Evaluating the Performance of RAG Methods for Conversational AI in the Airport Domain

Yuyang Li[1], Philip J.M. Kerbusch[2], Raimon H.R. Pruim[2], Tobias Käfer[1]

[1]Karlsruhe Institute of Technology, [2]Royal Schiphol Group
[2]Royal Schiphol Group, [1]Karlsruhe Institute of Technology
yuyang.li@kit.edu,
tobias.kaefer@kit.edu

arXiv:2505.13006v1 [cs.CL] 19 May 2025

Abstract

Airports from the top 20 in terms of annual passengers are highly dynamic environments with thousands of flights daily, and they aim to increase the degree of automation. To contribute to this, we implemented a Conversational AI system that enables staff in an airport to communicate with flight information systems. This system not only answers standard airport queries but also resolves airport terminology, jargon, abbreviations, and dynamic questions involving reasoning. In this paper, we built three different Retrieval-Augmented Generation (RAG) methods, including traditional RAG, SQL RAG, and Knowledge Graph-based RAG (Graph RAG). Experiments showed that traditional RAG achieved 84.84% accuracy using BM25 + GPT-4 but occasionally produced hallucinations, which is risky to airport safety. In contrast, SQL RAG and Graph RAG achieved 80.85% and 91.49% accuracy respectively, with significantly fewer hallucinations. Moreover, Graph RAG was especially effective for questions that involved reasoning. Based on our observations, we thus recommend SQL RAG and Graph RAG are better for airport environments, due to fewer hallucinations and the ability to handle dynamic questions.

1 Introduction

Amsterdam Airport Schiphol, one of the top 20 airports in the world, ranked by annual passenger numbers, handles thousands of flights each day. These airports rely on staff like gate planners and apron controllers to access and update data across systems. For these employees, traditional database queries can be complex and time-consuming for some employees who are not query experts when they need flight information. A conversational AI system with a natural language query (NLQ) interface allows all employees to interact with systems naturally, asking questions like, "Which fights are at ramp D07?" and receiving instant answers. This improves productivity, and streamlines workflows, especially in high-pressure areas like at the gate, where less educated workers require access to up-to-date information. By replacing strict query formats with intuitive, real-time responses, conversational AI enhances decision-making and efficiency, making it a suitable solution for dynamic environments such as airports.

Building such a system is challenging because flight data is stored by experts in tables using aviation abbreviations. We need our system to understand these datasets to answer questions from the airport domain. Additionally, ensuring aviation safety is a major concern; the system must be safe and enable employees to perform accurate operations. We address those challenges using two research questions.

The first question is how to handle flight data so that our system can answer different questions. We divided the questions into three types:

- **Straightforward questions:** Questions that can be directly answered from the flight data.
- **Questions involving specialized airport jargon, abbreviations, and incomplete queries:** Operators often use shorthand or omit context. Flight "KL0123" might be referred to as "0123" or "123," while gate "C05" might be shortened to "C5." Abbreviations like "KLM" for "KLM Royal Dutch Airlines" or "Delta" for "Delta Air Lines" are also common. Operators frequently ask short, incomplete questions, e. g., "Which flights are at D04?" or "What is the gate for that Delta airline?" Without resolving missing details such, these questions cannot be answered.
- **Dynamic questions:** Questions that involve additional calculations and reasoning, especially related to time. Examples include "What is the connecting flight's onramp time

for DL1000?" or "What is DL1000's next flight from the same ramp?" These queries require reasoning through connections between flights and retrieving specific details.

The second research question is about how to reduce hallucinations (Xu et al., 2024) for the safety of aviation operations. Hallucinations occur when LLMs generate information not based on facts or their training data. In high-safety environments such as airports, however the output should be factual and not imaginative (Jacobs and Jaschke, 2024). For example, if the system gives wrong gate numbers, flight schedules, or safety instructions, this might disrupt aviation operations, cause delays, or even risk passenger safety. Thus, accurate responses are important.

In this case study, we examine three Retrieval-Augmented Generation (RAG) techniques for the airport environment: Traditional RAG (Lewis et al., 2021) Retrieves relevant information from the flight database and uses LLMs to generate answers based on the retrieved data and original questions. SQL RAG (Guo et al., 2023) stores all datasets in an SQL database and converts natural language questions (NLQ) into structured SQL queries. Knowledge Graph-based Retrieval-Augmented Generation (Graph RAG) (Edge et al., 2024) aims to improve the performance of LLM tasks by applying RAG techniques to Knowledge Graphs (KGs), requiring the original datasets to be stored in the knowledge graph. A key challenge is retrieving the correct flight information from thousands of flights while minimizing hallucinations.

The paper is structured as follows: We first survey related work (Sec. 2), then present our dataset (Sec. 3), followed by a high-level description of our experiments (Sec. 4). We then present the results for the research questions (Sec. 5), and lastly conclude (Sec. 6). In the Appendix A, we provide further details, especially on the question generation and classification, next to our prompting.

2 Related Work

2.1 Traditional RAG

Traditional Retrieval-Augmented Generation (RAG) consists of two main stages: the Retriever and the Generator (Louis et al., 2023). The Retriever identifies relevant documents based on user input, and the Generator uses these documents to produce responses. We explore three retrieval methods: keyword search, semantic search, and hybrid search, using large language models (LLMs) for answer generation.

In keyword search, TF-IDF and BM25 are employed to evaluate retrieval performance. TF-IDF computes term frequency (TF) and inverse document frequency (IDF) (Liu et al., 2018; Robertson, 2004), measuring how important a term is within a document and across the corpus. BM25 extends TF-IDF with a term saturation function (Robertson and Zaragoza, 2009), reducing the influence of extremely frequent terms that often carry less informative value (Chen and Wiseman, 2023).

Semantic search methods include similarity search, vector databases like FAISS (Jegou et al., 2017; George and Rajan, 2022), k-Nearest Neighbors (KNN), Locality-Sensitive Hashing (LSH) (Jafari et al., 2021), and Maximal Marginal Relevance (MMR) (Mao et al., 2020). Unlike keyword search, semantic search aims to understand user intent and word meanings (Gao et al., 2024). Embedding models such as Word2Vec convert words into vectors (Mikolov et al., 2013), where cosine similarity measures similarities between queries and documents.

Hybrid search combines keyword and semantic methods, re-ranking results using the Reciprocal Rank Fusion (RRF) algorithm (Robert Lee, 2024). By combining two search methods, the hybrid search can not only find flight information by keywords but also find information by the deeper meaning of the queries (Sarmah et al., 2024).

2.2 SQL RAG

Text-to-SQL aims to transfer natural language automatically questions(NLQs) into SQL queries. LLMs recently emerged as an option for Text-to-SQL task (Rajkumar et al., 2022). The trick to handling text-to-SQL tasks with LLMs is to apply prompt engineering. Five prompt styles for Text-to-SQL are explored in the previous research (Gao et al., 2023). Basic Prompt (BSP) is a simple representation with no instructions; Text Representation Prompt (TRP) adds basic task guidance; OpenAI Demonstration Prompt (ODP) adds explicit rules like "Complete sqlite SQL query only,"; Code Representation Prompt (CRP) uses SQL-style schema descriptions with detailed database information like primary/foreign keys, and Alpaca SFT Prompt (ASP) adopts Markdown for structured training prompts. In (Gao et al., 2023), CRP achieves the best performance in most LLMs, by providing complete database information and utilizing the LLMs' strength in understanding code.

