RAG System for Airbnb Queries in Padua and its province with Self-Query Retrieval

Laura Legrottaglie
laura.legrottaglie@studenti.unipd.it

Abstract

This project focuses on developing a Retrieval-Augmented Generation (RAG) system to handle user queries about Airbnb accommodations in Padua and its province. The system aims to enhance LLM capabilities by integrating external data like reviews, house rules, and metadata to deliver accurate, context-aware, and user-focused responses. The implementation includes two retrieval strategies: a naive similarity-based retrieval and a self-query retrieval mechanism, which automatically extracts metadata from user queries using Llama 2 (13B) combined with few-shot learning. The system also integrates a context-aware question-answering pipeline powered by Llama 2 (13B) using the maximum marginal relevance (MMR) approach for retrieval to further enhance response quality. The results demonstrate the ability of self-query retrieval to produce more accurate responses with reduced hallucinations, although both retrieval approaches achieve comparable F1 BERT scores of 0.83 due to the semantic similarity of their outputs.

1. Introduction

A significant area within Natural Language Processing is Natural Language Generation (NLG), which is dedicated to producing human-like text from structured or unstructured data. As AI continues to advance, the requirement for generated content to be factually accurate, as well as rich in context, is increasing. While the classical NLG models, including sequence-to-sequence architectures, have achieved great results regarding the fluency and coherency of the generated text, they are limited when handling queries that require knowledge extending beyond their training data. Models like GPT and BERT-based text generators have shown remarkable potential, but they face significant challenges. One notable issue is hallucinations, where the models generate content that appears credible but is inaccurate or fabricated.[6] Additionally, their knowledge is limited to the time of their last update, restricting their ability to provide insight into events or developments that have occurred afterward. Furthermore, these models often lack domain-specific expertise due to limited training in specialized areas, leading to inaccuracies when addressing technical or highly specialized topics.

All these challenges have driven the development of hybrid models that integrate retrieval mechanisms with generative capabilities. Retrieval-Augmented Generation (RAG) has gained recognition as an effective approach by integrating external information into the generation process, improving the accuracy and reliability of the generated content. Beyond general applications, as open-domain question answering, RAG systems excel in specialized domains such as medicine, law, and finance by providing expert-level insights. [10][12][18] Furthermore, they play a pivotal role in dialogue systems, such as chatbots and virtual assistants, enabling personalized and contextual-aware interactions by retrieving relevant information from external databases.

Building on these capabilities, this project focuses on developing an advanced RAG system designed to handle Airbnb-related queries for accommodations in the city of Padua and its province. Using reviews, house rules, and metadata, the system aims to provide accurate, relevant, and user-focused answers tailored to individual needs.

This report is structured as follows: the following section provides an overview of RAG systems, including their state-of-the-art developments, architecture, and core components, Section III reviews related works focusing on the existing research in the role of query's metadata extraction by LLMs to enhance retrieval accuracy. Section IV details the processing pipeline, covering dataset construction and preprocessing. Section V outlines the setup of the RAG system, while Section VI presents the experimental results and evaluates the generated answers. Lastly, Section VII concludes the report with a summary of key findings and final reflections on the project, exploring potential future advances to the model.

2. Literature overview

The architecture of a RAG system is composed of three key components: the retrieval module, the generation module and the augmentation module. The process begins with a user query, followed by the retrieval of relevant data from external sources. The retrieved data is then augmented and provided as input to the generation module, which produces the final response. Unlike conventional NLP approaches that rely primarily on the static knowledge encoded within a pre-trained model's parameters, RAG systems dynamically integrate external information into the response generation process. While fine-tuning adjusts a model's parameters to perform specific tasks, and prompt engineering creatively designs inputs to obtain desired responses, RAG goes further by actively expanding the model's knowledge to improve answers.

2.1. RAG's research strategies

It is possible to distinguish between three RAG search paradigms: naive RAG, advanced RAG and modular RAG.

2.1.1 Naive RAG

The naive RAG paradigm represents the simplest form of a RAG system. In this approach, the focus is on the direct integration of retrieved information into the generation model without making specific optimizations or adjustments.

