Internal Representations as Indicators of Hallucinations in Agent Tool Selection

Kait Healy*, Bharathi Srinivasan *, Visakh Madathil*, Jing Wu *

Amazon
410 Terry Avenue North, Seattle, WA 98109, United States

Abstract

Large Language Models (LLMs) have shown remarkable capabilities in tool calling and tool usage, but suffer from hallucinations where they choose incorrect tools, provide malformed parameters and exhibit 'tool bypass' behavior by performing simulations and generating outputs instead of invoking specialized tools or external systems. This undermines the reliability of LLM based agents in production systems as it leads to inconsistent results, and bypasses security and audit controls. Such hallucinations in agent tool selection require early detection and error handling. Unlike existing hallucination detection methods that require multiple forward passes or external validation, we present a computationally efficient framework that detects tool-calling hallucinations in real-time by leveraging LLMs' internal representations during the same forward pass used for generation. We evaluate this approach on reasoning tasks across multiple domains, demonstrating strong detection performance (up to 86.4% accuracy) while maintaining real-time inference capabilities with minimal computational overhead, particularly excelling at detecting parameter-level hallucinations and inappropriate tool selections, critical for reliable agent deployment.

Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities as autonomous agents, enabling them to interact with external APIs, execute code, and orchestrate complex workflows (Brown et al. 2020; Schick et al. 2023; Qin et al. 2024). However, these models exhibit a critical vulnerability: **tool-calling hallucinations**, where they generate structurally plausible but functionally incorrect tool calls. Unlike textual hallucinations, tool-calling hallucinations manifest as inappropriate tool selection, malformed parameters, incorrect tool chaining, tool bypass behavior, or semantically incorrect function invocations that can lead to system failures or data corruption.

The detection of tool-calling hallucinations presents unique challenges. Tool calls possess rigid structural and semantic constraints—parameters must conform to specific types and must be accurately interpreted from the query to the agent, required arguments cannot be omitted, and function names must exist in the available repertoire. Moreover, the consequences of undetected hallucinations can be severe in computational domains where precision is critical. Incorrect tool usage can also exacerbate security vulnerabilities of agents due to incorrect data access

To address these challenges, we propose a novel approach that leverages the internal representations of LLMs during tool call generation to detect hallucinations in real-time. Our method builds upon recent advances in understanding LLM internal states (Su et al. 2024; Azaria and Mitchell 2023) and extends them to structured tool calling. We introduce an unsupervised training framework that automatically generates labeled data by masking ground-truth tool calls, prompting LLMs to predict appropriate functions, and training lightweight classifiers on the resulting contextualized embeddings.

Our key contributions are:

- We demonstrate that internal representations of LLMs contain discriminative information for detecting tool-calling hallucinations in reasoning tasks.
- We evaluate our approach on tool calling scenarios, showing effective real-time detection capabilities with minimal computational overhead.

Related Work

Hallucination Detection in Large Language Models

Hallucination detection has emerged as a critical challenge in deploying LLMs for real-world applications. Early approaches focused on post-processing methods that analyze generated text for factual inconsistencies (Maynez et al. 2020; Zhou et al. 2020; Ji et al. 2023; Wu et al. 2025b,a; Wu, Hobbs, and Hovakimyan 2023). These methods typically require external knowledge sources or multiple model generations to assess consistency. Recent work has explored uncertainty-based approaches that leverage model confidence signals. Kadavath et al. (2022) demonstrated that LLMs can express uncertainty about their knowledge, while Zhang et al. (2023) proposed enhanced uncertainty estimation methods for hallucination detection. However, these approaches struggle with the discrete nature of tool calling, where traditional uncertainty measures may not capture semantic correctness of API usage. Consistency-based methods represent another major direction. SelfCheckGPT (Manakul, Liusie, and Gales 2023)

*These authors contributed equally.

generates multiple responses and checks for consistency, assuming that hallucinated content will be inconsistent across samples. Non Contradiction Probability (NCP) (Hou et al. 2025) and Semantic Similarity (Kuhn, Arakelyan, and Percha 2023) are two techniques using this principle of consistency to detect hallucinations. After sampling, NCP probabilistically scores whether a response contradicts entries in a curated belief set using information-theoretic scoring or belief tree propagation, while semantic similarity measures the degree of meaning alignment between a model's outputs from multiple samples using cosine similarity or nearest-neighbor score between them. Li et al. (2023a) introduced evaluation frameworks for assessing hallucination detection methods. While effective for free-form text, these approaches face limitations in tool calling scenarios where there may be unique correct solutions, making consistency-based detection less reliable.