2.3 Graph RAG

A Knowledge Graph (KG) is a structured representation of entities (nodes), their attributes, and relationships (edges), typically stored in graph databases or triple stores (Sarmah et al., 2024). Its basic unit is a triple: subject, predicate, object. In Graph RAG (Retrieval-Augmented Generation), natural language questions are converted into query languages like SPARQL for RDF graphs or Cypher for Neo4j property graphs. Research indicates that Neo4j's labeled property graph model offers faster and more efficient real-time analysis and dynamic querying compared to complex RDF ontologies in enterprise projects (Barrasa et al., 2023). Neo4j's property graph model better meets industrial needs. Flight information can be automatically integrated into the knowledge graph by matching the row and column names of the flight table, with relationships manually defined based on flight numbers.

3 Dataset and Questions

Our flight information dataset is tabular containing thousands of flights with key details such as flight number, aircraft category, bus gate, bus service needed, flight UID, ramp, expected on-ramp time, connecting flight number, etc.

To evaluate the effectiveness of different retrieval methods, we classified the questions, and then based on these questions, we created two ground truth datasets: a straightforward dataset and a complicated, ambiguous dataset.

The straightforward dataset consists of unambiguous questions that can be directly answered from flight information. Examples include: "What category of aircraft is designated for flight KL1000?" and "Which ramp is assigned for flight KL1000?". Such questions are easily handled by retrieval methods to select the most relevant information. This dataset contains thousands of question-answer pairs, with around 100 to 200 pairs selected for the RAG methods comparison.

The complicated and ambiguous dataset contains questions with variables that may be unclear or missing from the flight information which cannot be directly queried from the tabular dataset. Examples are: "Which flight is at gate B24?" or "Which gate is assigned to the 0164 flight?", "When is Delta landing?" Here, 'B24' might relate to multiple flights or meanings (bus gate or ramp number), and '0164' is not a complete flight number, 'Delta' also needs clarification. This dataset also contains thousands of question-answer pairs, with 185 pairs randomly selected for comparison. More information on question generation and question classification is provided in the Appendix A.

4 Experiments

To handle the flight tabular dataset, our conversational AI should understand the meaning of these flight terms, it also needs to understand specific jargon and terminology. We explore three RAG methods for a conversational system on flight data.

Figure 1 shows the traditional RAG method. When a user asks a question, various retrieval methods are employed to retrieve the correct flight data from the flight information dataset. These methods are mainly divided into three categories: keyword search, semantic search, and hybrid search. After retrieving the relevant flight information, Large Language Models (LLMs) generate answers to the user's questions based on this data. Several LLMs were tested to assess their performance, including GPT models, Llama-3-8B-Instruct, BERT, and BERT-related models.

Figure 2 shows the SQL RAG method, which begins with users asking natural language questions. An LLM processes these questions using the SQL database schema to generate appropriate SQL queries. The queries retrieve relevant information from the SQL database, which the LLM then interprets and reformulates into human-readable answers. Following the approach in (Gao et al., 2023), we experimented with Code Representation Prompt (CRP) and OpenAI Demonstration Prompt (ODP) to fine-tune the prompts and improve the SQL RAG results. More details of SQL RAG prompts are provided in Appendix A.

Figure 3 shows the Graph RAG method, which also starts with users asking natural language questions. An LLM processes these questions using the graph schema from the graph database to generate graph queries. We use Neo4j's APOC plugin to extract the schema by executing 'CALL apoc.meta.schema() YIELD value RETURN value' and include it in the prompt. and the LLM interprets this data to formulate human-readable answers. The graph structure enables context-aware retrieval and reasoning, more details of Graph RAG prompts are provided in Appendix A.

The three RAG methods described above can handle straightforward datasets easily because the answers all exist in the flight tabular, we will add

some explanations about flight row names' meanings to the prompts, so that LLMs can generate better more accurate answers. However, questions about jargon and short sentences from complicated or ambiguous datasets need to be classified using a question classification prompt, as shown in Figure 4. After classification, each question is directed to different prompts to answer jargon and abbreviations.

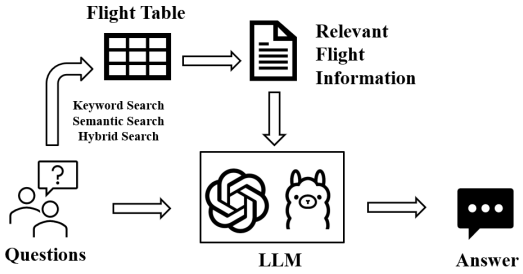

Figure 1: Traditional RAG Method

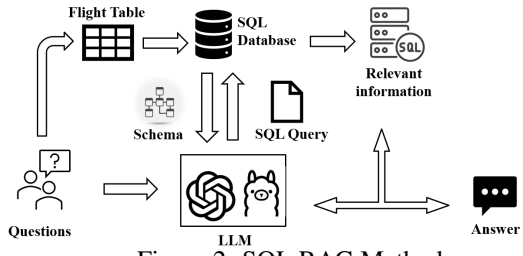

Figure 2: SQL RAG Method

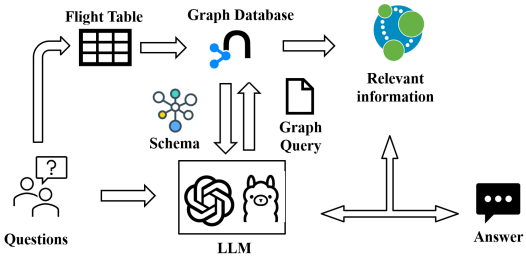

Figure 3: Graph RAG Method

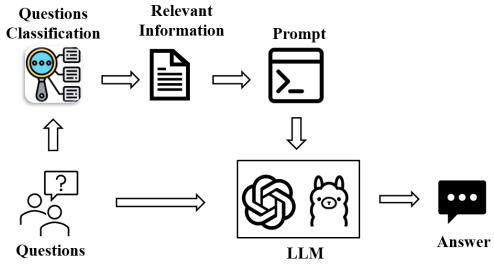

Figure 4: Method on Ambiguous question dataset

5 Results

In this section, we present the experimental results, structured using our research questions.

5.1 RQ1: How to handle flight data for different questions?

5.1.1 Straightforward questions

Table 1 summarizes the performance of various retrieval methods within the traditional RAG pipeline in the straightforward dataset. BM25 outperforms other methods, achieving approximately 86.54% accuracy in retrieving the correct articles. The hybrid search, which combines BM25 and the vector database FAISS in a 9:1 proportion, performs second best, with an accuracy of 85.78% for identifying the correct article as the highest-ranked and 98.00% accuracy for including the correct article within the top 10 results. This indicates the successful retrieval of correct articles among the top 10 most relevant ones. However, changing the proportion to 1:9 yields only 0.59% accuracy within the top 30 articles, suggesting that the correct articles rarely appear among the top 30 results. Following BM25 and the hybrid search, TF-IDF with cosine similarity and Euclidean distance achieve accuracies of 67.70% and 67.55%, respectively. The vector database FAISS alone performs the worst, with an accuracy of 0%.

Table 2 shows how various LLMs perform in generating answers for the simple dataset. Because the dataset is large, we randomly selected 100 questions for the experiment. The LLMs' answers were manually compared to the standard answers; correct ones were marked "True," and incorrect ones were "False." Accuracy was calculated by dividing the number of correct answers by the total number of questions. In these two tables, we chose BM25+GPT4 as the traditional RAG pipeline and achieved a total accuracy of 84.40% in the end. The reason keyword search outperforms semantic search is probably because, in the airport environment, most questions are about specific flights, times, or ramps. These questions don't require a deep semantic understanding of the content.

