IncidentNavigator: Intelligent Support Incident Resolution

Matéo Munoz[a,1], **Axel Colmant**[a,1] **and Mohamed Massamba SENE**[a,1]

[a]Université Claude Bernard Lyon 1

Abstract.
Incident resolution in high-stakes environments demands speed, precision, and adaptability, yet traditional methods often lead to inefficiencies and operator fatigue. This paper introduces IncidentNavigator, a system designed to streamline support incident resolution through dynamic query refinement, document retrieval, and guided response generation. By enhancing interactivity and transparency, the system offers context-aware solutions with clear source attribution, reducing errors and improving decision-making. Evaluation demonstrates high response relevancy (90%) and faithfulness (74%), underscoring its potential to transform incident management workflows. IncidentNavigator sets a new standard for intelligent and reliable support systems in critical environments.

Keywords: Incident resolution, Retrieval-Augmented Generation (RAG), Large Language Models (LLMs), intelligent incident management.

1 Introduction

Incident management systems are fundamental to maintaining operational continuity, yet operators frequently struggle with efficiently retrieving and applying solutions from historical databases due to system limitations and data fragmentation [8, 2]. Traditional static retrieval systems such as Elasticsearch[2] and Apache Solr[3], while efficient at keyword matching, lack the contextual understanding needed for technical incidents with varied terminologies. Even modern approaches like Dense Passage Retrieval (DPR) [5], which maps queries and passages into a shared dense vector space for more accurate semantic matching, and Query2Doc [12], which leverages query expansion to improve document retrieval relevance, have significantly enhanced semantic search capabilities. However, these methods primarily focus on static matching between queries and documents, offering limited opportunities for iterative query refinement or interactive engagement.

Large Language Models (LLMs) represent a major breakthrough in natural language processing, excelling at understanding and generating human-like text. Their pretraining on vast amounts of diverse data enables them to capture intricate linguistic patterns and perform complex reasoning. This makes them highly effective for tasks like question answering, summarization, and dialogue generation. However, LLMs face challenges such as hallucination, where they generate plausible-sounding but factually incorrect information, and their responses are limited to the knowledge encoded during training, which may become outdated.

[1] Equal contribution.
[2] https://www.elastic.co/elasticsearch
[3] https://solr.apache.org/

To address these limitations, Retrieval-Augmented Generation (RAG) combines the generative capabilities of LLMs with external knowledge retrieval. RAG enhances the reliability of responses by retrieving relevant documents from external knowledge bases or databases and grounding the generated output in this retrieved information. This hybrid approach mitigates hallucination risks, improves factual accuracy, and enables real-time access to up-to-date or domain-specific knowledge.

Examples of RAG systems include Fusion-in-Decoder (FID) [3], which integrates multiple retrieved documents into a single coherent response, and ColBERT [6], which uses contextual embeddings for efficient and accurate retrieval. While these systems demonstrate significant improvements in contextual understanding, they often rely on static retrieval mechanisms, limiting their ability to iteratively refine queries or support interactive engagement.

The emergence of dynamic retrieval systems, particularly RAG-turn [10] and HyDE [1], has shown how conversational AI can iteratively refine queries, significantly improving accuracy for vague or incomplete inputs [4, 11]. Recent work on plug-and-play retrieval feedback [13] has shown promising directions for maintaining system reliability.

A crucial limitation in current LLM-based solutions is the lack of transparency. While explainability frameworks like LIME [9] and SHAP [7] exist, they are rarely integrated into RAG systems, leaving a critical gap in solution traceability. To address this gap we present IncidentNavigator, a comprehensive solution that addresses these limitations through two key innovations:

- Dynamic LLM interaction for step-by-step query refinement, enhancing accuracy through conversational guidance;
- Enhanced transparency with direct source attribution to verified tickets and metadata, reducing hallucination risks.

This paper details the architecture, implementation, and evaluation of IncidentNavigator, demonstrating how the integration of advanced RAG techniques with interactive refinement can significantly improve incident resolution efficiency. Our work builds upon fundamental improvements in retrieval mechanisms while contributing to the growing field of practical LLM applications in critical operational environments, with particular emphasis on transparency, adaptability, and user interaction. The code and the demo video are available on Github[4] and Youtube[5] respectively.

