LatentReAct: Decomposed Latent Verification for Robust Tool-Using Agents

Zhengpeng Lan, Ronghua Ye(✉), and Jianmin Han

School of Computer Science and Technology, Zhejiang Normal University, Jinhua, China
{lanzhengpeng,rhye}@zjnu.edu.cn, hanjm@zjnu.cn

Abstract. Large Language Models (LLMs) exhibit promising capabilities as autonomous agents within the ReAct framework, yet they remain vulnerable to cascading errors in multi-step tool invocation scenarios. While introducing a lightweight verifier offers a natural mitigation strategy, we observe that end-to-end textual judgments from such verifiers are often unstable and unreliable. In this paper, we propose LatentReAct, a latent-space verification framework that enhances ReAct agents through decompositional representation validation. Instead of relying on generated text, LatentReAct extracts intermediate-layer latent representations from a frozen 3B-parameter model and decomposes action evaluation into multiple semantic dimensions. A lightweight policy head then scores candidate actions based on these latent features. Experiments on ToolBench and BFCL v4 demonstrate that LatentReAct significantly outperforms strong baselines, achieving 74.47% verification accuracy on ToolBench, improving end-to-end execution accuracy by up to 5 percentage points, and reducing unnecessary reasoning steps by 12–16%. These results highlight the potential of leveraging latent representations as a stable and cost-effective mechanism for building reliable LLM-based agents.

Keywords: Large Language Models · Autonomous Agents · ReAct · Latent Representation · Tool Use

1 Introduction

Large Language Models (LLMs) have emerged as powerful autonomous agents capable of performing complex tasks by interacting with external tools such as APIs and databases [1–3]. The ReAct framework [4], which interleaves reasoning and action, has become a widely adopted paradigm for tool-use agents. Despite its effectiveness, ReAct agents are prone to cascading errors: an incorrect action early in the loop can mislead subsequent steps and reduce overall task reliability [5].

Existing approaches to improve ReAct agents often rely on end-to-end generative evaluation, prompt engineering, or fine-tuning. For prompt engineering, methods such as Chain-of-Thought [6] have been explored; for fine-tuning, approaches like ReST meet ReAct [7] aim to improve agent performance. These methods are either fragile—susceptible to format errors and low evaluation accuracy—or computationally expensive,

D.-S. Huang et al. (Eds.): ICIC 2026, CCIS 3040, pp. 3–15, 2027.
https://doi.org/10.1007/978-981-92-3557-5_1

limiting their scalability. Even approaches that allow self-correction, such as Reflexion [8] and Self-Refine [9], cannot fully prevent error propagation, as the feedback arrives only after the faulty action has already influenced subsequent steps. Moreover, recent studies have shown that LLMs' internal representations encode richer information about truthfulness than their surface-level outputs [11], suggesting that real-time error interception may be possible by probing these hidden states.

To address these limitations, we introduce LatentReAct, a framework that performs real-time action verification directly in the latent space of a frozen 3B-parameter language model. Unlike prior methods that operate on surface-level text, LatentReAct extracts hidden states from intermediate transformer layers and evaluates candidate actions along three complementary dimensions: logical redundancy, target effectiveness, and format compliance. A lightweight policy network then scores each action, and a confidence-based routing mechanism determines whether to adopt the verifier's choice or fall back to the generator's default output.

We evaluate LatentReAct on two challenging benchmarks: ToolBench for component-level verification [2] and BFCL v4 for end-to-end multi-turn reasoning [10]. Experimental results show that our approach consistently improves both verification accuracy and downstream task performance, while reducing unnecessary reasoning steps. By decoupling verification from generation and operating in latent space, LatentReAct offers a lightweight, plug-and-play enhancement for existing ReAct agents.

