Practical Strategies for Enhancing Reliability of GenAI Systems in Customer Operations: An Overview

Flurin Gishamer and **Alexander Khalil Arwadi**
Open Systems, Zurich
`{fgishamer,aarwadi}@open-systems.com`

Abstract

In operational environments, the reliability of large language models (LLMs) outputs is crucial due to their important role in decision-making. This paper addresses a common issue known as hallucinations, where LLMs generate incorrect or irrelevant outputs. We will detail our approach to mitigate these errors, focusing on applications in support ticket routing and information extraction tasks. Our approach improves the reliability of the generated suggestions by integrating error detection and remediation mechanisms directly into the GenAI systems workflow. We employ classical encoder-based transformers, such as RoBERTa, to constrain predictions to a fixed set of outputs, ensuring more accurate and relevant responses. We will present empirical results demonstrating significant reductions in error rates, underscoring the effectiveness of our methods in enhancing system reliability. We also discuss the practical challenges of deploying these strategies, including output verification and domain adaptation. Our presentation offers an overview of our strategies and encourages their adoption to improve the reliability of GenAI systems.

1 Overview

Open Systems' customer support operations focus on assisting customers with technical issues related to network connectivity and the company's managed security services. There is a strong motivation to automate as many of these processes as possible due to the following reasons:

- Timely ticket processing and resolution of technical issues enhance efficiency and thus directly impact our ability to meet SLAs.
- Since solving such problems requires a high degree of specialist knowledge, only trained experts can handle such tickets.
- Consequently, resolving tickets is associated with high costs and limits developers' capacity to work on new features.

Quality of support is a key differentiator for Open Systems. In light of this, it becomes clear that the most effective way to optimize efficiency is through a human-in-the-loop approach. This approach leverages the expertise of our support engineers, allowing them to verify the correctness of proposed solutions before any action is taken or feedback is sent back to the customer. In section 3, we will describe three practical use cases in more detail, using them as examples to illustrate the challenges and our solution approaches.

2 Related Work

The problem of hallucination in LLMs, where models generate incorrect or irrelevant outputs, has been extensively studied. Comprehensive surveys (Huang et al., 2023) and (Tonmoy et al., 2024) discuss principles, taxonomy, and various mitigation techniques, emphasizing the need for robust error detection mechanisms and integration of these strategies to enhance output reliability.

Retrieval-Augmented Generation (RAG) has emerged as a promising approach to improve the accuracy and relevance of LLM outputs by integrating external knowledge. The advent of BERT (Devlin et al., 2018) marked a significant breakthrough in contextual embeddings. Sentence-BERT (Reimers and Gurevych, 2019) uses Siamese BERT networks to generate semantically meaningful sentence embeddings, enabling the creation of effective vector indices for text. These indices, combined with algorithms such as approximate nearest neighbor (ANN) search techniques (Andoni et al., 2018), efficiently retrieve semantically similar documents. Lewis et al. (Lewis et al., 2020) proposed a RAG framework combining retrieval and generation with a bi-encoder and cross-encoder architecture, significantly enhancing precision and recall in knowledge-intensive NLP tasks.

Adapting LLMs to specific domains is essential for improving performance in specialized tasks. Sun et al. (Sun et al., 2023) highlighted the benefits of using in-context demonstrations to augment prompts in text classification. Evoking reasoning via abstraction (Zheng et al., 2023) is an approach to elicit reasoning in LLMs, prompting models to consider problems from a higher-level perspective to generate more accurate and contextually appropriate responses.

Fine-tuning LLMs is another effective approach for domain adaptation. This involves training a pre-trained model on a smaller, domain-specific dataset to refine performance. Direct Preference Optimization (DPO) simplifies alignment to human preferences (Rafailov et al., 2024). Parameter-Efficient Fine-Tuning (PEFT) techniques such as LoRa (Hu et al., 2021) reduce computational resources, making fine-tuning more accessible. Additionally, leveraging synthetic data, as demonstrated by the Self-Instruct framework (Wang et al., 2022), allows companies with limited training data to generate high-quality, domain-specific datasets, enhancing the feasibility of fine-tuning for organizations with scarce data resources.

3 System and Domain Description

As mentioned in section 1, we focus on integrating AI capabilities into customer operations to assist support engineers effectively. The primary interface for support engineers is the ticketing system. We employ an automation framework that communicates via a clearly specified API to facilitate seamless interaction between the ticketing system and the AI system.

