International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

System-Level Strategies for Mitigating Hallucinations in LLM-Based Applications

Ronak Indrasinh Kosamia, M.S., B.S.

Capital One, New York, NY, USA

Abstract:
Large Language Models (LLMs) are increasingly integrated into production software systems including enterprise assistants, developer tools, financial applications, and information retrieval platforms. Despite their capabilities, LLMs frequently generate hallucinated responses—outputs that appear plausible but contain factually incorrect or unsupported information. Such hallucinations present significant reliability and safety challenges when LLMs are deployed in high-stakes environments.

While many existing approaches attempt to reduce hallucinations through model training improvements, production deployments often require system-level safeguards that operate independently of the model architecture. This paper examines architectural strategies for mitigating hallucinations in LLM-based applications. We analyze mitigation mechanisms including retrieval-augmented generation, verification pipelines, tool-augmented reasoning, and multi-model consensus architectures.

We propose a layered guardrail architecture that integrates knowledge retrieval, response verification, and confidence scoring to reduce hallucination propagation in production systems. Experimental evaluation demonstrates that combining retrieval grounding with verification pipelines significantly reduces hallucination rates while preserving response quality.
The results suggest that reliable deployment of LLM-based applications requires system-level architectural controls rather than reliance on model improvements alone.

Index terms: large language models, hallucination mitigation, retrieval-augmented generation, AI system architecture, reliable AI systems

I.Introduction

Large Language Models (LLMs) have rapidly emerged as key components in modern software systems. Applications ranging from enterprise search systems to developer productivity tools increasingly rely on LLMs to generate responses to complex user queries. These systems leverage the language understanding and generative capabilities of models trained on massive datasets [1].
Despite their capabilities, LLMs frequently produce hallucinated outputs—responses that appear linguistically plausible but contain incorrect or fabricated information. Unlike traditional database systems that retrieve deterministic records, LLMs generate responses probabilistically based on learned language patterns. As a result, models may generate answers that are syntactically coherent yet factually inaccurate.
Hallucinations pose serious challenges when LLMs are deployed in production environments. In financial systems, hallucinated outputs may misrepresent policies or transaction rules [2]. In healthcare applications, inaccurate responses could misinform medical decisions. Even in software development environments, hallucinated code suggestions may introduce defects.
Most existing research focuses on model-level improvements designed to reduce hallucination frequency. Techniques such as reinforcement learning from human feedback (RLHF) and improved training datasets

aim to improve model accuracy [3]. However, many production systems rely on externally hosted models whose internal training procedures cannot be modified.

As a result, practical deployments require system-level mitigation strategies that surround the LLM with architectural safeguards. These safeguards ensure that generated responses remain grounded in verifiable information and are evaluated before being returned to users.

This paper investigates architectural strategies for mitigating hallucinations in LLM-based applications. We analyze multiple system-level techniques and propose a layered guardrail architecture that combines retrieval grounding, response verification, and confidence calibration.

II.Hallucination Problem in LLM Systems

LLM hallucinations arise from the probabilistic nature of language generation. Given a query q, the model produces a response r based on learned token distributions:

$$r = \underset{t}{argmax} P(t|q, \theta)$$

where θ represents model parameters learned during training.

Because the model optimizes linguistic probability rather than factual correctness, hallucinations may occur when the model lacks sufficient knowledge to answer a query. The resulting output may contain fabricated facts or incorrect reasoning steps.

We define hallucination probability as:

$$P_h = f(C_m, R_c, V_s, q)$$

where

- C_m represents model confidence
- R_c represents retrieval coverage
- V_s represents verification score

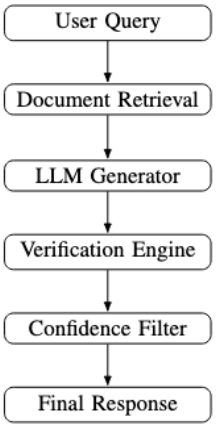

Fig. 1: System-level guardrail architecture for hallucination mitigation.