Tool-Augmented Language Models

The integration of external tools with language models has gained significant attention as a means to extend LLM capabilities beyond the parametric knowledge acquired from their training data. Toolformer (Schick et al. 2023) pioneered self-supervised learning for tool use, demonstrating that LLMs can learn when and how to call APIs through minimal demonstrations. This work established the foundation for automated tool learning but did not address the detection of incorrect tool usage. ToolLLM (Qin et al. 2024) scaled tool learning to thousands of real-world APIs, introducing comprehensive training frameworks and evaluation benchmarks. Gorilla (Patil et al. 2024) focused specifically on API calling accuracy, achieving strong performance through retrieval-augmented training. ReAct (Yao et al. 2023) proposed reasoning and acting paradigms that interleave tool use with reasoning, while Lu et al. (2023) developed plug-and-play compositional reasoning systems. Despite these advances, existing work primarily focuses on improving tool-use accuracy during training rather than detecting errors during inference. Our work addresses this gap by providing real-time detection capabilities that can identify when models generate inappropriate tool calls.

Internal Representation Analysis

Understanding the internal mechanisms of neural networks has become increasingly important for building reliable AI systems. Azaria and Mitchell (2023) demonstrated that internal states of LLMs indicate the veracity of responses, showing that classifiers trained on hidden representations can detect when models produce false statements. This work provided crucial evidence that internal representations encode semantic properties beyond surface-level text generation. Building on this foundation, Su et al. (2024) developed unsupervised methods for real-time hallucination detection using contextualized embeddings from transformer layers. Their approach eliminated the need for manual annotation by automatically generating training data through entity masking in Wikipedia articles. Our work extends these insights to the structured domain of tool calling, where the challenges and patterns differ significantly from free-form text generation. Recent work has also explored mechanistic interpretability of transformers (Elhage et al. 2021; Wang et al. 2022), providing insights into how these models process and represent information. Meng et al. (2022) investigated knowledge storage and editing in transformers, while Li et al. (2023b) analyzed attention patterns in reasoning tasks. These insights inform our understanding of where tool-calling information might be encoded within transformer representations.

Agent Systems and Reliability

The deployment of LLM-based agents in real-world scenarios has highlighted the need for robust error detection and mitigation strategies. Xi et al. (2023) surveyed the landscape of autonomous agents, identifying reliability as a key challenge for practical deployment. Wang et al. (2023) demonstrated lifelong learning agents but noted the challenges of ensuring consistent performance across diverse environments. Recent work has explored agent evaluation frameworks. Liu et al. (2023) introduced comprehensive benchmarks for agent capabilities, while Xu et al. (2023) proposed evaluation metrics for tool-using agents. However, these evaluation frameworks primarily focus on task completion rather than real-time error detection during agent execution. Safety and alignment in agent systems have also received attention. Hendrycks, Mazeika, and Woodside (2023) discussed risks associated with autonomous AI systems, while Kenton et al. (2021) explored alignment challenges specific to agent deployment. Our work contributes to this area by providing mechanisms for detecting potentially harmful or incorrect agent actions before they impact external systems.

Evaluation Methodologies for tool calling

Benchmarking tool calling capabilities has become increasingly sophisticated as the field has matured. APIBench (Patil et al. 2024) established evaluation protocols for API calling accuracy, focusing on syntactic and semantic correctness of generated calls. ToolBench (Qin et al. 2024) expanded evaluation to include multi-step reasoning and complex tool interactions. Recent work has emphasized the importance of real-world evaluation scenarios. Tang et al. (2023) introduced evaluation datasets based on actual API usage patterns, while Ruan et al. (2023) explored evaluation in dynamic environments where API specifications change over time. These works highlight the challenges of tool calling evaluation but have not addressed real-time error detection during generation.

Problem Formulation

We formalize tool-calling hallucination detection as a binary classification task operating on the internal representations of the last layer of large language models (LLMs) during tool call generation. Let $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ denote the set of available functions in an agent's toolkit. Given a user query q and contextual information c, an LLM $\mathcal{M}$ generates a tool call $\tilde{f}(\boldsymbol{a})$ where $\tilde{f} \in \mathcal{F} \cup \{\emptyset\}$ and $\boldsymbol{a}$ represents the function arguments.

We define a tool-calling hallucination as occurring when any of the following conditions hold:

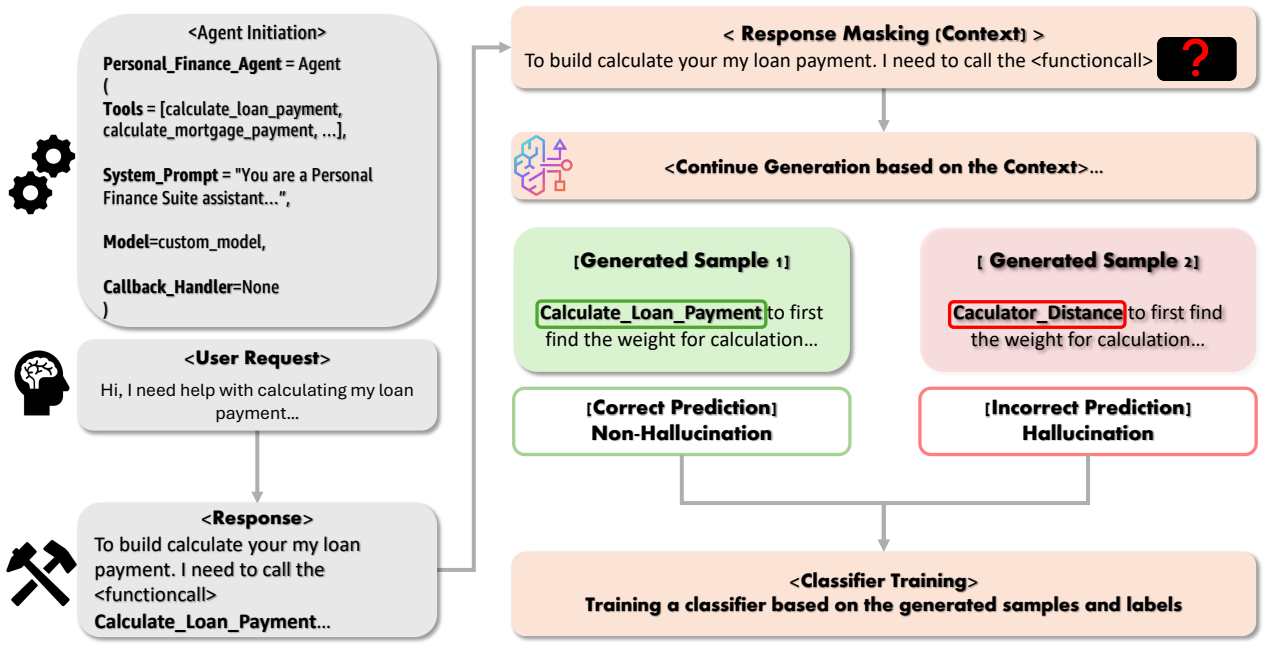

Figure 1: **Unsupervised, single-pass training pipeline for tool-calling hallucination detection.** Given a user query and available tools, the LLM generates candidate responses containing tool calls. We *mask* the tool-call segment to create a context-only prompt, ask the model to predict the call again, and assign labels by agreement with the reference call (non-hallucination vs. hallucination). We then train a lightweight classifier on last-layer representations to distinguish correct from hallucinated calls; at inference, the classifier scores each proposed call in real time and gates execution accordingly.

(1) **Function Selection Error**: $\tilde{f} \notin \mathcal{F}$ (invoking a non-existent function)

(2) **Function Appropriateness Error**: $\tilde{f} \in \mathcal{F}$ but $\tilde{f}$ is semantically inappropriate for query q given context c

(3) **Parameter Error**: $\boldsymbol{a} \notin \mathcal{D}_{\tilde{f}}$, where $\mathcal{D}_{\tilde{f}}$ denotes the valid argument domain for function $\tilde{f}$

(4) **Completeness Error**: Required parameters are missing from $\boldsymbol{a}$

(5) **Tool Bypass Error**: Generating the output without using $\tilde{f} \in \mathcal{F}$

Our objective is to learn a classifier $h_\theta : \mathbb{R}^d \to \{0, 1\}$ that maps the internal representation $\mathbf{z} \in \mathbb{R}^d$ of the LLM at tool call generation time to a binary decision, where 1 indicates a hallucinated tool call and 0 indicates a correct tool call. While we categorize hallucinations into five distinct types for theoretical completeness, our current work focuses on unified binary detection across all error categories, enabling the model to learn general patterns of incorrectness while maintaining practical deployment simplicity.

Method

We detect hallucinations in AI agent tool calling based on the hypothesis that the representations from the final transformer layer during tool call generation contain sufficient information to distinguish between correct and hallucinated calls.

Setup and Notation Let $\mathcal{D} = \{(q_i, c_i, f_i^*, \boldsymbol{a}_i^*, y_i)\}_{i=1}^N$ denote our collected tool calling instances, where q_i represents the user query, c_i the contextual information, f_i^* the ground-truth function, $\boldsymbol{a}_i^*$ the ground-truth arguments, and $y_i \in \{0, 1\}$ the binary label indicating whether the predicted tool call constitutes a hallucination. Given input (q_i, c_i), the LLM $\mathcal{M}$ generates a predicted call $\tilde{f}_i(\boldsymbol{a}_i)$. We extract features from the final layer hidden states of the same forward pass that produces this prediction.

Data Collection and Label Generation We utilize the Glaive Function-Calling dataset (GlaiveAI 2024) from Hugging Face, which comprises multi-domain agent interactions annotated with structured tool usage information and corresponding parameters, covering domains such as personal health, finance, and general utility calculations. We pre-process this dataset by grouping samples based on their designated tool names and normalizing argument structures to ensure consistency in both function and parameter representation. To evaluate tool-specific behaviors and hallucination rates, we develop specialized agents using the Strands framework (Agents 2025) that are programmatically matched to tool names and parameter schemas extracted from the pre-processed samples. These agents are then deployed to receive data samples filtered by tool assignment, enabling precise measurement of agent performance and tool invocation fidelity under controlled experimental conditions.