[4] https://github.com/MrSneaker/IncidentNavigatorProject
[5] https://www.youtube.com/watch?v=WaN5e42Wt4U

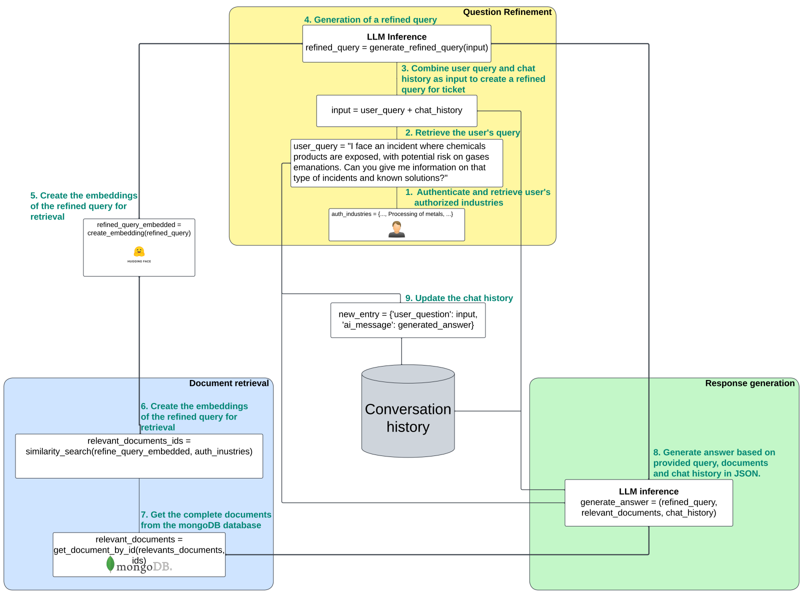
Question Refinement
4. Generation of a refined query
LLM Inference
refined_query = generate_refined_query(input)
3. Combine user query and chat history as input to create a refined query for ticket
input = user_query + chat_history
2. Retrieve the user's query
user_query = "I face an incident where chemicals products are exposed, with potential risk on gases emanations. Can you give me information on that type of incidents and known solutions?"
1. Authenticate and retrieve user's authorized industries
auth_industries = {..., Processing of metals, ...}
5. Create the embeddings of the refined query for retrieval
refined_query_embedded = create_embedding(refined_query)
HUGGING FACE
9. Update the chat history
new_entry = {'user_question': input, 'ai_message': generated_answer}
Conversation history
Document retrieval
6. Create the embeddings of the refined query for retrieval
relevant_documents_ids = similarity_search(refine_query_embedded, auth_inustries)
7. Get the complete documents from the mongoDB database
relevant_documents = get_document_by_id(relevants_documents, ids)
mongoDB.
Response generation
8. Generate answer based on provided query, documents and chat history in JSON.
LLM inference
generate_answer = (refined_query, relevant_documents, chat_history)

Figure 1. Representation of the generation process of the IncidentNavigator pipeline

2 The IncidentNavigator Pipeline

The IncidentNavigator pipeline, depicted in Figure 1, integrates various components to provide a seamless flow from user input to intelligent incident resolution. The pipeline operates in three interconnected phases: **Question Refinement**, **Document Retrieval**, and **Response Generation**. Below, we provide the theoretical framework for each phase:

2.1 *Question Refinement*

The process begins with *user query refinement*, which optimizes the input for downstream retrieval and response generation. Let n represent the current step in the conversation, and H the chat history, which is initialized as empty:

$$H = \emptyset$$

Let the initial user input be denoted as Q_{user}, which includes their question and contextual information. The refined query, Q_{refined}, is generated using the function f_{LLM}:

$$\begin{aligned} f_{\text{LLM}} : (Q_{\text{user}} \times H) &\to Q_{\text{refined}} \\ (Q_{\text{user}}, H) &\mapsto Q_{\text{refined}} \end{aligned}$$

Here:

- f_{LLM} is a non-deterministic function implemented by a Large Language Model. The non-determinism is achieved by adjusting the temperature parameter of the LLM, allowing for diverse query reformulations.
- $H = \{(Q_1, A_1), (Q_2, A_2), \ldots, (Q_{n-1}, A_{n-1})\}$ represents the sequence of prior user queries and system responses up to step $n - 1$.

The refined query Q_{refined} is embedded into a vector representation E_{refined} using a pre-trained embedding function ψ:

$$\begin{aligned} \psi : Q_{\text{refined}} &\to E_{\text{refined}} \\ Q_{\text{refined}} &\mapsto E_{\text{refined}} \end{aligned}$$

2.2 *Document Retrieval*

In this phase, relevant documents are retrieved from a vector store and a MongoDB database based on the refined query embeddings. Let the vector store be represented as a set of document embeddings $\{E_{\text{doc}_i}\}_{i=1}^{N}$, where N is the total number of documents. Retrieval is performed using similarity search:

$$\begin{aligned} \text{sim} : (E_{\text{refined}} \times E_{\text{doc}_i}) &\to \mathbb{R} \\ (E_{\text{refined}}, E_{\text{doc}_i}) &\mapsto \frac{E_{\text{refined}} \cdot E_{\text{doc}_i}}{\|E_{\text{refined}}\| \|E_{\text{doc}_i}\|} \end{aligned}$$

The top k documents are selected based on the similarity metric:

$$D = \text{argmax}^{k}_{i \in \{1,\ldots,N\}} \text{sim}(E_{\text{refined}}, E_{\text{doc}_i})$$

Here, $D = \{d_1, d_2, \ldots, d_k\}$ represents the set of retrieved documents, where each d_i contains metadata and historical ticket details fetched from the MongoDB ticket database. These documents are then passed to the response generation phase.