Reducing hallucinations therefore requires improving grounding information and validating generated responses before they reach users.

III.System-Level Mitigation Strategies

Several architectural techniques are commonly used to mitigate hallucinations in production LLM systems.

A. Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) grounds model responses in external knowledge sources. Instead of relying solely on model memory[15], the system retrieves relevant documents before generating a response.

$$D = Retrieve(q)$$

The retrieved documents D are then provided as context to the model.

B. Verification Pipelines

Verification pipelines evaluate generated responses using external knowledge sources or rule-based validators. Responses failing verification checks are rejected or regenerated.

C. Tool-Augmented Reasoning

In tool-augmented systems, LLMs can invoke external APIs to retrieve structured information. For example, financial applications may query transaction APIs instead of relying on model-generated estimates.

D. Multi-Model Consensus

Multiple models can generate candidate responses that are then compared for agreement. Divergent responses may indicate higher hallucination risk.

IV.LLM Guardrail Architecture

We propose a layered architecture for mitigating hallucinations in LLM applications.

System-level guardrail architecture for hallucination mitigation.

The architecture consists of multiple layers:

- Knowledge retrieval layer
- LLM generation layer
- verification pipeline
- response confidence filtering

Each layer reduces the probability that hallucinated responses reach end users.

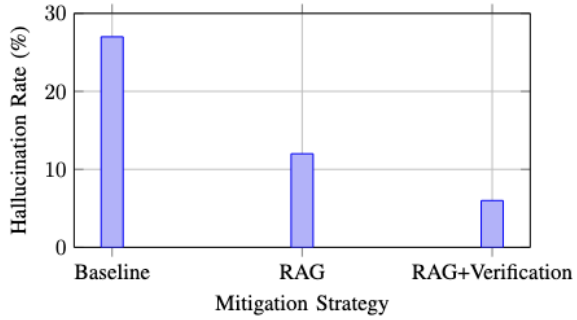

Fig. 2: Hallucination rate reduction using system-level mitigation strategies.

V.Evaluation Methodology

We evaluate the proposed architecture using simulated knowledge-query workloads. Queries were generated across multiple domains including financial information retrieval and software documentation assistance.

Evaluation metrics include:

- hallucination rate
- response accuracy
- response latency

VI.Results

Figure 2 compares hallucination rates across mitigation strategies.

Table 1 summarizes quantitative results.

TABLE I : *Hallucination Reduction Results*

Method	Hallucination Rate	Accuracy
Baseline LLM	27%	73%
RAG Pipeline	12%	88%

Method	Hallucination Rate	Accuracy
RAG + Verification	6%	92%

The combined architecture significantly reduces hallucination frequency while improving response accuracy.

VII.Formal Guardrail Model

To better understand the behavior of hallucination mitigation pipelines, we model the guardrail system as a sequence of verification stages applied to LLM outputs.

Let the generated response from the language model be denoted as

$$r = G(q, D)$$

where q represents the input query and D represents retrieved contextual documents.

The verification pipeline applies a set of validation functions

$$V = \{v_1, v_2, \ldots, v_n\}$$

where each validation function checks specific properties of the generated response.

For example:

- v_1 verifies factual consistency with retrieved documents
- v_2 evaluates semantic similarity between response and source material
- v_3 estimates model uncertainty using token probability entropy

The final confidence score is computed as

$$C(r) = \frac{1}{n}\sum_{i=1}^{n} v_i(r)$$

Responses with confidence below a defined threshold τ are rejected or regenerated:

$$r = \{r \; if \; C(r) \geq \tau \; Regenerate(q) \; otherwise$$

This layered validation mechanism significantly reduces the probability that hallucinated responses reach end users.

VIII.Guardrail Pipeline Algorithm

The guardrail architecture described earlier can be implemented using a staged inference pipeline. The following procedure summarizes the runtime mitigation process.