We collect agent interaction data from various reasoning domains spanning personal health calculations, financial computations, sustainability metrics, and general calculator func-

tions—domains where tool calling precision is critical for AI system accuracy. For each interaction $(q_i, c_i, f_i^*, \boldsymbol{a}_i^*)$, we obtain training labels through the following procedure: (1) remove the reference call $(f_i^*, \boldsymbol{a}_i^*)$ from the prompt while preserving (q_i, c_i), (2) prompt $\mathcal{M}$ to obtain agent response $\tilde{f}_i(\boldsymbol{a}_i)$, extract tool call if present and cache final-layer states from this forward pass, and (3) assign binary labels by comparing predicted and reference calls through function name matching and argument canonicalization. Specifically, a prediction receives label $y_i = 1$ (hallucinated) if the function name differs from the reference or if arguments fail to match after normalization; otherwise $y_i = 0$ (correct). This approach yields naturally occurring hallucination examples in computational domains where mathematical precision is paramount.

Final-Layer Feature Extraction We extract features from the final transformer layer hidden states of the forward pass that generated $\tilde{f}_i(\boldsymbol{a}_i)$. Let $\mathbf{h}_t^{(L)} \in \mathbb{R}^d$ denote the final-layer hidden state at token position t, where L is the number of transformer layers. We identify three critical token positions within the structured tool call: (1) t^{func}: the initial sub-token of the predicted function name, (2) $\mathcal{T}_{\text{args}} = \{t_1, t_2, \ldots, t_k\}$: all tokens spanning the argument region, and (3) t^{end}: the closing delimiter token (e.g., ")" or "}").

We construct a compact feature representation by concatenating representations from these three semantic regions:

$$\mathbf{z}_i = \Pi \left(\mathbf{h}_{t^{\text{func}}}^{(L)} \,\|\, \frac{1}{|\mathcal{T}_{\text{args}}|} \sum_{t \in \mathcal{T}_{\text{args}}} \mathbf{h}_t^{(L)} \,\|\, \mathbf{h}_{t^{\text{end}}}^{(L)} \right) \in \mathbb{R}^m \quad (1)$$

where $\|$ denotes vector concatenation and $\Pi : \mathbb{R}^{3d} \to \mathbb{R}^m$ represents an optional linear projection (identity mapping by default, yielding $m = 3d$).

Classifier Architecture and Training Objective We employ a lightweight feed-forward network to map feature vectors $\mathbf{z}_i$ to hallucination labels:

$$p_i = h_\theta(\mathbf{z}_i) = \sigma\left(\mathbf{w}_2^\top \phi(\mathbf{W}_1 \mathbf{z}_i + \mathbf{b}_1) + b_2\right) \in (0, 1) \quad (2)$$

where ϕ denotes the ReLU activation function, σ is the sigmoid function, and $\theta = \{\mathbf{W}_1, \mathbf{b}_1, \mathbf{w}_2, b_2\}$ are learnable parameters. Binary predictions follow $\hat{y}_i = \mathbf{1}\{p_i > \tau\}$ with threshold τ selected via validation set optimization.

The training objective employs standard binary cross-entropy loss:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log p_i + (1 - y_i) \log(1 - p_i)] \quad (3)$$

We train model-specific classifiers to account for architectural differences in internal representations across LLM families. Optional contextual features (e.g., available function specifications) can be incorporated via the projection layer Π. Post-training calibration using temperature scaling is applied on held-out validation data to ensure well-calibrated probability estimates.

Inference Protocol

Our method operates inline with generation, reusing the *identical forward pass* that produces the tool call:

1. The LLM generates tool call $\tilde{f}(\boldsymbol{a})$ through standard autoregressive generation
2. Extract token positions t^{func}, $\mathcal{T}_{\text{args}}$, and t^{end} from the final layer of this pass
3. Construct feature vector $\mathbf{z}$ according to Equation (1)
4. Compute hallucination probability $p = h_\theta(\mathbf{z})$ using the trained classifier
5. If $p > \tau$, execute predefined policy (execution blocking, user confirmation, fallback selection, or repair via re-prompting); otherwise proceed with normal tool execution

Since feature extraction requires only final-layer representations from a single forward pass, computational overhead remains negligible, enabling deployment in latency-sensitive real-time agent systems. All experimental evaluations in Section adhere to this inference protocol.

Experiments

Experimental Setup

Target Models We evaluate our approach on three representative open-source LLMs with diverse architectural characteristics:

- **Qwen7B**: A 7-billion parameter model employing Group Query Attention and RMSNorm, representing the Qwen model family's architectural innovations.
- **GPT-OSS-20B**: A 20-billion parameter model following the standard transformer architecture with multi-head attention and LayerNorm.
- **Llama-3.1-8B**: An 8-billion parameter model from the Llama family, incorporating RoPE positional embeddings and SwiGLU activation functions.

