2), and all LoRA training hyperparameters (Appendix B.2).

   Guidelines:

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

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

   Answer: [No]

   Justification: The codebase is proprietary and cannot be open-sourced. However, the evaluation methodology utilizes publicly available dataset schemas, and we have provided the exact system prompts and programmatic data-generation methodologies in the appendix to facilitate reproduction.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Dataset derivation, trajectory generation, and evaluation splits are detailed in Section 2. Exact LoRA training hyperparameters, targets, and optimization details are provided in Appendix B.2.

   Guidelines:

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

7. **Experiment statistical significance**

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

Answer: [Yes]

Justification: In Section 3.2, we report statistical significance comparing detection architectures using McNemar's test ($p < 0.001$) [McNemar, 1947]. Additionally, in the mechanistic evaluations (Section 4), we visualize trajectory variance using shaded regions representing ± 1 standard deviation.

Guidelines:

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

8. **Experiments compute resources**

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

Answer: [Yes]

Justification: Appendix B.2 explicitly details the compute resources utilized, noting that adapter training was performed on a single NVIDIA A100 (80GB) GPU, requiring approximately 3 to 5 GPU-hours per model due to the targeted loss masking.

Guidelines:

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

9. **Code of ethics**

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

Answer: [Yes]

Justification: The research conforms to the NeurIPS Code of Ethics, specifically focusing on improving the safety, reliability, and factual grounding of autonomous AI agents.

Guidelines:

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

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: The paper intrinsically addresses a safety and reliability issue (hallucination leading to unauthorized tool execution), discussing how the Latent Critic acts as a necessary autonomous safety boundary.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper does not release new foundational models or high-risk datasets; rather, it proposes an architectural method to improve the safety of existing models.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: The paper cites the base models (Qwen, Llama-xLAM) and the benchmarks (Tang et al. [2023], Vijayvargiya and Lokesh [2025]) used to derive the experimental scenarios.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [N/A]

    Justification: No new code or datasets are being released with this submission due to proprietary restrictions.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or human subjects research. Data generation and tracking were automated via programmatic masks and simulated LLM users.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

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

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

    Answer: [N/A]

    Justification: The research does not involve human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

    Answer: [Yes]

    Justification: Large Language Models (specifically Qwen-3.5-122B) are used as simulated users and Oracle Judges within the programmatic data generation pipeline. This core methodological usage is fully documented in Section 2.2 and Appendix A. LLM usage for standard writing/editing is omitted as per conference guidelines.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2026/LLM`) for what should or should not be described.