2.3 *Response Generation*

In the final phase, the retrieved documents D, along with the refined query Q_{refined} and chat history H, are passed to the LLM for response generation. Let g_{LLM} represent the generation function of the LLM:

$$\begin{aligned} g_{\text{LLM}} : (Q_{\text{refined}} \times D \times H) &\to A_{\text{generated}} \\ (Q_{\text{refined}}, D, H) &\mapsto A_{\text{generated}} \end{aligned}$$

The generated answer $A_{\text{generated}}$ is a JSON object containing:

- **Response:** The main answer to the user query in HTML format.
- **Highlighting:** The text is color-coded based on the specific retrieved document d_i that supports each part of the response. This ensures transparency by explicitly linking generated text to its sources.

This structure ensures traceability by providing direct references to the documents supporting the response, mitigating hallucination risks.

2.4 *Updating Chat History*

Finally, the chat history is updated with the user query Q_{user} and the generated answer $A_{\text{generated}}$ for subsequent interactions:

$$H_{\text{new}} = H \cup \{(Q_{\text{user}}, A_{\text{generated}})\}$$

This structured approach ensures that the system is not only efficient but also interpretable and robust, leveraging RAG techniques to handle complex incident queries in real time.

3 Implementation

To implement the theoretical framework described above, we used the following components:

- **Large Language Model (LLM):** Both f_{LLM} and g_{LLM} leverage the same LLM, specifically Llama3.3-70b from GroqCloud[6].
  - f_{LLM} refines the user's query by incorporating prior context and aligning it optimally for retrieval.
  - g_{LLM} synthesizes a response by combining the refined query, retrieved documents, and chat history to produce an accurate, contextually relevant answer.

  This separation allows the LLM to handle both query refinement and response generation with task-specific prompts.
- **Embedding Model:** We used E5 Large V2[7] to convert refined queries into dense vector representations, enabling precise similarity searches during retrieval.
- **Vector Store and Database:** Document retrieval was powered by Weaviate[8] for vector storage and similarity search, while MongoDB[9] stored and managed detailed ticket data, ensuring scalability and efficient information access.

This setup ensures seamless functionality across the pipeline.

4 Evaluation

4.1 *Retrieval Evaluation*

To evaluate the IncidentNavigator system, we created a dataset of 24 user questions[10]. These questions, written to simulate real-world queries, covered diverse incident management scenarios. Using our pipeline, we retrieved answers and their referenced documents to build the dataset and assess retrieval and response generation performance.

We compared the base pipeline with alternatives using smaller LLMs (Llama3-8B) and different embedding models (gtr-t5-large). Evaluation was conducted using the RAGAS framework[11], which employs Llama3-70b to score metrics like response relevancy, context precision, and faithfulness.

The results obtained are presented in Table 1. Our evaluation revealed that while smaller LLMs showed higher response relevancy (97.4% vs 90%), the base pipeline's superior context precision (82.9% vs 69.7% and 49.9%) and faithfulness (73.7% vs 65.1% and 65.4%) lead to responses that are better grounded in the retrieved context, reducing hallucination and improving answer reliability.

[6] https://console.groq.com/playground
[7] https://huggingface.co/intfloat/e5-large-v2
[8] https://weaviate.io/
[9] https://www.mongodb.com/
[10] https://github.com/MrSneaker/IncidentNavigatorProject/blob/main/data/test.csv
[11] https://docs.ragas.io/en/stable/

Metric	Base Pipeline	Pipeline with Llama3-8B	Pipeline gtr-t5-large
Mean Context Precision	82.9	69.7	49.9
Mean Context Recall	48.9	44.3	13.5
Mean Response Relevancy	90	97.4	82.7
Mean Faithfulness	73.7	65.1	65.4

Table 1. Performance metrics for the IncidentNavigator system.

4.2 *User Evaluation*

In this next subsection, we will compare our model's answers (Option A) to both a human-generated baseline answer (Option B) and a response generated by Llama3.3-70b without our pipeline (Option C). This was done using a user survey to assess users' preferred response based on a set of 10 questions with the answer of each model being presented as an option for selection. A total of 18 responses were obtained, and the results are presented in the Figure 2.