This selection provides coverage across different scales, architectural choices, and training methodologies to assess the generalizability of our approach.

Baseline Methods We compare against Non Contradiction Probability (NCP) (Hou et al. 2025) and semantic similarity (Kuhn, Arakelyan, and Percha 2023):

- **NCP** is the likelihood that a language model's output does not conflict with verified facts or logical premises. To calculate NCP, we prompt the agent with the same test sample multiple times (n=3) and measure consistency using agreement of function name and parameter.
- **Semantic Similarity** is another consistency technique measured using cosine similarity of responses from the agent over multiple invocations (n=3). Semantic similarity is calculated over the extracted tool call and parameters.

Both baselines require multiple forward passes (5× computational overhead) compared to our single-pass approach.

Table 1: Comparison with NCP and Semantic Similarity baselines on tool-calling hallucination detection. Our method achieves competitive performance while requiring only a single forward pass, compared to multiple sampling approach required for NCP and Semantic Similarity. Results show hallucination class performance (Precision/Recall/F1-Score/Accuracy).

Model	Method	Precision	Recall	F1-Score	Accuracy
GPT-OSS-20B	**Our Method**	**0.86**	**0.86**	**0.85**	**0.86**
	NCP	*1.0*	*0.79*	*0.88*	*0.95*
	Semantic Similarity	*1.0*	*0.79*	*0.88*	*0.95*
Llama-3.1-8B	**Our Method**	**0.73**	**0.73**	**0.72**	**0.73**
	NCP	*1.0*	*0.733*	*0.846*	*0.935*
	Semantic Similarity	*1.0*	*0.731*	*0.845*	*0.935*
Qwen-7B	**Our Method**	**0.81**	**0.74**	**0.77**	**0.74**
	NCP	*0.99*	*0.45*	*0.62*	*0.94*
	Semantic Similarity	*1.0*	*0.44*	*0.62*	*0.94*

Dataset Construction Our evaluation focuses on mathematical reasoning domains where tool-calling precision is critical. We developed five specialized AI agents using the Glaive dataset (gla 2024) as the foundation:

Agent Categories:

- **Quick Calculator**: Basic arithmetic operations, unit conversions, and mathematical computations
- **Personal Finance Suite**: Investment calculations, loan computations, and financial planning tools
- **Health Assistant**: BMI calculations, caloric needs estimation, and health metric tracking
- **Sustainability Assistant**: Carbon footprint calculations, energy efficiency metrics, and environmental impact assessments
- **Digital Commerce Assistant**: Price calculations, discount computations, and transaction processing

Data Processing: From the Glaive dataset's multi-turn conversations, we extracted 2,411 tool-calling instances per model. Each instance contains: (1) user query, (2) contextual information, (3) ground-truth function call, and (4) predicted function call with cached final-layer representations. Tool names and parameters were canonicalized to ensure consistent evaluation across different syntactic representations.

Training Configuration For each target model, we train a dedicated two-layer MLP classifier with model-specific input dimensions: 3584 for Qwen7B, 2880 for GPT-OSS-20B, and 4096 for Llama-3.1-8B. The classifier architecture consists of a hidden layer with 512 units and ReLU activation, followed by a single sigmoid output unit for binary classification. We apply dropout with probability 0.1 to the hidden layer during training to prevent overfitting.

We optimize the classifier using AdamW with a learning rate of 1×10^{-4} and weight decay of 1×10^{-5}. Training proceeds for up to 50 epochs with early stopping (patience=5) based on validation loss. We use a batch size of 32 samples and employ cosine annealing with warm restarts for the learning rate schedule to improve convergence stability.

Our evaluation protocol follows a 60%/20%/20% train/-validation/test split. The decision threshold τ is selected via grid search over the range [0.1, 0.9] with step size 0.05, optimizing for balanced performance on the validation set. Post-training calibration using temperature scaling is applied on the held-out validation set to ensure well-calibrated probability estimates. We report standard classification metrics including precision, recall, F1-score, and accuracy for both classes, along with macro and weighted averages to account for class imbalance effects.

Baseline Comparison

We compare our approach against NCP and semantic similarity, methods for detecting hallucinations in black-box LLMs. Both the baselines operate on the principle that consistent facts align across multiple samplings, while hallucinated facts diverge. We adapt this approach to tool-calling scenarios by generating multiple tool call samples and measuring consistency.

The results show that while non-contradiction probability and semantic similarity baselines achieve higher precision and overall accuracy, our method based on internal LLM states provides a competitive and robust approach to hallucination detection with higher recall over the baselines across all models—especially noteworthy for GPT-OSS-20B, where recall is highest among all methods (0.86 vs. 0.74 for Qwen-7B and 0.73 for Llama-3.1-8B), showing greater sensitivity to diverse hallucination types.