In the following we describe three use cases that illustrate the challenges of our domain, and will serve as examples throughout the rest of this paper:

3.1 Maintenance Window Extraction

An important task is extracting and interpreting details from maintenance notification emails sent by internet service providers (ISPs). These emails contain critical information such as start and end times of planned maintenance, impact (duration of the actual maintenance works), and specific identifiers for affected hosts (where hosts refers to the actual hardware units Open System deploys at customers sites). The extraction process involves the following steps:

1. Identifying whether the email is an ISP maintenance notification.
2. Extracting the start and end times of the planned maintenance.
3. Extracting the impact of the maintenance works.
4. Extracting identifiers for the affected hosts.

Previously, this process required a support engineer to manually read through emails and input dates and times into a predefined template.

3.2 Finding Relevant Documents

Support engineers often require additional information from an internal knowledge base to resolve tickets. Experienced engineers manually attach relevant articles to tickets, but this process is time-consuming and often neglected due to its inefficiency. The manual process involves:

1. Reading the ticket description and any attached information.
2. Sifting through the knowledge base to identify relevant documents.
3. Attaching document links to the ticket for easy access.

For instance, a ticket mentioning a domain connectivity issue should prompt the engineer to attach a document detailing troubleshooting steps for such incidents, to provide valuable references to the support engineer tasked with resolving the ticket.

3.3 Ticket Queue Assignment

Before an engineer can resolve a ticket, an experienced engineer reads through it and assigns it to one of three possible queues. The three queues are defined as follows:

1. **Process**: Tickets with low technical complexity but involving many steps. These tasks are often tedious and time-consuming.
2. **Dispatch**: Tickets requiring technical knowledge related to networking and security but expected to take less than 15 minutes to resolve.
3. **Routine**: Very complex technical issues that are anticipated to take over 15 minutes to resolve.

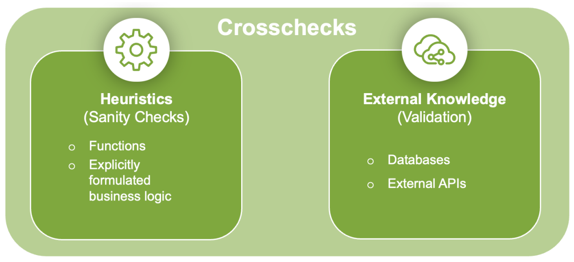

Figure 1: Approaches used in handling semantic errors in generative AI systems.

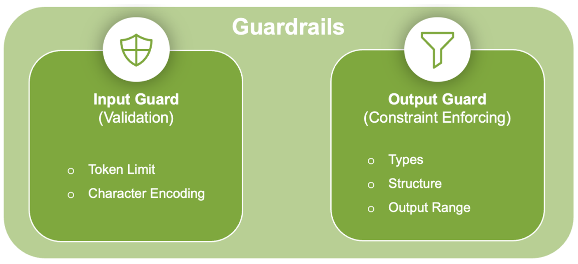

Figure 2: Approaches used in handling syntax errors in generative AI systems.

After the experienced engineer assigns a ticket to a queue, it automatically appears in the inbox of engineers assigned to a given queue.

Previously, this process required manual reading and categorization which was time-consuming and introduced a bottleneck.

4 Error Remediation

Error remediation in generative AI systems involves addressing both semantic and syntactic errors to improve the reliability of system outputs.

4.1 Semantic Issues

As shown in figure 1 we found that counteracting semantic errors, such as non-factual or nonsensical outputs, can be done most effectively using crosschecks in the form of heuristics and external knowledge validation. Heuristics integrate business logic by explicitly stating constraints in program code, helping enforce rules and correct errors. External knowledge validation involves using databases and APIs to cross-check and verify outputs. Addressing semantic errors aims to ensure accurate and, more importantly, contextually appropriate outputs.

4.1.1 Date extraction

Extracting the planned maintenance window's start and end dates and times is challenging due to the diverse formats in maintenance window emails from globally distributed ISPs (e.g., dd.mm.yyyy, mm.dd.yyyy, or "6 June 2020"). While a well-designed prompt enables the LLM to extract dates, discerning the exact date format consistently requires additional steps. We implemented a function that uses heuristics to determine the correct date format by comparing the extracted date string with the email's date (event date) and choosing the date format, yielding the closest date while ensuring this date lies in the future. In addition, this function ensures that the resulting time range remains within predefined boundaries.

4.1.2 Time Zone Extraction

Extracting time and timezone information from maintenance window emails presents another challenge. We developed a function that separately extracts time and timezone from the email and uses timezone libraries to convert the time to UTC accurately, so all resulting dates are in UTC such that the system can convert them to a consumer's timezone.