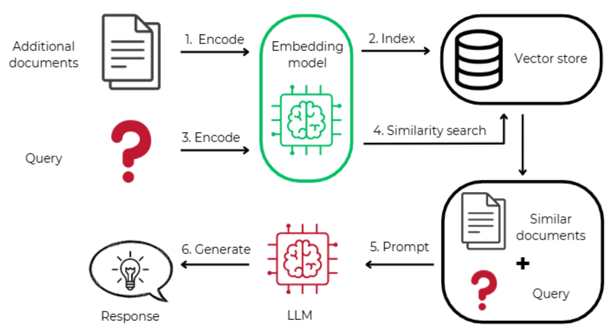

Figure 1. Naive RAG processing flow

It operates through three key stages:

- **Indexing** involves cleaning external data into plain text. The text is then divided into manageable segments, which are then transformed into vector embeddings using a model that captures their semantic meaning. These vectors are stored in a specialized database optimized for similarity searches.

- **Retrieval** begins when a user submits a query. The system converts the query into a vector representation using the same model applied during indexing. It then searches the database to identify and retrieve the most K relevant text segments, ranked by their similarity to the query.

- **Generation** involves constructing a comprehensive prompt using the retrieved segments and the user's query. A large language model processes this prompt to generate a response, drawing from either the retrieved information or its built-in knowledge, depending on the task requirements. In some scenarios, the system can also integrate prior conversational exchanges, enabling it to handle multi-turn dialogues effectively. [4]

2.1.2 Advanced RAG

In advanced RAG, complex techniques are used to improve the retrieval step. It's possible to divide advanced RAG techniques into three areas: Data-Indexing and Pre-Retrieval, Retrieval and Post-Retrieval.

- **Data-Indexing and pre-retrieval** focus on improving data quality and refining query clarity. This involves optimizing the indexing structure by increasing data granularity, adding metadata, removing redundancies through clustering or LLM-based deduplication, and summarizing verbose content to extract key facts.[7][2] Query optimization further refines the user's input through rewriting, transformation, or expansion to align it more effectively with retrieval systems. [22][9]

- **Retrieval** techniques optimize information retrieval by efficiently organizing and filtering data. Techniques like hierarchical index retrieval refine searches by starting broadly and narrowing results through subcategories.[7] Self-query retrieval extracts metadata directly from user queries to enable precise structured searches, while hybrid search combines keyword-based and semantic methods for balanced retrieval.[16] Additionally, graph search leverages knowledge graphs to uncover not just direct matches but contextually relevant insights.[21]

- **Post-retrieval** techniques ensure that the retrieved context is effectively integrated with the query. Re-ranking algorithms reorder documents based on relevance, while context compression prioritizes the most significant information, optimizing token usage within the language model.[3][5]

2.1.3 Modular RAG

The modular RAG paradigm represents the most advanced approach currently available in the development of RAG systems. Unlike earlier RAG paradigms, which primarily follow a fixed "retrieve-and-read" process, Modular RAG introduces a modular architecture where components such as retrieval, pre-processing, generation, and post-processing

operate independently. This modularity allows for targeted optimization and scalability across diverse use cases.
A key feature of Modular RAG is the introduction of specialized modules. Among these, the Search module adapts to various data sources, enabling direct searches using LLM-generated queries across databases, search engines, and knowledge graphs.[17] The Memory module exploits iterative self-enhancement to align retrieved information more closely with data distribution, storing useful data in an expandable memory pool.[1] Additionally, the Predict module reduces redundancy by generating relevant context directly through the LLM, ensuring precision and coherence.[19]
Innovative patterns within Modular RAG further enhance its functionality. Flexible workflows, such as the "Rewrite-Retrieve-Read" model, refine retrieval through query rewriting and LLM feedback mechanisms.[8] Other approaches, like "Generate-Read" replace traditional retrieval with LLM-generated content, while "Recite-Read" emphasizes retrieval from the model's internal knowledge.[20][13]