Internal state methods enable inference-time hallucination detection, operating directly within the LLM's decoding process, which eliminates the need for expensive sampling or repeated passes over the output. Unlike external-sampling such as NCP and semantic similarity or post-hoc semantic scoring, our method leverages information that is only available during stepwise generation—unlocking model-specific heuristics unreachable by black-box techniques. This makes our technique more adaptable for streaming, edge, and low-latency production scenarios, where fast and actionable hallucination risk scores are needed.

Table 2: Detailed classification results for GPT-OSS-20B, Llama-3.1-8B and Qwen7-B using our final-layer feature extraction approach.

Model	Class	Precision	Recall	F1-Score	Support
GPT-OSS-20B	Non-Hall.	0.87	0.97	0.92	1824
	Hall.	0.86	0.53	0.66	587
	Macro avg	0.86	0.75	0.79	2411
	Weighted avg	0.86	0.86	0.85	2411
	Accuracy		*0.86*		
Llama-3.1-8B	Non-Hall.	0.73	0.82	0.77	1375
	Hall.	0.71	0.61	0.66	1036
	Macro avg	0.72	0.71	0.71	2411
	Weighted avg	0.73	0.73	0.72	2411
	Accuracy		*0.73*		
Qwen-7B	Non-Hall.	0.91	0.76	0.83	1667
	Hall.	0.34	0.62	0.44	333
	Macro avg	0.63	0.69	0.64	2000
	Weighted avg	0.81	0.74	0.77	2000
	Accuracy		*0.74*		

Main Results

Table 2 presents our approach's performance on GPT-OSS-20B, Llama-3.1-8B and Qwen-7B, which demonstrate the effectiveness of internal representation-based hallucination detection.

GPT-OSS-20B achieves strong overall performance with 86% accuracy and demonstrates effective hallucination detection with 86% precision and 53% recall. The model maintains high precision in non-hallucination detection (87%) while achieving balanced macro-averaged performance (F1=0.79). Llama-3.1-8B shows more balanced performance across both classes, with nearly equal class distribution (1375:1036 ratio) and consistent precision (73% vs 71%) for both hallucination and non-hallucination detection.

The results demonstrate that internal transformer representations contain sufficient information to distinguish between correct and hallucinated tool calls. The computational efficiency of our approach—requiring only final-layer representations from a single forward pass—makes it suitable for real-time deployment in production agent systems.

Ablation Study on Feature Extraction Methods

To understand the impact of different feature extraction strategies on hallucination detection performance, we conduct a comprehensive ablation study examining various approaches to aggregate transformer hidden states into fixed-size representations for classification. We conduct all of the experiments on Qwen-7B.

Feature Extraction Methods We evaluate ten distinct feature extraction methods, each representing different strategies for converting variable-length sequence representations into fixed-size feature vectors:

Basic Pooling Methods:

- **Last-Layer Representations**: Mean pooling across the sequence dimension of the final transformer layer, producing a fixed-size representation equal to the model's hidden dimension.
- **Max Pooling**: Element-wise maximum across the sequence dimension, capturing the most activated features for each dimension.
- **Min Pooling**: Element-wise minimum across the sequence dimension, identifying consistently low-activated features.

Token-Specific Methods:

- **CLS Token**: Uses only the [CLS] token representation from the final layer, leveraging the token specifically designed for classification tasks.
- **Last Token**: Extracts features from the final token in the sequence, capturing end-of-sequence information.

Statistical Aggregation Methods:

- **Statistical**: Concatenates mean and standard deviation across the sequence dimension, doubling the feature dimensionality ($2d$ features).
- **Statistical Extended**: Combines mean, standard deviation, minimum, and maximum statistics, quadrupling the feature dimensionality ($4d$ features).

Advanced Aggregation Methods:

- **Attention-Weighted Pooling**: Computes attention weights based on L2 norms and performs weighted averaging across the sequence.
- **First-Last-Mean**: Concatenates the first token, last token, and mean-pooled representations, tripling the feature dimensionality ($3d$ features).

Table 3: Ablation study on feature extraction methods for hallucination detection using Qwen-7B.

Method	Acc.	AUC	Prec.	F1	Dim.	Time Complexity
Last-Layer Rep.	**0.746**	**0.721**	**0.749**	**0.748**	d	$O(n \cdot d)$
Statistical	0.745	0.719	0.748	0.746	$2d$	$O(n \cdot d)$
Attention Weighted	0.744	0.719	0.748	0.746	d	$O(n^2 \cdot d)$
Last Token	0.744	0.720	0.747	0.745	d	$O(1)$
First-Last-Mean	0.743	0.719	0.747	0.744	$3d$	$O(n \cdot d)$
Statistical Ext.	0.743	0.718	0.746	0.744	$4d$	$O(n \cdot d)$
CLS Token	0.742	0.719	0.746	0.744	d	$O(1)$
Multi-Scale	0.742	0.718	0.745	0.743	$5d$	$O(n \cdot d)$
Max Pooling	0.741	0.717	0.744	0.742	d	$O(n \cdot d)$
Min Pooling	0.739	0.716	0.743	0.741	d	$O(n \cdot d)$

- **Multi-Scale**: Combines multiple pooling strategies (mean, max) with positional tokens (first, middle, last), creating a comprehensive representation ($5d$ features).