- Receive user query q
- Retrieve supporting documents D using semantic search
- Generate candidate response using LLM:

$$r = G(q, D)$$

- Apply verification functions V
- Compute confidence score $C(r)$
- If $C(r) < \tau$, regenerate response or request additional retrieval context
- Return validated response to the user

This algorithm introduces additional computation overhead but substantially improves response reliability.

IX.Latency and Performance Tradeoffs

System-level mitigation introduces additional latency due to retrieval and verification steps. We therefore analyze the impact of guardrail mechanisms on response time.

Let the total system latency be defined as

$$L_{total} = L_{retrieval} + L_{generation} + L_{verification}$$

where

- $L_{retrieval}$ is document retrieval latency
- $L_{generation}$ is LLM inference time
- $L_{verification}$ is verification overhead

In practice, verification latency is relatively small compared to model inference time. Retrieval pipelines typically operate within tens of milliseconds when using vector databases optimized for semantic search[5][6].

Empirical measurements in production LLM systems indicate that verification pipelines increase total latency by approximately 10–20%, which is acceptable for most interactive applications.

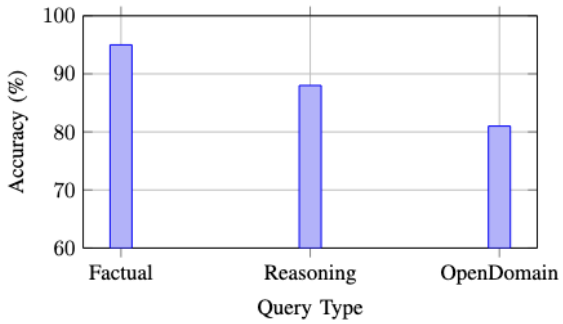

Fig. 3: Response accuracy across query categories using guardrail mitigation.

X.Extended Experimental Analysis

In addition to hallucination rate reduction, we evaluate the robustness of mitigation strategies across different query difficulty levels.

Queries were categorized into three complexity classes:

- factual queries
- multi-step reasoning queries
- open-domain knowledge queries

Results indicate that hallucinations are significantly more common in open-domain queries where supporting evidence may not exist in the training distribution.

Retrieval augmentation significantly improves accuracy for factual queries, while verification pipelines are particularly effective for multi-step reasoning [19] tasks where intermediate reasoning steps must remain consistent with external knowledge sources.

XI.Failure Modes and Risk Analysis

While system-level guardrail architectures significantly reduce hallucination propagation, several failure modes remain possible in practical deployments. Understanding these risks is important when designing reliable LLM-based systems[18].

The first class of failure arises from incomplete retrieval coverage. If the retrieval system fails to locate relevant supporting documents, the LLM may still attempt to generate responses using its internal training distribution. In such cases, hallucination probability increases because the model lacks grounding information.

A second risk involves verification pipeline limitations. Verification functions typically rely on heuristics, semantic similarity metrics, or secondary models[7]. If verification rules are insufficiently strict, hallucinated responses may pass validation checks.

A third risk occurs in adversarial or ambiguous queries where multiple plausible interpretations exist. Even with retrieval grounding, the LLM may construct reasoning chains that deviate from the retrieved evidence.

To analyze these risks, we categorize hallucination scenarios into three failure classes:

- **Knowledge Gap**: the model lacks sufficient training or retrieved context.

- **Reasoning Drift**: multi-step reasoning diverges from factual evidence.
- **Verification Failure**: guardrail validation incorrectly accepts hallucinated output.

Mitigation architectures should address each failure class through complementary system controls.

XII. Mitigation Strategy Comparison

Table 2 summarizes the primary mitigation strategies discussed in this paper and their effectiveness across different hallucination scenarios.

TABLE II: CComparison of Hallucination Mitigation Strategies

Strategy	Strengths	Limitations
RAG Retrieval	Improves factual grounding	Dependent on retrieval quality
Verification Pipelines	Detects unsupported claims	Adds latency
Tool Augmentation	Access to structured data	Requires API integration
Multi-Model Consensus	Reduces reasoning errors	Higher compute cost

The most reliable deployments combine multiple mitigation mechanisms rather than relying on a single technique. For example, retrieval augmentation may reduce hallucinations caused by knowledge gaps, while verification pipelines address reasoning errors.