Table 3 shows the performance of SQL RAG. The results indicate that CRP significantly outperforms ODP in most of the cases. EM(Exact Match) measures the strict match between the predicted SQL query and the ground truth regarding syntax and structure. while EX(Execution Match) evaluates whether the execution outputs of the predicted SQL match the ground truth on the database. Few-shot learning was applied using 47 manually created examples, including questions, SQL queries, and corresponding answers. With CRP, GPT-4

Retrieval Methods	Total Rows	Accuracy (Highest)	Accuracy (Top 10)	Accuracy (Top 30)
BM25	1350	86.54%	100%	100%
TF-IDF + Cosine Similarity	1350	67.70%	100%	100%
TF-IDF + Euclidean Distance	1350	67.55%	100%	100%
Word2Vec + Cosine Similarity + MMR	1350	33.70%	34.00%	34.00%
LSI	1350	21.82%	37.00%	45.00%
FAISS	1350	0%	1.00%	12.00%
Hybrid Search (BM25 : FAISS = 9:1)	1350	85.78%	98.00%	98.00%
Hybrid Search (BM25 : FAISS = 5:5)	1350	82.37%	98.00%	98.00%
Hybrid Search (BM25 : FAISS = 1:9)	1350	0.59%	0.59%	0.59%

Table 1: Retrieval method results for the Traditional RAG in the straightforward dataset

Model Name	Accuracy
GPT-4	88.78%
GPT-4o Mini	88.12%
GPT-3.5 Turbo	83.33%
Llama-3-8B-Instruct	76.54%
RoBERTa	56.16%
BERT	29.73%
DistilBERT	28.00%
DeBERTa	41.89%
mDeBERTa	53.33%
Electra	41.33%
Electra Large	41.33%

Table 2: LLMs results in straightforward dataset

achieves the highest performance (EM: 78.72%, EX: 80.85%), followed by GPT-4o Mini, Llama-3-8B-Instruct and GPT-3.5 Turbo, CRP consistently delivers better accuracy in most of LLMs, indicating the importance of detailed schema representation for SQL generation.

LLM	ODP		CRP	
	EM	EX	EM	EX
GPT-4	74.47%	78.72%	76.60%	80.85%
GPT-4o Mini	76.60%	70.21%	78.72%	80.85%
GPT-3.5 Turbo	38.30%	38.30%	25.53%	27.70%
Llama-3-8B-Instruct	31.91%	29.79%	68.83%	46.81%

Table 3: SQL RAG results on the straightforward dataset.

Table 4 presents the performance of Graph RAG, showing strong results across all models when using the schema prompt. GPT-4 leads with the highest accuracy (EM: 14.89%, EX: 91.49%), followed by GPT-4o Mini (EM: 10.64%, EX: 89.36%).

The differing EM and EX results between SQL RAG and Graph RAG indicate the differences between the two methods. In SQL RAG, the data is highly structured, leading to more fixed SQL queries and higher EM scores whenever we execute it. In contrast, Graph RAG shows a much lower EM but high EX, indicating that the graph query language is more flexible and can generate different formats while still providing highly accurate answers.

LLM	Schema Prompt	
	EM	EX
GPT-4	14.89%	91.49%
GPT-4o Mini	10.64%	89.36%
GPT-3.5 Turbo	10.64%	82.98%

Table 4: Graph RAG Results with schema prompt on the straightforward dataset.

5.1.2 Specialized airport jargon, abbreviations, and incomplete questions

As mentioned in the dataset section, we manually created a complicated, ambiguous dataset containing thousands of airport jargon, abbreviations, and incomplete questions. We classified these questions into six categories: Time Ambiguous Questions (TAQ), and Time With Ambiguous Flight Number Questions (TWAQ). Board Gate Questions (BGQ), Next Flight Questions (NFQ), Board Questions of Aircraft (BQA), and Ambiguous Flight Number Questions (AFQ).

Board Questions of Aircraft (BQA) and Ambiguous Flight Number Questions (AFQ) involve abbreviations and jargon, such as "Where is the delta?" and "At what gate is the 144?" Without the full airline names or additional flight details, these questions are challenging to answer. Time Ambiguous Questions (TAQ), Board Gate Questions (BGQ), and Time With Ambiguous Flight Number Questions (TWAQ) represent incomplete questions like "Which flight is currently at gate F09?" or "What's at C14?" These lack critical details such as flight numbers. Next Flight Questions (NFQ), on the other hand, are dynamic and will be discussed further in a later section.

We analyzed 220 questions in total to evaluate the robustness of the question classification prompt. Since large language models (LLMs) showed some variability in each time response, we employed a few-shot learning approach by integrating 60 carefully selected question classification examples

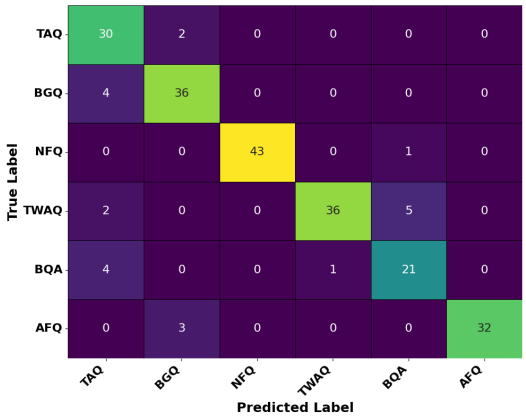

Figure 5: Confusion Matrix of Question Classifications

within the context window into the prompt. These 60 examples included six different questions and their correct categories. We repeated the classification experiments five times on the same questions. The accuracies for these five times' classification runs were 90.45%, 90.45%, 90.91%, 90.45%, and 90.00%, the average accuracy is 90.45%. The low variance among these runs suggests our prompt is robust and effective. Few-shot learning with extensive examples significantly improved accuracy and ensured consistent performance for different question types.

The final classification results are shown in the Figure 5. Although most questions were classified correctly, about 22 questions were misclassified. However, TAQ and BGQ share the same subsequent step of extracting a gate number, so swapping them does not affect outcomes. Similarly, TWAQ and BQA both prompt users for additional information; hence the confusion between these two also does not have too much impact on final results. When TWAQ or BQA are misclassified as TAQ, the system fails to extract a gate number, returns ['0'], and prompts the user for more details before re-running RAG. Because subsequent steps rely on correct classification, we added additional measures to mitigate the impact of misclassification. Our experiments show that most errors occur within these similar categories, and we have worked to minimize them as much as possible. Further details on the question classification prompts are provided in Appendix A.

5.1.3 Dynamic questions

The Next Flight Questions (NFQ) involves two situations: determining the next flight from the same airline or the same ramp. For the same airline, the answer is directly found in the table 'connecting flight number'. For the same ramp, we need to determine the expected on-ramp time for the current flight and then identify the closest expected on-ramp time for other flights at that ramp. Dynamic questions require additional calculation and reasoning. for example, if the question is 'What is the expected on-ramp time for the connecting flight of DL0123?' we must first identify DL0123's next connecting flight, then we can find its expected on-ramp time. We created a dataset of 30 reasoning questions to test RAG methods. As shown in Table 5, Graph RAG performed well, leveraging graph relationships for improved retrieval.