We evaluate each feature extraction method on the Qwen-7B using the digital commerce assistant dataset. All experiments use identical training configurations, including the same train-test split (70-30), optimization parameters, and evaluation metrics to ensure fair comparison.

Our ablation study reveals several key insights for hallucination detection in language models:

1. **Simple aggregation methods are effective**: Mean pooling across the sequence dimension provides the best balance of performance and simplicity.
2. **Diminishing returns of complexity**: More sophisticated aggregation methods with higher dimensionality do not necessarily yield better performance, suggesting that the base transformer representations already capture the essential information.
3. **Sequence-level information is valuable**: Methods that aggregate information across the entire sequence consistently outperform single-token approaches, indicating that hallucination signals are distributed throughout the generated text.
4. **Robustness across methods**: The relatively small performance variance across methods suggests that the underlying transformer representations are robust and that multiple aggregation strategies can effectively capture hallucination patterns.

Based on these findings, we recommend using last-layer representations (mean pooling) as the default feature extraction method for transformer-based hallucination detection, as it provides optimal performance while maintaining computational efficiency and implementation simplicity.

Conclusion and Limitations

In this paper, we presented a novel approach for detecting tool-calling hallucinations in LLMs by leveraging their internal representations, demonstrating strong performance across multiple models with accuracies ranging from 72.7% to 86.4%. Our method's effectiveness is particularly notable in GPT-OSS-20B, which achieved 86% precision in hallucination detection while maintaining balanced performance across classes. The approach's ability to operate in real-time using only last-layer representations from a single forward pass makes it practically viable for deployment in production environments. Future work will investigate the possibility of developing a unified hallucination detector that works effectively across different model families, potentially identifying common patterns in internal representations that generalize across architectures.

However, reference-agreement labeling inherits problems from dataset calls and standardization rules, and determining function equivalence beyond string matching remains difficult without additional context. While our three-point feature design is intentionally simple, future work could explore field-aware pooling or lightweight attention over argument spans while keeping single-pass requirements, alongside expanded experiments, accuracy analysis, and robustness studies across new tools.

References

2024. Glaive Function Calling Dataset v2. https://huggingface.co/datasets/glaiveai/glaive-function-calling-v2. Accessed: 2025-09-30.

Agents, S. 2025. Strands Agents: Model-driven AI agent framework. https://strandsagents.com/latest/. Accessed: 2025-10-31.

Azaria, A.; and Mitchell, T. 2023. The internal state of an llm knows when its lying. *arXiv preprint arXiv:2304.13734*.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901.

Elhage, N.; Nanda, N.; Olsson, C.; Henighan, T.; Joseph, N.; Mann, B.; Askell, A.; Bai, Y.; Chen, A.; Conerly, T.; et al. 2021. A mathematical framework for transformer circuits. *Anthropic*.

GlaiveAI. 2024. Glaive Function Calling v2. https://huggingface.co/datasets/glaiveai/glaive-function-calling-v2. Accessed: 2025-10-31.

Hendrycks, D.; Mazeika, M.; and Woodside, T. 2023. An overview of catastrophic AI risks. *arXiv preprint arXiv:2306.12001*.

Hou, B.; Zhang, Y.; Andreas, J.; and Chang, S. 2025. A Probabilistic Framework for LLM Hallucination Detection via Belief Tree Propagation. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 3076–3099.

Ji, Z.; Lee, N.; Frieske, R.; Yu, T.; Su, D.; Xu, Y.; Ishii, E.; Bang, Y. J.; Madotto, A.; and Fung, P. 2023. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12): 1–38.

Kadavath, S.; Conerly, T.; Askell, A.; Henighan, T.; Drain, D.; Perez, E.; Schiefer, N.; Hatfield-Dodds, Z.; DasSarma, N.; Tran-Johnson, E.; et al. 2022. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*.

Kenton, Z.; Everitt, T.; Weidinger, L.; Gabriel, I.; Mikulik, V.; and Irving, G. 2021. Alignment of language agents. *arXiv preprint arXiv:2103.14659*.

Kuhn, F.; Arakelyan, K.; and Percha, B. 2023. Semantic INconsistency Index for Hallucination Detection in LLMs. *arXiv preprint arXiv:2503.05980*.

Li, J.; Cheng, X.; Zhao, W. X.; Nie, J.-Y.; and Wen, J.-R. 2023a. HaluEval: A large-scale hallucination evaluation benchmark for large language models. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 6449–6464.