3. Related works

The approach adopted in this project utilizes a self-query retrieval mechanism that autonomously extracts metadata implicitly or explicitly contained in the user's query and retrieves documents based on the applied metadata filters. Several scientific papers have explored the integration of self-query mechanisms with LLMs to extract information from queries in RAG systems. Poliakov et al. [11] have introduced a novel approach called Multi-Meta-RAG that leverages database filtering with metadata extracted by LLMs to refine document selection, improving the accuracy of responses to complex multi-step queries. Tang et al. [14] have proposed Self-Retrieval, an innovative end-to-end framework that unifies indexing, retrieval, and reranking within a single LLM. This approach enhances retrieval accuracy by employing self-supervised learning and constrained decoding to generate and rank relevant passages effectively. Futhermore, Wang et al. [15] have introduced MIGRES, a framework that identifies and utilizes missing information to iteratively refine queries and filter irrelevant content in multi-hop question answering.

4. The dataset

The dataset has been collected using web scraping tools provided by Apify [1]. It consists of 12,867 textual entries, including reviews and house rules, from 112 Airbnb accommodations located in Padua and its province. Alongside the textual data, metadata has also gathered to provide additional context, including the name of the accommodation, its specific location, accommodation type (e.g., entire apartment or private room), number of bedrooms and bathrooms, and boolean property regulations (e.g., whether smoking, pets, infants, children, or events are allowed). After preprocessing and cleaning the dataset, all texts are translated from their original language into English to facilitate further analysis. Data exploration has revealed that most documents are under 100 words in length, with an average of 110 texts per accommodation.

[1] https://apify.com/triangle/airbnb-scraper

5. RAG system set-up

5.1. Text splitting

Before embedding documents in a vector store for retrieval, since some texts contain up to 500 words, they are splitted into smaller, overlapping chunks. This approach ensures that each piece fits within the model's input size constraints and results in more uniform embedding vectors. To address this, the *RecursiveCharacterTextSplitter* is used. This splitter divides the text into chunks of 1000 characters while maintaining an overlap of 100 characters between consecutive chunks to preserve context across splits. This is due to improve similarity comparisons during retrieval, as each chunk carries a comparable amount of context.

5.2. Embedding model

To embed accommodation documents for efficient retrieval, the HuggingFace Sentence-Transformers model all-MiniLM-L6-v2 [2] is employed. This lightweight model with six transformer layers is designed to generate embeddings that effectively capture semantic relationships within textual data. To optimize performance, a caching mechanism is implemented using a local file store. This ensures that previously computed embeddings are reused, reducing computational overhead for repeated queries. The model supports a maximum input length of 384 tokens, making it well-suited for the majority of the documents, with longer texts being truncated to this limit by default. It outputs 384-dimensional vectors, offering a good balance between capturing meaningful semantics and maintaining computational efficiency.

5.3. Vector database

The RAG system will be tested with a wide range of queries, including general questions about accommodations in Padua, such as pricing or general feedback, as well as more specific user requests. Specifically, for general questions, only the naive retrieval approach is utilized, as there are no metadata to extract for these types of queries. However, for queries related to specific user requests, such as amenities or features of the desired accommodation, both

[2] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

retrieval approaches can be compared. The vector database used in the implemented RAG system is ChromaDB, selected for its high performance and its support for metadata filtering.
In cases where metadata filtering is applied, documents relevant to the user's query are retrieved using the Maximum Marginal Relevance (MMR) method instead of using cosine similarity. The retrieval is based on the following formula:

$$\text{MMR} = \arg\max_{d_i \in D \setminus R} \left[\lambda \cdot \text{Sim}_1(d_i, q) - (1 - \lambda) \cdot \max_{d_j \in R} \text{Sim}_2(d_i, d_j) \right]$$

where D is the set of all candidate documents, R is the set of the already selected documents, q is the query, $\text{Sim}_1(d_i, q)$ is the similarity between the document d_i and the query q, referred to as the relevance term, and $\text{Sim}_2(d_i, d_j)$ is the similarity between the document d_i and a previously selected document d_j, referred to as the redundancy or diversity term. The parameter λ controls the trade-off between relevance and diversity, where $\lambda = 1$ maximizes relevance and ignores diversity, while $\lambda = 0$ maximizes diversity and ignores relevance.
In the proposed retrieval system, the λ parameter is set to 0.8, achieving a balance that slightly favors diversity while still maintaining a strong emphasis on relevance.
The vector store is utilized in the processing pipeline to retrieve the top 10 most relevant documents.