Our main contributions are summarized as follows:

- We propose leveraging intermediate latent representations from LLMs to assess actions in ReAct loops, effectively decoupling model understanding from fragile text outputs.
- We design a decomposed, multi-dimensional verification framework focusing on logical redundancy, target effectiveness, and format compliance, enabling structured supervision in complex tool-use scenarios.
- We empirically validate our approach on ToolBench and BFCL v4, demonstrating substantial improvements in agent reliability and efficiency, including higher turn-level success rates and fewer reasoning steps.

2 Related Work

2.1 ReAct and Closed-Loop LLM Agents

The ReAct paradigm [4], which interleaves reasoning traces with actions, has become a cornerstone for building tool-using LLM agents. By enabling models to think before they act, ReAct facilitates flexible multi-step decision-making. However, its sequential nature makes it vulnerable to error propagation: a single mistake in an early reasoning step can cascade, leading to complete task failure in complex, multi-turn scenarios [5].

Subsequent frameworks have sought to address this fragility through various mechanisms. For instance, Reflexion [8] introduces episodic memory of verbal reflections, allowing agents to learn from past failures across multiple trials. (evaluated using 3 maximum trials in our experiments) Voyager [12] uses a skill library to improve exploration in Minecraft. Despite these advances, such methods still operate primarily at the text level, relying on generated outputs for self-correction. This reliance makes them

susceptible to issues such as context drift and format inconsistencies (we discuss these limitations in more depth in Sect. 2.3).

In contrast, our LatentReAct framework decouples verification from generation by operating directly on intermediate latent representations, enabling real-time error interception before flawed actions are executed or propagated.

2.2 Latent Representations in LLMs

A growing body of work has demonstrated that the intermediate hidden states of LLMs encode rich semantic, syntactic, and factual knowledge [13, 14]. Researchers have leveraged these representations for various purposes, including probing linguistic properties [15], locating and editing factual associations [16], and steering model generation [17].

More recently, latent features have been used to detect hallucinations [18] and assess model confidence [19]. Furthermore, Orgad et al. [11] demonstrated that truthfulness-related information is highly localized in specific tokens (e.g., exact answer tokens), enabling fine-grained error type prediction and revealing discrepancies between internal encoding and external behavior. However, these applications have primarily focused on analysis or generation control. To the best of our knowledge, we are the first to exploit latent embeddings from a smaller (3B) LLM as a real-time verifier within an agentic ReAct loop.

By extracting hidden states from intermediate layers, we decouple the model's internal understanding from its potentially flawed textual output, mitigating context drift and enabling more stable action evaluation.

2.3 Verification and Self-Correction in Agentic Systems

Enhancing agent reliability through verification and correction has been a key focus of recent research. Self-Refine [9] iteratively improves a single generation through self-feedback, but operates within a single turn and lacks persistent memory. Reflexion [8], as discussed, extends this idea across episodes by storing verbal reflections. CRITIC [20] leverages external tools to validate intermediate steps.

While effective, all these methods share a fundamental limitation: they rely on surface-level text for evaluation, which can be brittle due to generation errors, hallucination, or formatting issues. Moreover, because feedback is typically derived after an action has been taken or a trajectory has completed, they fail to prevent errors from contaminating the ongoing execution—they can only attempt recovery after the fact.

LatentReAct addresses these gaps by introducing a latent-space verification mechanism that operates before action execution. By evaluating candidate actions along multiple decomposed dimensions—such as logical redundancy, target effectiveness, and format compliance—our framework intercepts errors in real time, offering a more robust and fine-grained form of supervision.

2.4 Benchmarks for Tool Learning and Dataset Curation

Evaluating tool-augmented LLMs requires benchmarks that stress multi-step reasoning, API call correctness, and robustness to realistic challenges. ToolBench [2] provides a large-scale collection of tool-use trajectories, while the Berkeley Function Calling Leaderboard (BFCL v4) [10] focuses on scenarios with missing parameters, long contexts, and hallucinated functions.