4.2 Syntax Issues

Syntax errors in generative AI systems involve challenges related to token limits, character encodings, and structural constraints, such as type mismatches or empty results. As can be seen in figure 2, we implement input and output guards that validate and enforce constraints to address these types of errors, ensuring outputs adhere to acceptable ranges and formats. This approach aligns with the Guardrails framework (Jarvis, 2023), where input guards ensure that only valid and intended text is passed to the LLM. In contrast, the output guard ensures that the generated outputs conform to the specified output format, handling correct type enforcement and empty response handling. We are aware that guardrails are often used to mitigate harmful or inappropriate outputs; however, in our case, we use them exclusively to handle syntactic issues.

4.2.1 Token Limit

The LLM's input text can be lengthy, often consisting of the prompt, email content, and format instructions. Composite prompts might exceed the LLM's token limit, causing errors. We apply text truncation to prevent this. If the text exceeds the token limit, the truncator only shortens the email's text, ensuring that preceding and following prompt instructions or in-context demonstrations are not affected.

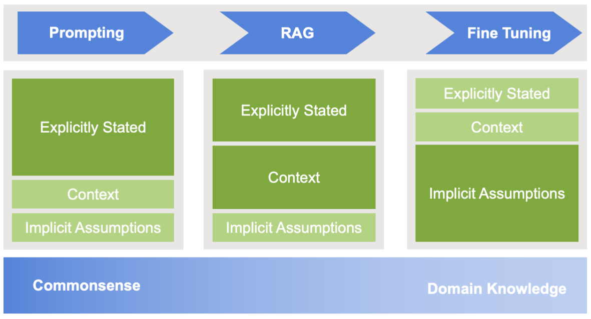

Figure 3: Framework for integrating external knowledge in generative AI systems.

4.2.2 Structural Validation

To enforce a specific structure in the LLM's output, we utilize Langchain's JSON output parser (Chase, 2022), which guides the LLM to produce a JSON string. Despite generally following format instructions, LLMs can still produce errors. The two primary issues are non-JSON formatted responses and correctly formatted JSON outputs with incorrect or missing keys/values. To address these, we implemented output guards. The first guard retries calling the LLM until it produces a correctly formatted JSON response, with a maximum retries parameter to prevent infinite loops. The second guard verifies that the generated response contains the expected keys/values, handling retries in the same manner as the first guard. This approach yields valid responses in a majority of cases, for the remaining cases, we return an empty response, which is handled by the calling automation framework.

5 Knowledge Integration

Integrating external knowledge is crucial for improving the accuracy and relevance of AI-generated outputs. We employ several methods to achieve this, including prompt enrichment with in-context demonstrations, RAG, and fine-tuning.

Figure 3 illustrates our framework for integrating external knowledge in generative AI systems. It distinguishes between use cases based on the level of domain knowledge required. For tasks that can be addressed with common sense, prompting often yields satisfactory results due to the model's pre-existing knowledge. However, for tasks requiring a high degree of domain-specific knowledge, fine-tuning becomes essential to make this implicit knowledge accessible to the model. RAG is particularly effective for intermediate scenarios where additional context is necessary but does not require extensive domain-specific knowledge.

5.1 Prompting

Prompting involves providing the AI model with specific instructions or examples to guide its output generation. This method leverages the model's pre-existing knowledge and directs it towards generating more accurate and relevant responses. One effective approach is prompt enrichment with in-context demonstrations, where a few examples relevant to the task are included in the prompt. Those examples help the model to understand the task better and generate more precise outputs, as pointed out in (Brown et al., 2020)

5.2 Retrieval-Augmented Generation (RAG)

Retrieval-augmented generation (RAG) enhances the accuracy and relevance of LLM outputs by integrating external knowledge through bi-encoder and cross-encoder components. In a one-pass system, bi-encoders create vector embeddings for queries and documents. These embeddings enable efficient retrieval based on semantic similarity, quickly identifying relevant documents from a vector index using optimized algorithms such as approximate nearest neighbor search. In a two-pass system, the initial retrieval by the bi-encoder is followed by a cross-encoder that refines the ranking of the retrieved documents. The cross-encoder computes similarity scores for each query-document pair, ensuring the most relevant documents are selected and ranked optimally. From this, it should become apparent that applying a re-ranker to a large result-set is prohibitive, as this pairwise comparison has to be performed for each document; it should also be noted that re-ranking will not improve a system's recall, as it only aims to optimize the final ranking of the documents retrieved during the first pass.

5.3 Fine-Tuning

Fine-tuning adjusts a pre-trained model on a specific dataset related to the target domain. This process refines the model's understanding and improves its performance on domain-specific data.

6 Empirical Results

Here, we show the results of our experiments for the use cases "Ticket Queue Assignment" and "Relevant Document Retrieval" outlined in section 3.

Approach	Precision	Recall	F1
Prompting	0.58	0.50	0.52
CARP	0.67	0.45	0.52
roBERTa	0.83	0.84	0.83

Table 1: Performance metrics of different approaches for ticket queue classification. Prompting achieves moderate precision, while CARP improves precision at the cost of recall, maintaining the same F1 score. roBERTa demonstrates the best overall performance with high precision, recall, and F1 score.