5.4. Automatic metadata extraction from a user's query

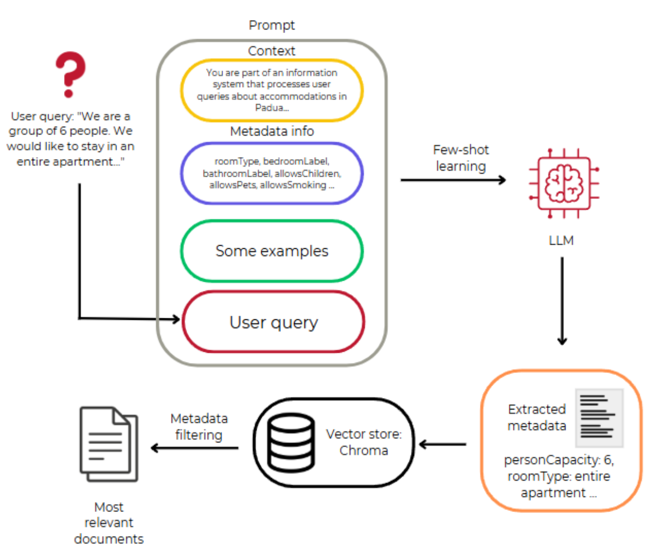

Figure 2. Self-Query Retrieval Methodology Employed

Metadata is automatically extracted from user queries using few-shot learning with Llama 2 (13B)[3] as the large language model. As shown in Fig. 2, the process involves crafting a prompt template that guides the LLM to analyze the query and extract only the relevant metadata fields from a predefined set. Examples of input queries and their corresponding metadata are provided in the prompt, enabling the LLM to learn the structure and context of the task, even with minimal examples. The designed prompt can be seen in the appendix (Section C).
The implementation then combines a text-generation pipeline with the structured prompt using *LangChain*. The Llama 2 model operates in 4-bit precision using an optimized NF4 quantization configuration, reducing the memory footprint and making the model faster. The temperature hyperparameter, which controls prediction randomness by scaling the output probability distribution, is set to 0.1, while the maximum output length is capped at 128 tokens. Extracted metadata will subsequently be utilized for filtering during vector store retrieval, enabling more precise and context-aware results.

5.5. Context-aware response generation

In the final stage of the RAG system, an additional text-generation pipeline is integrated for question-answering. This pipeline uses a well-defined prompt template that specifies a clear structure for generating informative responses. The template incorporates metadata from the retrieved documents, such as the number of bathrooms, bedrooms, room type, and other relevant features, which are passed into the template to enrich the context and enhance the quality of the model's responses. The designed prompt is the following: "You are an assistant for question-answering tasks. The theme of the questions is: AirBnB reviews and house rules in Padua and province. Use the following pieces of retrieved context to answer the question. Feel free to ignore some context if it is not useful for the answer. If you don't know the answer or the retrieved context does not contain any information about the question, clearly apologize and state that you don't know. Ensure all responses are well-supported by the retrieved documents".
The text generation process is configured with a low temperature of 0.1 to minimize randomness and produce highly focused outputs. Additionally, the maximum output length is set to 512 tokens to ensure the generation of complete, non-truncated responses.
The following section will assess the quality of the answers generated by the question-answering system, comparing the results obtained from both the naive retrieval approach and the self-query approach.

6. Results

Examples of the questions and the generated answers can be found in the appendix (Section D). To assess the

[3] https://huggingface.co/meta-llama/Llama-2-13b-chat-hf

performance of the system, the following metrics are used:

- BERTScore: this metric is used to measure the semantic similarity between the generated response and the user query using contextual embeddings from the pretrained transformer model BERT. Instead of relying solely on exact word matches or n-gram overlap, BERTScore evaluates how well the meaning of the generated text aligns with the query by comparing the dense contextual embeddings using cosine similarity. It outputs scores for precision, recall, and F1, with higher values indicating better semantic similarity.