However, raw trajectories in such datasets often contain incomplete or noisy sequences, making them unsuitable for direct supervised training of a verifier. To obtain high-quality supervision for our policy network, we reconstruct and clean ToolBench using a three-step process:

1. Filtering only successful ReAct trajectories.
2. Identifying steps annotated as erroneous actions.
3. Pairing each erroneous action with its immediate correction to form labeled binary classification examples.

This curation process repurposes ToolBench—originally designed for generative repair—into a high-quality training set for latent-space verification, enabling our model to distinguish correct and incorrect actions independently of fragile text output.

3 Method

We propose LatentReAct, a latent-space verification framework that enhances ReAct agents by leveraging the internal representations of a frozen 3B language model. Unlike approaches that rely on text-level regeneration or full-model fine-tuning, LatentReAct performs structured action evaluation directly in the latent space, enabling lightweight and stable verification.

The core insight is that intermediate hidden states of LLMs encode richer decision-relevant signals than their final textual outputs. By extracting these latent features and training a small policy network to score candidate actions, we can intercept erroneous actions before execution—without modifying the backbone model. A threshold-based routing mechanism then dynamically chooses between the verifier's top candidate and the generator's default output, balancing reliability and efficiency.

3.1 Frozen Latent-Space Verification Framework

As illustrated in Fig. 1, LatentReAct introduces a verification layer that operates directly on the intermediate representations of a frozen LLM. The framework decomposes action evaluation into multiple semantic dimensions, extracts deep latent features at critical token positions, and scores candidates via a lightweight policy network.

Semantic Decoupling and Prompt Construction To obtain discriminative features, we decompose action evaluation into N semantic dimensions (e.g., logical redundancy, target effectiveness, format compliance). For each dimension I, a deterministic mapping

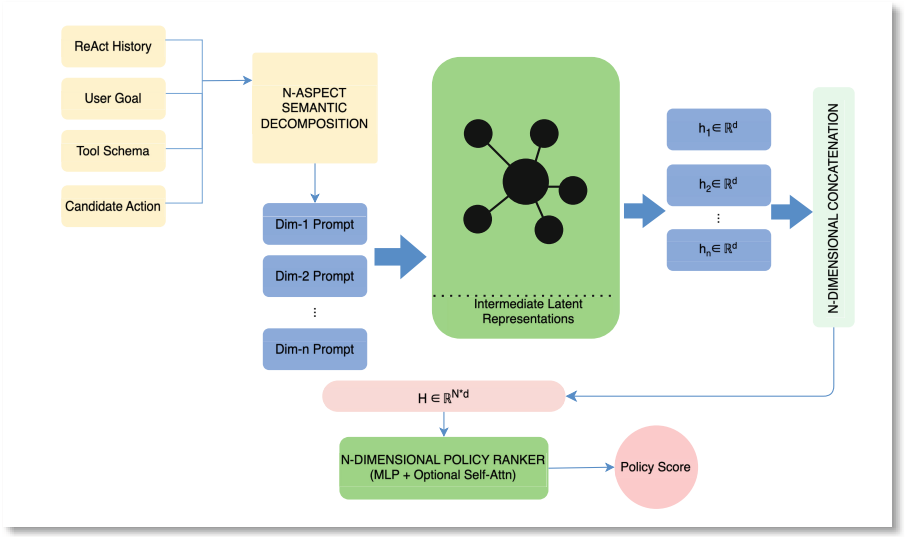

Fig. 1. Architecture of the latent verifier. For each semantic dimension, a structured prompt is constructed, and hidden states are extracted from intermediate layers at the verdict token position. These features are fused and scored by a policy network.

function $\mathcal{F}_{dec}^{(i)}$ constructs a prompt p_i that integrates the reasoning history s_{his}, user goal g_{usr}, tool schema $\mathcal{T}_{sch}$, and candidate action a_c:

$$p_i = \mathcal{F}_{dec}^{(i)}(s_{his}, g_{usr}, \mathcal{T}_{sch}, a_c) \tag{1}$$

Each prompt is fed into the frozen verifier M (a 3B LLM). Instead of using the generated text, we probe the model's internal states. To align with the model's decision point, we locate the token index t_i corresponding to the last token of the generated verdict v_i (e.g., "VALID"):

$$t_i = \max\left\{k \mid y_k^{(i)} \in v_t\right\} \tag{2}$$

where $y_k^{(i)}$ is the $k-th$ generated token for prompt p_i.