RAG Pipeline	Reasoning Question Dataset
Graph RAG	68.75%
SQL RAG	6.25%
Traditional RAG	9.38%

Table 5: Performance of different RAG pipelines on the reasoning question dataset

5.2 RQ2: How to reduce hallucinations?

Hallucinations mainly happen in traditional RAG when LLMs generate flight destinations not included in our dataset. This issue mainly exists in responses to complex and ambiguous queries. After performing question classification and retrieving flight information, we conducted few-shot learning with 20 examples, observing a hallucination rate of approximately 10%. This phenomenon is likely due to the excessive amount of information included in the input prompts for traditional RAG, which increases the likelihood of hallucination compared to SQL RAG and Graph RAG. Additionally, airline companies often reuse flight numbers, leading to conflicting data in LLM training and causing the generation of information absent from the dataset.

SQL RAG and Graph RAG reduce hallucinations by converting natural language questions into SQL or Cypher queries. Thereby, the input to the LLM is accurate data, which significantly reduces hallucinations. However, if the question requires a lot of context, the conversion to a query may fail.

It is important to note that hallucinations are not common even in traditional RAG and are not eliminated in SQL RAG or Graph RAG. Additionally, calculating the exact accuracy or rate of hallucinations across these RAG methods is challenging. However, SQL RAG and Graph RAG tend to reduce the occurrence of hallucinations compared

to traditional RAG. Given the high safety requirements in airport and aviation environments, SQL RAG and Graph RAG are safer for aviation operations. Both support dynamic storage of real-time flight information. Among them, Graph RAG performs better due to its stronger reasoning capabilities, enabling it to handle more complex queries effectively. More details of the experiment are provided in the Appendix A.

6 Conclusion

Our evaluation of three RAG methods shows that of the traditional RAG methods, BM25+GPT-4 is more efficient than other methods, because of the terminology used in the airport. However, traditional RAG can produce hallucinations, which poses a safety risk. SQL RAG and Graph RAG produce fewer hallucinations, and Graph RAG on top has higher accuracy. Our overall system effectively handles specialized airport terminology through question classification and prompt engineering; specifically, we address airport jargon and abbreviations. Graph RAG is particularly effective in handling reasoning tasks and questions about dynamic data, making it efficient in the airport domain.

7 Future Work

In our current research, the experiments are based on a static environment that does not capture any real-time changes such as delays or gate changes. In future research, we plan to connect the system with live APIs that provide real-time flight status and gate information, so that the system can dynamically retrieve and use real-time data. Another limitation is the relatively small size of our current dataset. In future work, we want to significantly expand and diversify the dataset. A larger and more diverse dataset will help ensure that our performance improvements hold across different scenarios and strengthen the validity of our conclusions.

Limitations

We openly acknowledge that this study is not a finalized product but an initial research investigation. The system's performance in the real world has not been demonstrated; it is a prototype that was tested in a controlled environment. Moreover, our evaluation is specific to the Schiphol airport domain; adapting the model to other airports or domains may present new challenges. Any deployment would need careful incremental trials, user feedback, and regulatory compliance checks to meet the high-reliability standards expected in aviation contexts.

Ethics Policy

Our research uses a dataset that has been authorized by Amsterdam Airport Schiphol and contains outdated flight information. Most of the flight information is publicly available online and does not include sensitive information. In this paper, the dataset is not publicly released, and it is only used to discuss its structure and to provide examples of question-answer pairs. No personal or confidential data are involved. Importantly, this work is an exploratory study focused on benchmarking performance in a controlled environment without impacting actual airport operations. We have implemented methods to reduce AI hallucinations—a key safety concern in this domain. However, any future deployment would require additional security reviews and strong safeguards to prevent misuse.

Acknowledgments

This work was supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Research Unit FOR 5339 (Project No.459291153). We also gratefully acknowledge Amsterdam Airport Schiphol for providing the raw dataset used in this study, and we thank our co-authors from the Royal Schiphol Group for their valuable contributions.

References

J. Barrasa, J. Webber, and J. Webber. 2023. *Building Knowledge Graphs: A Practitioner's Guide*. O'Reilly.

Xiaoyin Chen and Sam Wiseman. 2023. Bm25 query augmentation learned end-to-end. *Preprint*, arXiv:2305.14087.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *Preprint*, arXiv:2404.16130.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *Preprint*, arXiv:2308.15363.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-augmented generation for large language models: A survey. *Preprint*, arXiv:2312.10997.

Godwin George and Rajeev Rajan. 2022. A faiss-based search for story generation. In *2022 IEEE 19th India Council International Conference (INDICON)*, pages 1–6.

Chunxi Guo, Zhiliang Tian, Jintao Tang, Shasha Li, Zhihua Wen, Kaixuan Wang, and Ting Wang. 2023. Retrieval-augmented gpt-3.5-based text-to-sql framework with sample-aware prompting and dynamic revision chain. *Preprint*, arXiv:2307.05074.

Sven Jacobs and Steffen Jaschke. 2024. Leveraging lecture content for improved feedback: Explorations with gpt-4 and retrieval augmented generation. *Preprint*, arXiv:2405.06681.

Omid Jafari, Preeti Maurya, Parth Nagarkar, Khandker Mushfiqul Islam, and Chidambaram Crushev. 2021. A survey on locality sensitive hashing algorithms and their applications. *Preprint*, arXiv:2102.08942.

Hervé Jegou, Matthijs Douze, and Jeff Johnson. 2017. Faiss: A library for efficient similarity search. Facebook AI Research, Data Infrastructure, ML Applications. Accessed: insert access date.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Preprint*, arXiv:2005.11401.

Cai-zhi Liu, Yan-xiu Sheng, Zhi-qiang Wei, and Yong-Quan Yang. 2018. Research of text classification based on improved tf-idf algorithm. In *2018 IEEE International Conference of Intelligent Robotic and Control Engineering (IRCE)*, pages 218–222.

Antoine Louis, Gijs van Dijck, and Gerasimos Spanakis. 2023. Interpretable long-form legal question answering with retrieval-augmented large language models. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24)*, Maastricht University, Law & Tech Lab. Under review. Code available at https://arxiv.org/abs/2309.17050.

Yuning Mao, Yanru Qu, Yiqing Xie, Xiang Ren, and Jiawei Han. 2020. Multi-document summarization with maximal marginal relevance-guided reinforcement learning. *Preprint*, arXiv:2010.00117.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *Preprint*, arXiv:1301.3781.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *Preprint*, arXiv:2204.00498.

Heidi Steen Robert Lee. 2024. Hybrid search using vectors and full text in azure ai search. *https://learn.microsoft.com/en-us/azure/search/hybrid-search-overview*.

Stephen Robertson. 2004. Understanding inverse document frequency: On theoretical arguments for idf. *Journal of Documentation - J DOC*, 60:503–520.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3:333–389.

Bhaskarjit Sarmah, Benika Hall, Rohan Rao, Sunil Patel, Stefano Pasquali, and Dhagash Mehta. 2024. Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction. *Preprint*, arXiv:2408.04948.

Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. 2024. Hallucination is inevitable: An innate limitation of large language models. *Preprint*, arXiv:2401.11817.

A Appendix

A.1 Data Generation

In this section, we explained the methods to generate ground truth datasets including Question generation and Question classification.