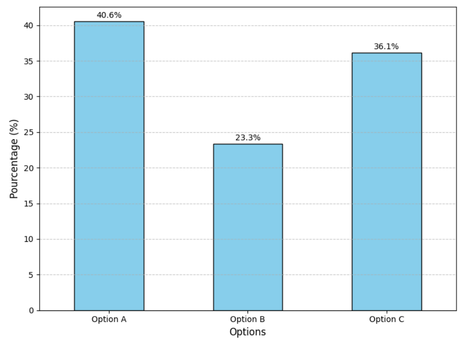

Figure 2. Frequency distribution of Options chosen

Based on the results, our approach received the highest frequency with approximately 40.6%, followed by Llama3.3-70b's unprocessed output with 36.1%, while the human-generated baseline received the lowest with 23.3%. Interestingly, this suggests that both AI-generated responses (Options A and C) were preferred over the human baseline, with our approach being slightly preferred over Llama3.3-70b's unprocessed output.

5 Conclusion

IncidentNavigator demonstrates the transformative potential of integrating Retrieval-Augmented Generation (RAG) and Large Language Models (LLMs) for intelligent incident management. By combining dynamic query refinement, robust document retrieval, and transparent response generation, the system addresses key challenges in traditional incident resolution workflows. The evaluation results underscore its efficacy, particularly in generating relevant and faithful responses while maintaining source traceability.

This work contributes to the growing field of AI-assisted support systems by providing a scalable and explainable framework adaptable to various high-stakes environments. Future research could explore the integration of additional explainability techniques and the expansion of the pipeline to support multilingual and multimodal contexts. Overall, IncidentNavigator paves the way for more efficient, reliable, and user-centric solutions in critical operational domains.

Acknowledgements

We would like to express our deepest gratitude to Bruno Yun, our professor at Université Claude Bernard Lyon 1 in charge of the course Theory and Practical Applications of LLMs. His guidance and support throughout the project, from its inception to its current state, have been invaluable.

References

[1] Better RAG with HyDE - Hypothetical Document Embeddings - Zilliz Learn — zilliz.com. https://zilliz.com/learn/improve-rag-and-information-retrieval-with-hyde-hypothetical-embeddings. [Accessed 17-01-2025].

[2] N. Ashish, D. Kalashnikov, S. Mehrotra, and N. Venkatasubramanian. An event based approach to situational representation. 2009. doi: 10.48550/ARXIV.0906.4096. URL https://arxiv.org/abs/0906.4096.

[3] G. Izacard and E. Grave. Leveraging passage retrieval with generative models for open domain question answering. 2020. doi: 10.48550/ARXIV.2007.01282. URL https://arxiv.org/abs/2007.01282.

[4] Z. Jiang, F. F. Xu, L. Gao, Z. Sun, Q. Liu, J. Dwivedi-Yu, Y. Yang, J. Callan, and G. Neubig. Active retrieval augmented generation. 2023. doi: 10.48550/ARXIV.2305.06983. URL https://arxiv.org/abs/2305.06983.

[5] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. Dense passage retrieval for open-domain question answering. 2020. doi: 10.48550/ARXIV.2004.04906. URL https://arxiv.org/abs/2004.04906.

[6] O. Khattab and M. Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. 2020. doi: 10.48550/ARXIV.2004.12832. URL https://arxiv.org/abs/2004.12832.

[7] S. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. 2017. doi: 10.48550/ARXIV.1705.07874. URL https://arxiv.org/abs/1705.07874.

[8] Y. Remil, A. Bendimerad, R. Mathonat, and M. Kaytoue. Aiops solutions for incident management: Technical guidelines and a comprehensive literature review. 2024. doi: 10.48550/ARXIV.2404.01363. URL https://arxiv.org/abs/2404.01363.

[9] M. T. Ribeiro, S. Singh, and C. Guestrin. "why should i trust you?": Explaining the predictions of any classifier. 2016. doi: 10.48550/ARXIV.1602.04938. URL https://arxiv.org/abs/1602.04938.

[10] K. Shuster, S. Poff, M. Chen, D. Kiela, and J. Weston. Retrieval augmentation reduces hallucination in conversation. 2021. doi: 10.48550/ARXIV.2104.07567. URL https://arxiv.org/abs/2104.07567.

[11] J. Sun, C. Xu, L. Tang, S. Wang, C. Lin, Y. Gong, L. M. Ni, H.-Y. Shum, and J. Guo. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. 2023. doi: 10.48550/ARXIV.2307.07697. URL https://arxiv.org/abs/2307.07697.

[12] L. Wang, N. Yang, and F. Wei. Query2doc: Query expansion with large language models. 2023. doi: 10.48550/ARXIV.2303.07678. URL https://arxiv.org/abs/2303.07678.

[13] W. Yu, Z. Zhang, Z. Liang, M. Jiang, and A. Sabharwal. Improving language models via plug-and-play retrieval feedback. 2023. doi: 10.48550/ARXIV.2305.14002. URL https://arxiv.org/abs/2305.14002.