Latent Feature Extraction.To capture high-level semantic abstractions while reducing layer-specific noise, we extract hidden states from a set of deep intermediate layers L_{deep} (e.g., layers 24–28). The dimension-specific representation h_i is obtained by mean pooling over these layers at the verdict token position t_i:

$$h_i = \frac{1}{|L_{deep}|}\sum_{l\in L_{deep}} \phi_l^{(t_i)}(p_i) \tag{3}$$

Here, $\phi_l^{(t_i)}(p_i) \in \mathbb{R}^{\mathrm{d}}$ is the hidden state of layer l at token t_i, and d is the hidden dimension. This yields a compact representation that encodes the model's internal confidence prior to vocabulary projection.

The representations from all N dimensions are concatenated to form a unified state matrix $H \in \mathbb{R}^{\mathbb{N}\times\mathbb{d}}$:

$$H = \text{Concat}(h_1, h_2, \ldots, h_N) \tag{4}$$

Policy Network for Scoring. To model interactions between semantic dimensions (e.g., how format compliance affects action validity), we process H through a lightweight policy network. The architecture combines multi-head attention (MHA), layer normalization (LN), and an MLP scoring head:

$$S_\theta(a) = \text{MLP}(\text{Flatten}(\text{LN}(H + \text{MHA}(H)))) \tag{5}$$

where θ denotes trainable parameters. The MLP consists of two hidden layers with ReLU activations. This design captures cross-dimensional dependencies while remaining computationally efficient.

Training Objective. The policy network is trained with a pairwise margin ranking loss. Given a positive action a^+ (the correct correction) and a negative action a^- (the erroneous action) from the same context, we enforce:

$$\mathcal{L}(\theta) = \max\left(0, -\left(S_\theta\left(a^+\right) - S_\theta\left(a^-\right)\right) + \gamma\right) \tag{6}$$

where γ is a margin hyperparameter. This encourages the network to assign higher scores to correct actions.

3.2 Confidence-Based Action Selection

As illustrated in Fig. 2, after the policy network scores K candidate actions $\mathcal{A}_{cand} = \{a_1, \ldots, a_K\}$, we compute the margin between the top score S_{max} and the second-highest score S_{second}:

$$\Delta = S_{\max} - S_{\text{second}} \tag{7}$$

This margin reflects the verifier's confidence: a large Δ indicates a clear preference for one action, while a small Δ suggests uncertainty. Based on a threshold τ, the system dynamically routes execution:

Verifier-Guided Selection ($\Delta > \tau$). When confidence is high, the system executes the top-scoring action a_{max}. This path leverages the verifier's ability to filter out hallucinated or suboptimal actions.

Deterministic Fallback ($\Delta \leq \tau$). When the verifier is uncertain, the system falls back to the original generator's default action a_1 (sampled with temperature $\text{T} = 0$). This prevents the verifier from introducing noise when its signal is ambiguous.

By adaptively switching between these modes, LatentReAct combines the verifier's error-correction capabilities with the generator's base reliability, improving overall trajectory quality without compromising the model's inherent reasoning.