A.1.1 Question Classification

To classify the questions, a flight information dataset is used to create different categories of questions. The flight information dataset contains information for thousands of flights which include several key items: The flight number identifies a specific flight, aircraft category, bus gate, bus service needed (remote or none), flight UID, direction (departure or arrival), ramp, main ground handler, expected on-ramp time, expected off-ramp time, connecting flight number, connecting flight UID, modified date and time, previous ramp, aircraft registration, flight state, MTT (minimum transfer time), MTT single leg, EU indicator, safe town airport (J or P), scheduled block time, best block time, expected block time, expected tow-in time, expected tow-off time, actual final approach time, actual block time, actual take-off time, actual boarding time, actual tow-in request time, actual tow-off time, actual on-ramp time, actual off-ramp time, flight nature, push back, and pier. Based on this flight information, we make some classifications for the questions.

The Question Classification pipeline is shown in Figure 6 Multiple types of questions need to be addressed in the project. Firstly, there are Heterogeneous datasets, which contain different formats

of datasets, including static data and dynamic data. Static data are flight information that remains constant for example, flight number, flight uid, EU indicator, flight nature, etc. while dynamic data are the flight information that changes dynamically, such as the time information expected on-ramp time, expected off-ramp time, modified date and time, connecting flight number, etc. this information are changed dynamically. Secondly, there are communication specifics of operations specialists' questions, which require handling abbreviations and short sentences. Thirdly, there are ambiguity resolution questions, which include ambiguity questions such as airport slang, and short questions that assume context. For example, some user questions are very short and not clear, such as "What is at A74?" or "Delta airline, any information?" These types of questions are also taken into consideration.

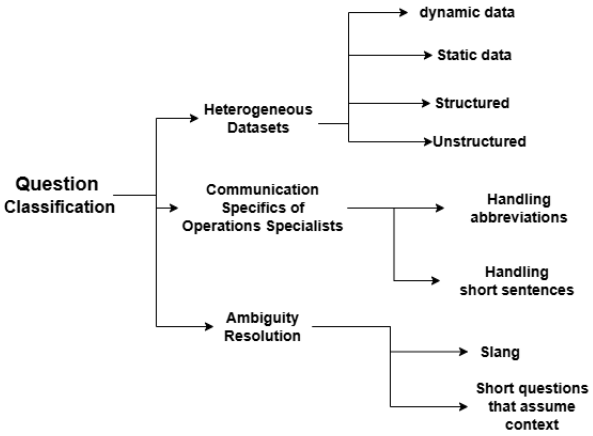

Figure 6: Question Classifications

To evaluate how effectively different retrieval methods performed, several tests were run. Two ground truth datasets were created: a straightforward dataset and a complicated and ambiguous dataset. The straightforward dataset contains questions without any ambiguities, which can be directly retrieved from flight information articles, Examples of straightforward questions were: "What category of aircraft is designated for flight KL1000?", "Which ramp is assigned for flight KL1000?", and "What time is the expected on-ramp for flight KL0923?" Such questions are easily identified for retrieval methods, which enables the selection of the most relevant articles.

The second type of dataset is a complicated and ambiguous dataset; it is made of variables that may be ambiguous and missing from the flight information dataset. Examples of such questions are: "It is now 2023-05-14 18:07:34+0000. Which flight is at gate B24?" or "Can you tell me which flight is scheduled at gate B24 for 2023-05-14 18:07:34+0000?" The gate number remains a constant variable in this case, but the given time indicates a random variable that is one hour before the scheduled block time in the flight table. When there aren't sufficient keywords, the pipeline finds it very challenging to find the exact correct content to generate correct answers. Besides, the complicated and ambiguous dataset also includes questions with ambiguous information, such as "Which flight is at gate B24?" These questions lack specific time and aircraft. Which results in multiple flight information that mention gate B24. In addition, many articles contain the B24 gate, in this case, BM25 is capable of retrieving all articles containing B24 as a keyword, and it returns correct articles within the top 30 results, indicating that the relevant article is among these top 30 articles.

A.1.2 Question generation

This part includes how to generate benchmark datasets.

As previously mentioned, we manually created two benchmark datasets: a straightforward dataset and a complicated/ambiguous dataset. The straightforward dataset contains questions that can be directly answered using flight information tables. In contrast, the complicated/ambiguous dataset includes more vague questions that depend on variables like time, airline, and flight number. For example, the question "Which flight is in B24?" could refer to many flights, so additional information is needed for an accurate answer. To generate the straightforward dataset, we created question templates with placeholders like: "Is there a problem with aircraft separation at <gate_nr>?" "What airlines have flights departing from gate <gate_nr>?"; "Can you tell me the aircraft category for flight <flight_number>?" We then manually filled these placeholders with actual gate and flight numbers from the flight information table.

To enrich our questions, we used language models to generate more variations. To enrich our questions, we used language models to generate more variations. For example, as Figure 7 shows, we took the question "What is the aircraft category for flight [flight_number]?" and prompted the model: We provide prompts like: "For each example question, please generate new, unique questions similar to the examples given, Do not repeat any spe-

cific flight numbers or questions from the examples. Use '[flight number]' as a placeholder for the flight number. Return only the question text." The same method will also be used for other types of straightforward questions. After that, the exact values in the placeholders such as [flight number], [ramp], [bus gate], etc., will be queried from the flight information dataset manually.Using this approach, we generated thousands of straightforward questions to test the performance of the conversational AI system. During our experiments, we randomly selected 150-200 question-answer pairs from the straightforward dataset. When evaluating different RAG methods and the performance of language models, we manually labeled each response as 'True' or 'False' to calculate accuracy.

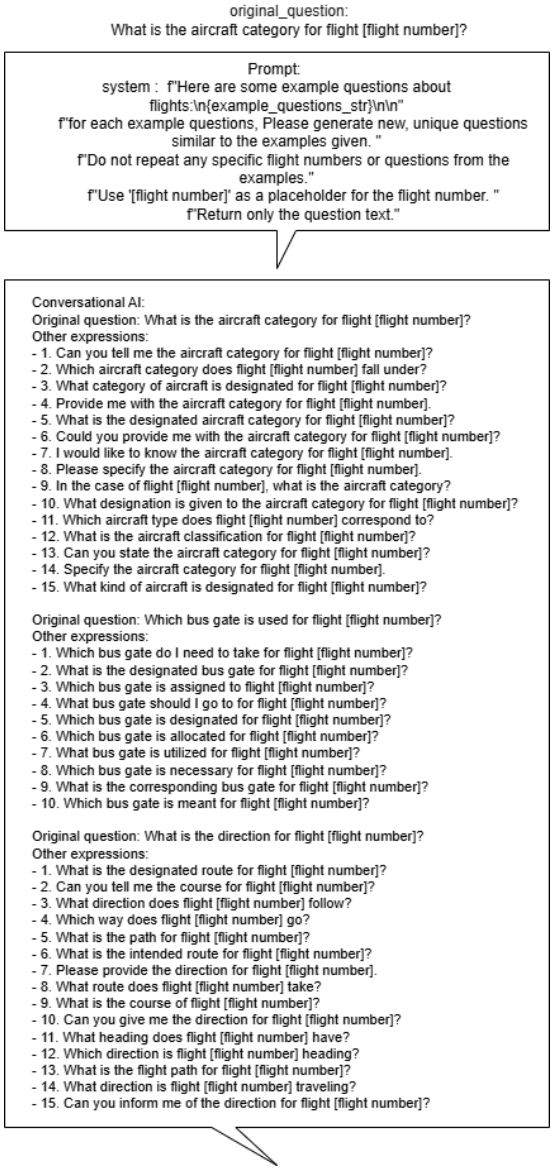

Figure 7: Dataset generation examples of straightforward questions

Similarly, for the ambiguous and complicated dataset, examples of such questions include: "It is now 2023-05-14 18:07:34+0000. Which flight is at gate B24?" or "Can you tell me which flight is scheduled at gate B24 for 2023-05-14 18:07:34+0000?" In these cases, the gate number is constant, but the provided time varies—usually set to one hour before the scheduled block time in the flight table. When keywords are insufficient, the system struggles to find the exact information needed for correct answers. The dataset also contains questions with ambiguous information, such as: "Which flight is at gate B24?" These questions lack specific time or aircraft details, leading to multiple flights associated with gate B24. As Figure 8 showed, during our experiments, we classified these complicated questions. We randomly selected 100-200 question-answer pairs, manually labeled their categories for question classification, and marked their prompt engineering results as 'True' or 'False' after classification.