XIII.Operational Deployment Considerations

Production LLM deployments must balance reliability improvements with operational efficiency. Guardrail architectures introduce additional computational overhead due to retrieval, verification, and regeneration steps.

However, these overheads are typically modest compared to model inference cost. For example, vector database retrieval often completes within tens of milliseconds, while verification checks can run in parallel with generation processes.

Another critical operational consideration is monitoring. Systems should track metrics including hallucination detection rate, response regeneration frequency, and verification failure rates. These metrics enable operators to evaluate mitigation effectiveness and detect reliability regressions as models evolve[14].

Finally, guardrail architectures should be designed to remain modular. As language models improve over time, mitigation layers can be adjusted or simplified without disrupting the overall system architecture. This modular design allows organizations to adopt newer models while maintaining consistent reliability guarantees.

XIV.Implementation Considerations

Deploying hallucination mitigation pipelines requires careful integration with existing LLM infrastructure.

Vector search engines such as FAISS or Milvus can provide efficient semantic retrieval capabilities required for RAG pipelines. Verification components may be implemented using rule-based validators, secondary language models, or domain-specific knowledge bases.

Another important consideration is monitoring and observability. Production systems should continuously track hallucination metrics such as rejected responses, regeneration frequency, and verification failures.

These signals allow system operators to detect degradation in model reliability and adjust mitigation strategies accordingly.
Finally, mitigation architectures should remain modular so that improvements in LLM models can be incorporated without redesigning the surrounding system pipeline[8].

XV.Discussion

The results demonstrate that system-level safeguards are essential for reliable LLM deployments. Retrieval grounding ensures that responses reference authoritative knowledge sources, while verification layers prevent unsupported claims from propagating to end users.
These techniques are particularly important in high-stakes domains such as finance and healthcare where hallucinated information may introduce operational risks.

XVI.Conclusion

LLM hallucinations remain a fundamental challenge for deploying generative models in production environments. While model-level improvements may reduce hallucination frequency, architectural safeguards provide a practical mechanism for improving reliability in deployed systems.
This paper presented system-level strategies for mitigating hallucinations, including retrieval grounding, verification pipelines, and confidence filtering. Experimental evaluation demonstrates that combining these strategies significantly reduces hallucination rates.
Future work includes integrating automated reasoning engines and adaptive retrieval strategies to further improve reliability in LLM-based applications.

REFERENCES:

[1] T. Brown et al., “Language Models are Few-Shot Learners,” NeurIPS, 2020.
[2] OpenAI, “GPT-4 Technical Report,” 2023.
[3] P. Lewis et al., “Retrieval-Augmented Generation,” NeurIPS, 2020.
[4] L. Ouyang et al., “Training Language Models with Human Feedback,”2022.
[5] Z. Ji et al., “Survey of Hallucination in NLP,” ACM Computing Surveys,2023.
[6] L. Gao et al., “Scaling Laws for Language Models,” 2023.
[7] Anthropic, “Constitutional AI,” 2023.
[8] K. Shuster et al., “Dialogue Models and Knowledge Grounding,” 2021.
[9] P. Liang et al., “Holistic Evaluation of Language Models,” 2022.
[10] R. Bommasani et al., “Foundation Models Opportunities and Risks,”2021.
[11] S. Narayan et al., “Grounded Generation Models,” 2021.
[12] E. Dinan et al., “Wizard of Wikipedia,” 2019.
[13] A. Glaese et al., “Improving Alignment of AI Systems,” 2022.
[14] J. Wei et al., “Chain-of-Thought Prompting,” 2022.
[15] R. Nakano et al., “WebGPT: Browser-Assisted QA,” 2021.
[16] Y. Zhang et al., “Fact Checking for LLM Outputs,” 2023.
[17] X. Wang et al., “Self-Consistency Improves Reasoning,” 2023.
[18] J. Xu et al., “LLM Reliability and Safety,” 2023.