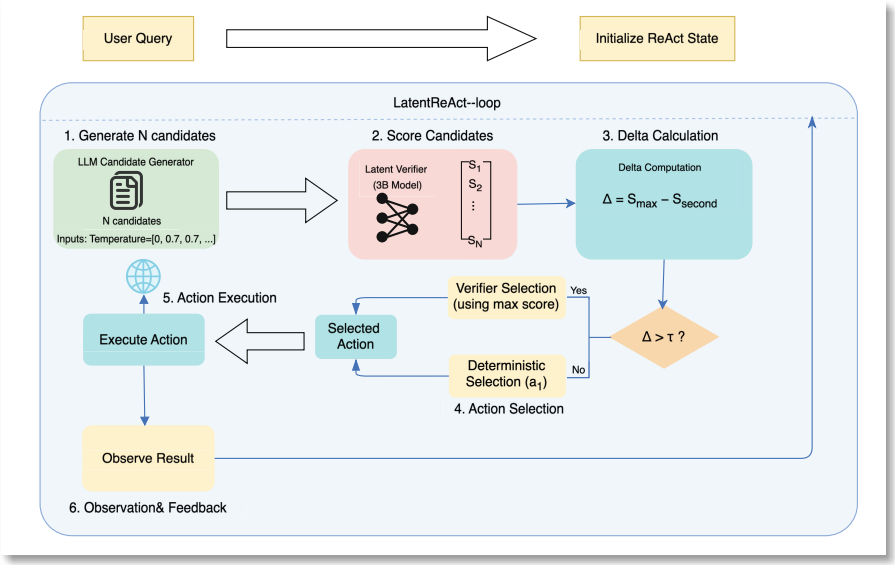

Fig. 2. LatentReAct in the ReAct loop. Candidate actions are scored by the verifier, and a confidence margin Δ determines whether to follow the verifier's top choice or fall back to the generator's default output.

4 Experiments

We evaluate LatentReAct on multi-step tool-augmented reasoning tasks, focusing on two questions: (1) Can the latent verifier effectively distinguish correct actions from erroneous ones? (2) Does integrating this verifier into the ReAct loop improve end-to-end task performance?

4.1 Datasets

Verifier Training To train the latent policy network, we use the ToolBench dataset. Since ToolBench originally contains ReAct trajectories with error-correction patterns, we extract high-quality pairwise training data without manual annotation. Specifically, we identify steps where an invalid tool call (negative sample a^-) is immediately followed by a corrective action (positive sample a^+). After filtering incomplete or corrupted sequences, we obtain 7,597 training pairs, 949 validation pairs, and 952 testing pairs.

End-to-End Evaluation For multi-turn evaluation, we use four challenging subsets from the Berkeley Function Calling Leaderboard (BFCL v4): base, miss_param, miss_func, and long_context. These subsets test agent robustness under realistic conditions—missing parameters, hallucinated functions, and extended contexts—directly aligning with our verification objectives.

4.2 Baselinesa

We compare LatentReAct against five baselines spanning both component-level and system-level evaluations:

Vanilla ReAct [4] The standard ReAct agent without verification, serving as our primary end-to-end baseline.

Zero-Shot Direct Prompting Directly prompts the 3B backbone to evaluate candidate actions in natural language.

Logit-Based Verification Uses raw output token probabilities (logits) of generated verdicts instead of latent features.

Single-Layer Representation Extracts features only from the final transformer layer (layer -1), ablating our intermediate pooling strategy.

Random Feature Trains the policy network on random Gaussian noise $\mathcal{N}(0, 1)$ as a sanity check to ensure performance gains stem from genuine semantic signals.

4.3 Evaluation Metrics

End-to-End Metrics Following established practice [4], we report:

Success Rate (SR): Percentage of queries correctly resolved within a maximum number of steps.

Average Trajectory Length (ATL): Mean number of steps taken; lower ATL indicates fewer redundant API calls.

Entry Accuracy (EA): A stricter metric requiring the entire multi-turn trajectory to be error-free.

Component-Level Metric For the verifier itself, we report Accuracy (Acc) on the ToolBench test set, measuring its ability to distinguish valid actions from erroneous ones.

4.4 Implementation Details

We use a frozen Qwen2.5-3B model as the backbone for both generation and verification. Latent features are extracted from layers 24–28 via mean pooling, as we empirically found these layers optimally capture reasoning semantics. The policy network is a lightweight MLP with multi-head attention, trained using the Adam optimizer (learning rate 1×10^{-4}, batch size 32).