Document Format	NDCG	Recall@k=10
Original	0.17	0.28
Chunked	0.23	0.32
Re Ranker	0.22	0.32

Table 2: Performance metrics of different document retrieval approaches used. "Original" refers to the unmodified documents, "Chunked" refers to documents split into smaller parts for more precise retrieval, and "Re Ranker" refers to documents re-ranked by a fine-tuned model.

6.1 Ticket Queue Assignment

In table 1, we observe that prompting for queue classification achieves an F1 score of only 0.52. CARP, which uses in-context demonstrations to enrich the prompt with examples of different classes (Sun et al., 2023), increases the precision from 0.58 to 0.67. However, this approach results in a drop in recall, maintaining the F1 score at 0.52. This indicates that while precision improves, the model becomes less capable of correctly identifying all relevant instances. Finally, finetuning an encoder-based model using labeled data (obtained through process feedback) delivers the best results, with an F1 score of 0.83. We hypothesize that the improved performance can be attributed to the model's ability to leverage labeled data to learn more nuanced distinctions between classes, thus improving both precision and recall.

6.2 Relevant Document Retrieval

A qualitative assessment with support engineers confirmed the viability of the RAG system for relevant document retrieval, especially in comparison to the manual approach.

As can be seen in table 2, the improvements shown in the "Chunked" row demonstrate the effectiveness of splitting documents into smaller, paragraph-level units. By doing so, we increased the recall@k=10 from 0.28 to 0.32 and the NDCG from 0.17 to 0.23. The expected outcome for re-ranking is that it would not enhance recall@k, but improve the NDCG. However, it also failed to improve the ranking, as can be seen in the decrease of the NDCG from 0.23 to 0.22. We attribute this to the fact that the mapping of the tickets to the relevant documents was not exhaustive, leading to a high number of false negatives in the dataset.

7 Conclusion

Our experiments demonstrated that the outlined error remediation and knowledge integration strategies can significantly enhance the reliability of generative AI systems in customer operations. By employing techniques such as RAG and fine-tuning and addressing both semantic and syntactic errors, we successfully productionized a generative AI system that significantly improves the productivity of our support engineers. Future work will focus on integrating process feedback to enhance label quality and employing synthetic data where obtaining high-quality labels is not feasible. These efforts aim to enable a higher degree of domain adaptation, thereby facilitating the automation of more complex processes, such as multi-step workflows.

7.1 RAG

Integrating a cross-encoder into a RAG system can improve NDCG scores i.e. the quality of the obtained ranking. However, this approach requires a high-quality dataset with accurate labels. In the presence of noisy data, the advantages of such a two-pass system often diminishes, as the noise can lead to unreliable evaluation and performance outcomes. Under such conditions, a simpler bi-encoder-based system may be more effective, as it is less sensitive to data quality issues and can still provide robust retrieval performance without the added complexity.

7.2 Fine Tuning for Domain Adaptation

Fine-tuning, as an alternative to prompting or RAG, is particularly useful for classification problems in natural language. This approach often lends itself well to smaller transformer-based models, which offer several advantages:

- **Resource Efficiency**: Smaller models require fewer computational resources, especially during inference, but also for training.

- **Confidence Scores**: For encoder-based models, fine-tuning provides not only the predicted class but also a confidence score. This feature allows for the adjustment of the decision thresholds, enabling the optimization of the precision-recall ratio for a given use cases.
- **Fixed Set of Outputs**: As the number of categories is fixed, the model will always predict one of the predefined classes. We can thereby circumvent the issue of out-of-domain hallucinations altogether.

References

Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. 2018. Approximate nearest neighbor search in high dimensions. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3287–3318. World Scientific.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Harrison Chase. 2022. Github repository: Langchain. *Langchain*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232*.

Colin Jarvis. 2023. How to implement llm guardrails. *OpenAI Cookbook*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Xiaofei Sun, Xiaoya Li, Jiwei Li, Fei Wu, Shangwei Guo, Tianwei Zhang, and Guoyin Wang. 2023. Text classification via large language models. *arXiv preprint arXiv:2305.08377*.

SM Tonmoy, SM Zaman, Vinija Jain, Anku Rani, Vipula Rawte, Aman Chadha, and Amitava Das. 2024. A comprehensive survey of hallucination mitigation techniques in large language models. *arXiv preprint arXiv:2401.01313*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*.

Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H Chi, Quoc V Le, and Denny Zhou. 2023. Take a step back: Evoking reasoning via abstraction in large language models. *arXiv preprint arXiv:2310.06117*.