Li, K.; Hopkins, A. K.; Bau, D.; Viégas, F.; Pfister, H.; and Wattenberg, M. 2023b. Inference-time intervention: Eliciting truthful answers from a language model. *arXiv preprint arXiv:2306.03341*.

Liu, X.; Yu, H.; Zhang, H.; Xu, Y.; Lei, X.; Lai, H.; Gu, Y.; Ding, H.; Men, K.; Yang, K.; et al. 2023. AgentBench: Evaluating LLMs as agents. *arXiv preprint arXiv:2308.03688*.

Lu, P.; Peng, B.; Cheng, H.; Galley, M.; Chang, K.-W.; Wu, Y. N.; Zhu, S.-C.; and Gao, J. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842*.

Manakul, P.; Liusie, A.; and Gales, M. J. 2023. SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models. *arXiv preprint arXiv:2303.08896*.

Maynez, J.; Narayan, S.; Bohnet, B.; and McDonald, R. 2020. On faithfulness and factuality in abstractive summarization. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 1906–1919.

Meng, K.; Bau, D.; Andonian, A.; and Belinkov, Y. 2022. Locating and editing factual associations in GPT. *Advances in Neural Information Processing Systems*, 35: 17359–17372.

Patil, S. G.; Zhang, T.; Wang, X.; and Gonzalez, J. E. 2024. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, volume 37.

Qin, Y.; Liang, S.; Ye, Y.; Zhu, K.; Yan, L.; Lu, Y.; Lin, Y.; Cong, X.; Tang, X.; Qian, B.; Zhao, S.; Hong, L.; Tian, R.; Xie, R.; Zhou, J.; Gerstein, M.; Li, D.; Liu, Z.; and Sun, M. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Ruan, J.; Chen, Y.; Zhang, B.; Xu, Z.; Bao, T.; Du, G.; Shi, S.; Mao, H.; Zeng, X.; and Zhao, R. 2023. TPTU: Task planning and tool usage of large language model-based AI agents. *arXiv preprint arXiv:2308.03427*.

Schick, T.; Dwivedi-Yu, J.; Dess'ı, R.; Raileanu, R.; Lomeli, M.; Zettlemoyer, L.; Cancedda, N.; and Scialom, T. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36.

Su, W.; Wang, C.; Ai, Q.; Hu, Y.; Wu, Z.; Zhou, Y.; and Liu, Y. 2024. Unsupervised real-time hallucination detection based on the internal states of large language models. *arXiv preprint arXiv:2403.06448*.

Tang, Q.; Deng, Z.; Lin, H.; Han, X.; Liang, Q.; and Sun, L. 2023. ToolAlpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Wang, K.; Variengien, A.; Conmy, A.; Shlegeris, B.; and Steinhardt, J. 2022. Interpretability in the wild: A circuit for indirect object identification in GPT-2 small. *arXiv preprint arXiv:2211.00593*.

Wu, J.; Chen, S.; Gutfraind, A.; Heo, I.; Liu, S.; Li, C.; Curuksu, J.; and Sharps, M. 2025a. Building more accountable multi-modal LLMs through spatially-informed visual reasoning. In *NeurIPS 2025 Workshop on Evaluating the Evolving LLM Lifecycle: Benchmarks, Emergent Abilities, and Scaling*.

Wu, J.; Chen, S.; Heo, I.; Gutfraind, S.; Liu, S.; Li, C.; Srinivasan, B.; Zhang, X.; and Sharps, M. 2025b. Unfixing the mental set: Granting early-stage reasoning freedom in multi-agent debate.

Wu, J.; Hobbs, J.; and Hovakimyan, N. 2023. Hallucination improves the performance of unsupervised visual representation learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 16132–16143.

Xi, Z.; Chen, W.; Guo, X.; He, W.; Ding, Y.; Hong, B.; Zhang, M.; Wang, J.; Jin, S.; Zhou, E.; et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*.

Xu, Y.; Liu, H.; Xu, Q.; Liang, H.; Zhou, Y.; Zhang, X.; Hu, Y.; Huang, Z.; Liang, K.; Liu, Z.; et al. 2023. Lemur: Harmonizing natural language and code for language agents. *arXiv preprint arXiv:2310.06830*.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Zhang, T.; Qiu, L.; Guo, Q.; Deng, C.; Zhang, Y.; Zhang, Z.; Zhou, C.; Wang, X.; and Fu, L. 2023. Enhancing uncertainty-based hallucination detection with stronger focus. *arXiv preprint arXiv:2311.13230*.

Zhou, C.; Neubig, G.; Gu, J.; Diab, M.; Guzman, P.; Zettlemoyer, L.; and Ghazvininejad, M. 2020. Detecting hallucinated content in conditional neural sequence generation. *Findings of the Association for Computational Linguistics: EMNLP 2020*, 1421–1436.