We employ a pairwise margin ranking loss with margin $\gamma = 0.5$. During inference, $K = 3$ candidate actions are generated with temperature $T = 0.7$, to balance candidate diversity and computational cost, while the deterministic fallback action uses temperature $T = 0$. The confidence threshold τ is tuned on the validation set.

4.5 Main Results

End-to-End Performance Table 1 compares LatentReAct with Vanilla ReAct on four BFCL v4 subsets. LatentReAct consistently outperforms the baseline across all metrics.

Compared to the Vanilla ReAct baseline, Entry Accuracy (EA) improves by 3–5 percentage points. Turn Success Rate (SR) increases from approximately 89% to 97%–98%. Meanwhile, Average Trajectory Length (ATL) decreases by 12%–16% (e.g., from 3.55 to 2.98 on the miss_param subset).

Error analysis reveals that Vanilla ReAct failures are dominated by empty_turn_model_response errors (over 90% on the base subset), indicating severe hallucination or formatting collapse. LatentReAct intercepts these errors at the latent level, preventing error propagation and improving overall trajectory quality. Furthermore, it is worth noting that while Reflexion relies on multi-episode environment interactions (evaluated over 3 trials in our setup) to accumulate verbal reinforcement, LatentReAct achieves superior performance within a strictly single trial. This demonstrates that our performance gains stem from effective real-time internal state probing, rather than relying on extensive test-time trial-and-error, making LatentReAct significantly more efficient in practical API-calling scenarios.

Table 1. End-to-end results on BFCL v4. EA: Entry Accuracy (%); SR: Turn Success Rate (%); ATL: Average Trajectory Length (steps). Best results in bold.

Dataset	Method	EA (%)	SR (%)	ATL (steps)
Base	Vanilla ReAct	29.0	89.7	3.61
	Reflexion	**32.0**	88.5	3.68
	LatentReAct (Ours)	**32.0**	**97.3**	**3.16**
Long Context	Vanilla ReAct	28.0	89.7	3.61
	Reflexion	**31.0**	89.1	3.68
	LatentReAct(Ours)	29.0	**97.9**	**3.12**
Miss Func	Vanilla ReAct	30.0	90.0	3.42
	Reflexion	32.0	90.2	3.48
	LatentReAct (Ours)	**35.0**	**97.7**	**2.97**
Miss Param	Vanilla ReAct	35.0	89.3	3.55
	Reflexion	**37.0**	88.6	3.61
	LatentReAct (Ours)	36.0	**98.1**	**2.98**

Verifier Component Analysis To understand the source of these gains, we evaluate the latent verifier in isolation on the ToolBench test set. Table 2 compares its accuracy against several baselines.

Zero-shot direct prompting achieves only 51.17% accuracy, barely above random guessing (51.47%), indicating that raw LLM judgment is highly unreliable for this

task when elicited through simple prompting. Logit-based verification (61.03%) offers a modest improvement but still falls short of the performance achieved by latent representations.

Single-layer features (layer -1) achieve 72.90% accuracy, confirming that even surface-level representations contain useful signals. Our latent verifier using layers 24–28 achieves 74.47% accuracy with a higher average margin (0.4559), indicating more confident predictions.

These results validate our core hypothesis: intermediate hidden states encode richer evaluative semantics than final-layer outputs, which are constrained by vocabulary projection.

Table 2. Component-level verification accuracy on ToolBench. Avg. Margin reflects the confidence gap between positive and negative predictions.

Verifier Strategy	Test Acc (%) ↑	Avg. Margin
Zero-Shot Direct Prompting	51.17	–
Random Feature (Sanity Check)	51.47	0.0430
Logit-Based Verification	61.03	–
Single-Layer (Layer -1)	72.90	0.3741
Latent Verifier (Ours, Layers 24–28)	**74.47**	**0.4559**

Table 3. Ablation study of dimensional combinations on the ToolBench test set. Avg. Margin reflects the confidence gap between positive and negative predictions. Best results are highlighted in bold.