A.2 Experiment

A.2.1 Question Classification and Prompt Engineering

In this step, questions are categorized into six types: Time Ambiguous Questions, Board Gate Questions, Next Flight Questions, Time with Airline Questions, Board Questions of Airline, and Ambiguous Flight Number Questions. The definitions of these question types are given to LLMs, and prompt them to response values from ['1'] to ['6']. Time Ambiguous Questions are questions that include specific times or terms referring to the current moment such as 'currently', 'at this moment', 'right now', 'now', 'when', 'last hour', 'next hour', etc., returning ['1']. Board Gate Questions are the questions that include gate numbers, like B24, A74, and C07, and are brief in length. returning ['2']. Next Flight Questions are the questions that inquire about a flight number and its next or connecting flight. returning ['3']. Time with Aircraft Questions involve references to time—exact moments or terms like 'right now', 'later', 'soon', 'a while', 'one hour ago', etc., and also mentions airline names like KLM, Delta, Transavia, EasyJet...etc, returning ['4']. Board Questions of Aircraft includes airline names, such as KLM, Delta, Transavia, EasyJet, etc.. returning ['5']. Ambiguous Flight Number Questions are the queries containing flight numbers that may have been ignored in the airline prefix,

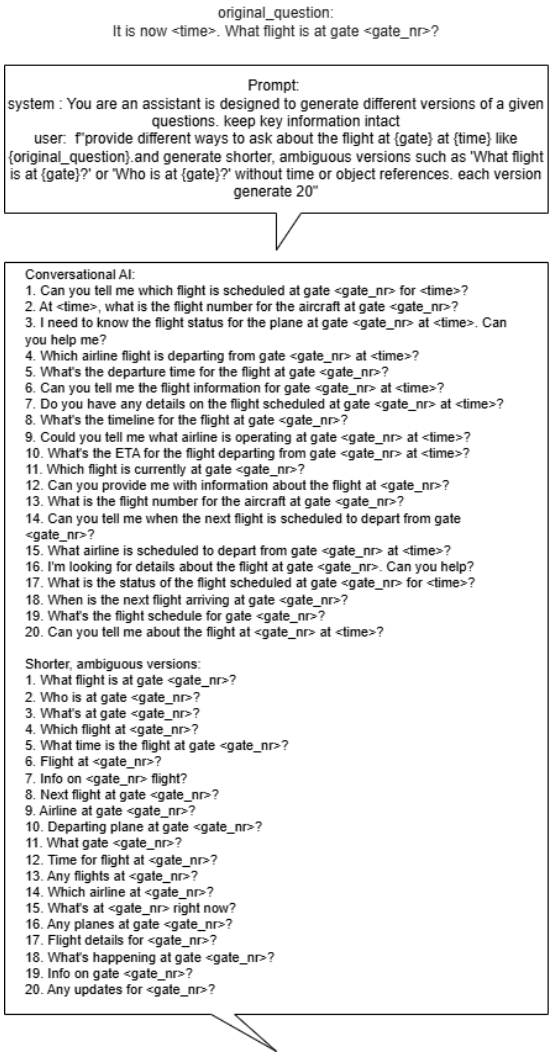

Figure 8: Dataset generation examples of complicated/ambiguous questions

for example, "Which gate is assigned to the 0164 flight?", "At what gate is the 0164?". These flight numbers might be incomplete, possibly consisting only of numbers, or may include letters but do not directly mention a specific airline's name, then these are Ambiguous Flight Number Questions return ['6']. The details of this prompt are shown in the Figure 9

After classifying questions, we use different prompts for each type. For Time Ambiguous Questions and Board Gate Questions, we direct them to prompts that extract gate numbers from the queries. In traditional Retrieval Augmented Generation (RAG), we can query the ramp or gate table for language models. For example, the ambiguous question "Which airline at A74?" is reformatted to extract the ramp number like ['ramp': 'A74']. If it does not extract the ramp information successfully, it will return ['0']. After testing these prompts, we achieve over 80% accuracy in extracting gate numbers. We then retrieve all tables containing these gate numbers from the flight information database. These tables, along with the original question, are used to generate answers. SQL RAG and Graph RAG directly generate query languages based on the ramp and gate numbers.

Questions that include airline names, like Time with Aircraft Questions and Board Questions of Aircraft, don't provide enough information because the airport has many flights from the same airline. Therefore, we prompt users to provide more details. For example, if someone asks, "Which flight is at Delta?", we respond: "I cannot determine the specific location of the Delta flight with the information provided. Please provide additional information like: - Flight UID (Unique Identifier) - Flight Number (Flight_NR) - Aircraft Registration - Connecting Flight UID (The UID of any connected flight provided by the airline) - Connecting Flight Number (The number of any connected flight provided by the airline). If you do not have this information, I can still attempt to process your query but it might require additional search time. In this case, please let me know if you are looking for information about the Ramp (Gate), Bus Gate, or Pier." After the user provides more information, we use the RAG methods again.

For Next Flight Questions, there are two scenarios: the next flight is from the same airline or the same departure ramp. If it's the same airline, we return the connecting_flight_nr from the table. If it's the same ramp, we find all flights at that ramp and identify the one with the closest on-ramp time. To handle these questions, we write prompts for the RAG methods to find the relevant results. For the Ambiguous Flight Number Questions, we would like to extract the number and match it with the real-time flight APIs to find the relevant flights that contain those mentioned numbers. However, Since we were researching in the static environments, we responded: " We could not find more information about the flight number you mentioned, could you please provide us with more information?"