Dimension Combination	Test Acc (%) ↑	Avg. Margin
Full 5 Dimensions ($D_1 \sim D_5$)	**74.68**	**0.5099**
Subset$\{D_1, D_2, D_4\}$	74.58	0.4774
Subset $\{D_1, D_3, D_5\}$	70.48	0.4774
Subset $\{D_3, D_4, D_5\}$	63.45	0.3052
Latent Verifier (Ours, D_1, D_2, D_3)	74.47	0.4559

5 Analyses

5.1 Ablation Study: Feature Dimension Contribution

To better understand the contribution of each semantic dimension and justify our feature selection, we conduct an ablation study on the ToolBench test set. We define the five extracted heuristic dimensions as follows: logical redundancy (D_1), target effectiveness (D_2), format compliance (D_3), contextual redundancy (D_4; e.g., whether the

action merely repeats information from the dialogue history), and parameter correctness (D_5; e.g., verifying specific parameter values for validity). We then compare our adopted three-dimensional configuration ($\{D_1, D_2, D_3\}$) with the full feature set as well as several representative subsets.

As shown in Table 3, utilizing all five dimensions achieves an accuracy of 74.68%, which serves as an approximate upper bound. Notably, our selected configuration $\{D_1, D_2, D_3\}$ achieves a highly competitive accuracy of 74.47%. Despite reducing the feature dimensionality by 40%, this subset preserves 99.7% of the full-set performance, indicating that the essential discriminative signals are largely captured by these three dimensions.

Further analysis reveals that D_1 and D_2 act as the primary drivers of verification capability. Combinations that retain both dimensions, such as $\{D_1, D_2, D_4\}$, maintain accuracy above 74.5%. In contrast, removing these two key dimensions results in substantial performance degradation. For instance, the subset $\{D_3, D_4, D_5\}$ drops to 63.45% accuracy, with the average margin decreasing to 0.3052. This observation suggests that D_4 and D_5 contribute relatively limited independent signals and mainly introduce redundant information.

Overall, these results support two conclusions. First, our three-dimensional decomposition $\{D_1, D_2, D_3\}$ captures the core semantic signals required for reliable action verification. Second, additional dimensions introduce limited complementary information while increasing computational complexity. This justifies our design choice of adopting $\{D_1, D_2, D_3\}$ as the final feature set, achieving a favorable balance between accuracy and efficiency.

6 Conclusion

In this paper, we propose LatentReAct, a novel closed-loop reasoning framework that addresses the persistent issues of hallucination and redundant loops in LLM-based agents. Instead of relying on surface-level textual evaluation, our framework introduces a shared latent verifier that intercepts the generation process by extracting deep semantic features from intermediate transformer layers (Layers 24–28). By decoupling the evaluation into orthogonal dimensions—Logical Redundancy, Target Effectiveness, and Format Compliance—the system dynamically routes the reasoning trajectory using a threshold-based confidence margin.

Extensive experiments demonstrate that LatentReAct significantly outperforms the standard ReAct paradigm. On the challenging BFCL v4 benchmark, our framework elevates the Turn Success Rate to near-perfect levels (over 97%) while substantially reducing the average trajectory length. Furthermore, rigorous component-level ablation studies and sanity checks validate that intermediate hidden states retain exponentially richer evaluative signals than final logits. Ultimately, LatentReAct offers a highly robust, computationally efficient, and plug-and-play architectural enhancement for future autonomous agents.

References

1. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., et al.: Toolformer: language models can teach themselves to use tools. In: Advances in Neural Information Processing Systems 36 (NeurIPS 2023), pp. 68539–68551. Curran Associates, Inc., Red Hook, NY, USA (2023)
2. Qin Y, Liang S, Ye Y, Zhu K, Yan L, Lu Y, et al.: Toolllm: Facilitating Large Language Models To Master 16000+ Real-World Apis. arXiv preprint https://arxiv.org/abs/2307.16789, (2023)
3. OpenAI: Function Calling and Other API Updates. OpenAI Blog. https://openai.com/blog/function-calling-and-other-api-updates (2023). Accessed 03 May 2024
4. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., et al.: ReAct: synergizing reasoning and acting in language models. In: International Conference on Learning Representations (ICLR 2023) (2023)
5. Verma M, Bhambri S, Kambhampati S.: On the Brittle Foundations of ReAct Prompting for Agentic Large Language Models. arXiv preprint https://arxiv.org/abs/2405.13966, (2024)
6. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., et al.: Chain-of-thought prompting elicits reasoning in large language models. Adv. Neural Inf. Proces. Syst. **35**, 24824–24837 (2022)
7. Aksitov R, Miryoosefi S, Li Z, Li D, Babayan S, Kopparapu K, et al.: ReST Meets ReAct: Self-Improvement for Multi-step Reasoning LLM Agent. arXiv preprint https://arxiv.org/abs/2312.10003, (2023)
8. Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., Yao, S.: Reflexion: language agents with verbal reinforcement learning. In: Advances in Neural Information Processing Systems, vol. 36, (2023)
9. Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., et al.: Self-refine: iterative refinement with self-feedback. In: Advances in Neural Information Processing Systems, vol. 36, (2023)
10. Patil, S.G., Mao, H., Cheng-Jie Ji, C., Yan, F., Suresh, V., Stoica, I., et al.: The Berkeley function calling leaderboard (BFCL): from tool use to agentic evaluation of large language models. In: Forty-second International Conference on Machine Learning (ICML 2025) (2025)
11. Orgad, H., Toker, M., Gekhman, Z., Reichart, R., Szpektor, I., Kotek, H., et al.: LLMs know more than they show: on the intrinsic representation of LLM hallucinations. In: The Thirteenth International Conference on Learning Representations (ICLR 2025) (2025)
12. Wang G, Xie Y, Jiang Y, Mandlekar A, Xiao C, Zhu Y, et al.: Voyager: An Open-Ended Embodied Agent with Large Language Models. arXiv preprint https://arxiv.org/abs/2305.16291, (2023)
13. Tenney, I., Das, D., Pavlick, E.: BERT rediscovers the classical NLP pipeline. In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL 2019), pp. 4593–4601. Association for Computational Linguistics, Florence, Italy (2019)
14. Belinkov, Y.: Probing classifiers: promises, shortcomings, and advances. Comput. Linguist. **48**(1), 207–219 (2022)
15. Liu, N.F., Gardner, M., Belinkov, Y., Peters, M.E., Smith, N.A.: Linguistic knowledge and transferability of contextual representations. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019), pp. 1073–1094. Association for Computational Linguistics, Minneapolis, USA (2019)
16. Meng, K., Bau, D., Andonian, A., Belinkov, Y.: Locating and editing factual associations in GPT. In: Advances in Neural Information Processing Systems 35 (NeurIPS 2022), pp. 17359–17372. Curran Associates, Inc., Red Hook, NY, USA (2022)

17. Subramani, N., Suresh, N., Madotto, A.: Extracting latent steering vectors from pretrained language models. In: Findings of the Association for Computational Linguistics: ACL 2022, pp. 2308–2323. Association for Computational Linguistics, Dublin, Ireland (2022)
18. Azaria A, Mitchell T.: The Internal State of an LLM knows when it's Lying. arXiv preprint https://arxiv.org/abs/2304.13734, (2023)
19. Kadavath S, Conerly T, Askell A, et al.: Language Models (Mostly) Know What They Know. arXiv preprint https://arxiv.org/abs/2207.05221, (2022)
20. Gou, Z., Shao, Z., Gong, Y., Shen, Y., Yang, Y., Duan, N., et al.: CRITIC: large language models can self-correct with tool-interactive critiquing. In: The Twelfth International Conference on Learning Representations (ICLR 2024) (2024)