A.2.2 SQL RAG

the OpenAI Demonstration Prompt(ODP) and Code Representation Prompt(CRP) prompts are showed in Figure 10, Figure 11

The ODP, as shown in Figure 10, focuses on simplicity and explicit rules. It lists table names and their respective columns without additional data

For the classification of questions, please adhere to these guidelines:
1. Time Ambiguous Questions (type ID 1): Questions containing specific times or terms that refer to the current moment such as 'currently', 'at this moment', 'right now', 'now', 'when', 'last hour', 'next hour', etc., are considered Time Ambiguous Questions. Return ['1'].
2. Board Gate Questions (type ID 2): Questions that include gate numbers, like B24, A74, and C07, and are brief in length, are classified as Board Gate Questions. Return ['2'].
3. Next Flight Questions (type ID 3): If the question has a flight number and inquires about the next flight or connecting flight, it is classified as Next Flight Question. Return ['3'].
4. Time with Aircraft Questions (type ID 4): If the question includes references to time, whether exact moments or terms like 'right now', 'later', 'soon', 'a while', 'one hour ago', etc., and also mentions airline names like KLM, Delta, Transavia,EasyJet...etc. It falls under Time with Aircraft Questions. Return ['4'].
5. Board Questions of Aircraft (type ID 5): When the question mentions an airline name like KLM, Delta, Transavia,EasyJet...etc. it is identified as Board Questions of Aircraft. Return ['5'].
6. Ambiguous Flight Number Questions(type ID 6): refers to queries containing flight numbers that may have omitted the airline prefix. These flight numbers might be incomplete, possibly consisting only of numbers, or may include letters but do not directly mention a specific airline's name. If a question appears to inquire about specific flight information, and the flight number seems incomplete or is presented in a non-standard format (such as using verbal numbers), or unconventional numerical expressions (for example, "seven eight seven") classify such questions as Type 6.
If a question matches other categories but explicitly mentions an incomplete flight number (lacking the airline code), it should primarily be classified under Ambiguous Flight Number Questions(type ID 6)
Ambiguous Flight Number Questions(type ID 6) examples: "Which gate is assigned to the 0164 flight?", "At what gate is the 0164?", "At what gate is the seven eight seven?"
Other Priority Rules: Time Information > Gate Number. Therefore, if a question contains both time and gate number (e.g., 'Which flight is currently at gate C07?'), it should be categorized as a Time Ambiguous Question, even if it includes a gate number. Questions containing only time information without an airline name are also Time Ambiguous Questions, for instance: 'Can you tell me the flight information for C07 at 2023-05-14 17:07:39+0000?' return ['1']! if there are gate numbers and time at the same time then return ['1'] not ['4']!
Be attention that Time Ambiguous Questions(['1']) must NOT contain any airline names! and Time with Aircraft Questions (['4']) must contain airline names in the question! so 'At 2023-05-14 17:07:39+0000, what is the flight number for the aircraft at C07?' is Time Ambiguous Questions don't have any airlines in the question. so it also returns ['1']!!! not ['4']!!
For questions that contain both time information and an airline name, if they are more detailed, classify them under Time with Aircraft Questions (['4']). If the question contains only the airline name and is short, it is a Board Question of Aircraft (['5']).
If a question is brief, containing only a gate number without time or airline details, it should be classified as a Board Gate Question (['2']). (e.g., 'Which airline at A65?'), it is to be classified as a Board Gate Question (['2'])
notice, if questions are like: "Flight number at C14?", "Airline at gate B15?", and "Info on A65 flight?", there are "flight", "Flight Number", and "airline" terms like these in the questions, but the word is very general, it is not specific Flight number or a specific airline name, but the gate number is very clear, then this is: Board Gate Questions (type ID 2) return ['2']
Please classify the questions accurately based on the above rules.

Figure 9: Prompts of Question Classifications

types or constraints. The ODP style emphasizes straightforward task instructions, such as "Include only valid SQL syntax, without additional formatting or explanation" guiding the model to generate SQL queries directly without unnecessary explanations.

In contrast, the CRP, shown in Figure 11, adopts a detailed SQL-style schema description. This approach uses CREATE TABLE statements to include comprehensive database information, such as column types and relationships (e.g., primary and foreign keys). By simulating database creation scripts, CRP uses the model's coding capabilities to enhance query precision, especially for complex databases with intricate relationships.

ODP is suitable for simpler, direct tasks, while CRP is better for handling more complex databases with comprehensive schema context.

A.2.3 Graph RAG in dynamic dataset

The Prompt of Graph RAG is shown in Figure 15, focusing on guidelines for writing a Cypher query. The schema is extracted from the Neo4j graph database using the APOC plugin, specifically through 'CALL apoc.meta.schema() YIELD value RETURN value', and then used in the prompts. As shown in Figure 16, Graph RAG enables flights to be connected through their relationships, allowing retrieval of detailed information about connecting flights. In contrast, traditional RAG and SQL RAG

```
Prompt：f “Complete SQL query only for the MySQL
database flight1. Include only valid SQL syntax,
without additional formatting or explanation.
Tables in the database flight1, with their properties:

flight_no_sensitive_information3(aircraft_category,
bus_gate, bus_service_needed, flight_nr, flight_uid,
direction, ramp, main_ground_handler,
expected_onramp, expected_offramp, ……)

Translate the following natural language query into a
SQL query:
Question: {Question}

SELECT ”

messages=[
        {"role": "system", "content": "You are a
SQL expert. Translate natural language queries to
SQL."},
        {"role": "user", "content": prompt} ]
```

Figure 10: ODP Prompt for SQL RAG

```
Prompt: f ‘Complete SQL query for the MySQL database `flight1`
only.
Include only valid SQL syntax, without additional formatting or
explanation.
Tables in the database `flight1`, with their properties:

CREATE TABLE flight_no_sensitive_information3 (
   int,
   aircraft_category int,
   bus_gate text,
   bus_service_needed text,
   flight_nr text,
   flight_uid text,
   direction text,
   ramp text,
   main_ground_handler text,
   expected_onramp text,
   expected_offramp text,
……
);

Question: What is the expected onramp time of flight KL0618?
SELECT ’

messages=[
        {"role": "system", "content": "You are a SQL expert.
Translate natural language queries to SQL."},
        {"role": "user", "content": prompt} ]
```

Figure 11: CRP Prompt for SQL RAG

treat connecting flights as merely a column in the table, limiting access to further relational information.

A.2.4 Hallucination Analyses

This section provides additional information about hallucinations in different RAG methods.

The hallucination in traditional RAG is illustrated in Figure 12. Using the question "Which flights are at B18" as an example, this query is classified as a Board Gate Question (BGQ) during the question classification step. For traditional RAG, the gate number "B18" is extracted from the question, and all table rows containing "B18" are retrieved. These rows, along with the question itself, are then passed to the LLM to generate the final answer. However, due to the large amount of flight information stored in LLMs, hallucinations are more likely to happen if the retrieval process brings in too much unrelated information.

In contrast, for SQL RAG and Graph RAG, the retrieval process is more precise. In SQL RAG (Figure 13), the natural language question is first converted into an SQL query that retrieves only the relevant information—in this case, flight numbers at gate B18. The results are then passed to the LLM to generate the final answer. Similarly, in Graph RAG (Figure 14), a Cypher query retrieves only the flight numbers associated with gate B18. Since both SQL RAG and Graph RAG retrieve more targeted and accurate information compared to traditional RAG, they significantly reduce the likelihood of hallucinations.

It is important to note that hallucinations are not common even in traditional RAG, and they are not eliminated in SQL RAG or Graph RAG. Additionally, calculating the exact accuracy or rate of hallucinations across these RAG methods is challenging. However, because SQL RAG and Graph RAG retrieve information more accurately, they tend to reduce the occurrence of hallucinations compared to traditional RAG.

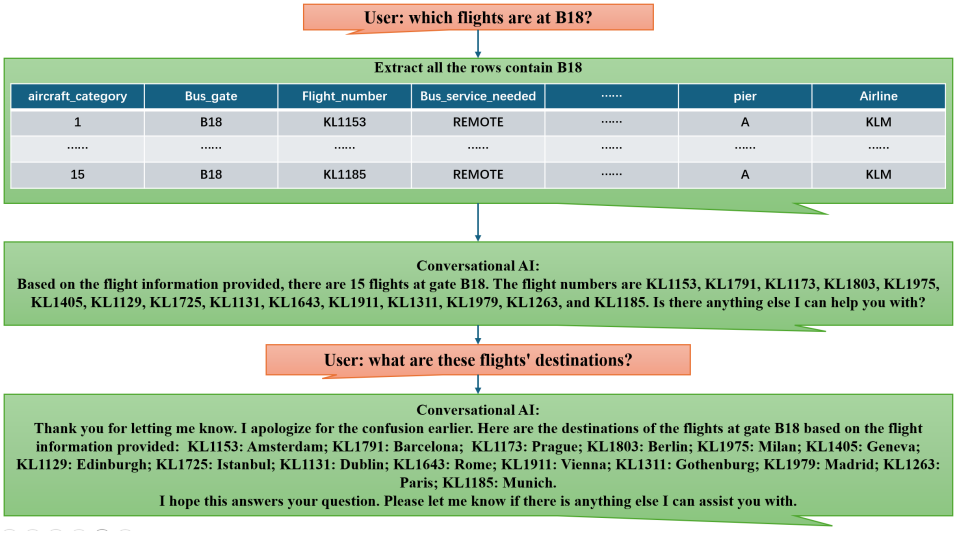

Figure 12: Hallucination case for traditional RAG

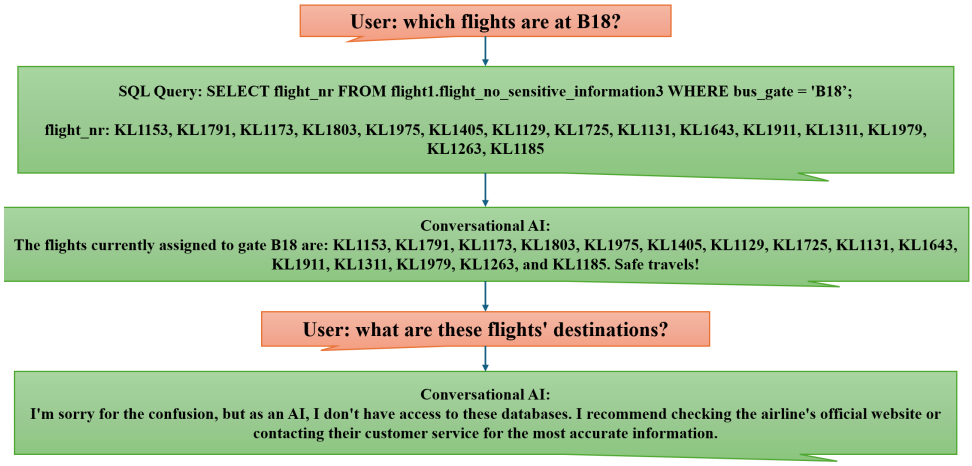

Figure 13: The same case for SQL RAG

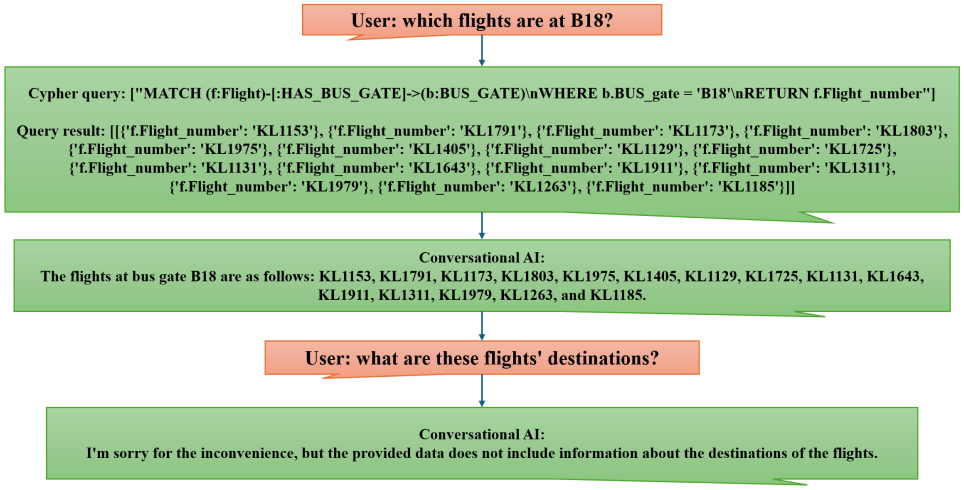

Figure 14: The same case for Graph RAG

Prompt: f 'You are a Cypher query expert for Neo4j. Your task is to convert the user's question into a precise and executable Cypher query that directly answers the question about flights, airlines, and airports.

****Instructions:****

- **Use the provided schema to generate the Cypher query.**
- **Output only the Cypher query, without any additional text, explanations, comments, or formatting.**
- **Ensure the Cypher query is syntactically correct and can be executed directly in Neo4j without any modifications.**
- **Do not assume any data not present in the schema.**
- **Use appropriate Cypher syntax and best practices.**

Schema:
{schema}

Question:
{question}

Provide only the Cypher query that answers the question.
messages=[
{"role": "system", "content": "You translate natural language questions into Cypher queries."},
{"role": "user", "content": prompt}
]

Figure 15: Graph RAG Prompt

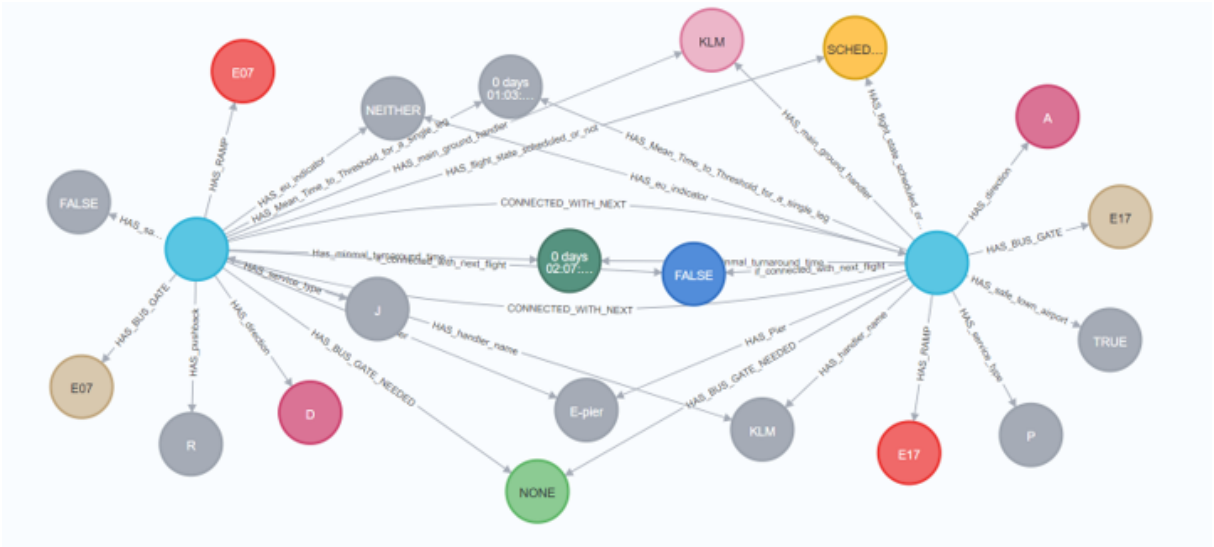

Figure 16: Graph RAG in Reasoning Questions