- Perplexity is used to assess the quality and fluency of the generated text. It measures how well a language model, such as GPT-2 in this case, can predict the next tokens in a sequence. A lower perplexity score indicates that the text aligns well with the language model's expectations, suggesting it is more fluent and coherent, while a higher value indicates the presence of unnatural or awkward phrasing in the generated responses. Perplexity values below 20 are often considered good, while scores between 20 and 50 may still be acceptable, showing that the prediction of the next tokens is inherently harder.

- Human evaluation is used to provide a qualitative assessment of the system's performance judging relevance and coherence.

- Efficiency measures the response time of the system for each query assessing its computational efficiency and comparing retrieval strategies in terms of latency.

6.1. General questions

The system is able to provide well-structured answers with clear explanations. The responses are logically organized, highly coherent, and generally aligned with the questions, although they may occasionally include irrelevant or less plausible statements. The time needed to answer to the users' queries with naive retrieval ranges between 30 and 60 seconds. The F1 BERT scores achieve approximately 0.8, reflecting a strong semantic similarity between the generated responses and the queries, while the perplexity values are around 10, indicating high fluency of the generated text.

6.2. Specific user requests

For user-focused questions, such as requests for accommodations with specific amenities or characteristics, the system effectively provides multiple options, each with detailed information. The RAG system with naive retrieval generates relevant answers but they are usually subjected to hallucinations. For instance, it may suggest accommodations that do not fully meet the client's requirements, such as discrepancies in capacity, number of bedrooms or bathrooms, or house rules regarding pets and children. In contrast, while the self-query retrieval approach requires slightly more time — just a few seconds longer than the 50 seconds needed by naive retrieval — it effectively extracts metadata from the queries and answers to the questions listing properties that are highly targeted for the user with still some minor artifacts in the explanations. However, both retrieval methods achieve good perplexity values around 20, demonstrating their ability to generate fluent responses, and attain similar F1 BERT scores of around 0.83. This outcome can be attributed to the naive approach which generates semantically relevant responses through hallucinations. These hallucinations, in fact, make the generated text semantically relevant to the query, boosting its BERTScore even if its content is less accurate.

6.3. Unrelated questions

The system has also been tested on unrelated questions, like Japanese restaurants, and successfully states that the retrieved context lacks specific information, suggesting alternative resources, such as searching online or asking accommodation hosts.

7. Conclusion

This project successfully demonstrates the effectiveness of a RAG system designed to respond to queries about Airbnb accommodations in Padua. By incorporating metadata extraction and advanced retrieval strategies, the system provides accurate, coherent, and user-centric responses. The self-query retrieval approach, which extracts metadata directly from user queries, has proven to be effective in targeting relevant documents, offering more accurate and specific responses compared to the naive retrieval method. Both methods have achieved similar F1 BERT scores of approximately 0.83 due to the semantic similarity of their outputs. However, the self-query approach excels in factual accuracy, reducing hallucinations. This shows how advanced retrieval techniques can enhance relevance and precision, highlighting their potential for adoption in real-world and domain-specific applications.

While the results are promising, the relatively small size of the dataset limits the system's ability to generalize and address a broader range of queries. Expanding the dataset with additional reviews and metadata would provide richer context and improve the robustness of the retrieval and generation processes. Future improvements could also focus on boosting accommodations with fewer reviews to ensure balanced visibility, prioritizing more recent reviews to reflect more current conditions and incorporating metrics that better assess factual grounding.

References

[1] Xin Cheng, Di Luo, Xiuying Chen, Lemao Liu, Dongyan Zhao, and Rui Yan. Lift yourself up: Retrieval-augmented text generation with self memory, 2023.

[2] Peter Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(9):1537–1555, 2012.

[3] H. Zahera D. Vollmers, M. Ali and A. Ngonga Ngomo. Document reranking using gat-cross encoder, 2024.

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

[5] Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model, 2024.

[6] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55:1–38, 2023.

[7] Ramnath Kumar, Anshul Mittal, Nilesh Gupta, Aditya Kusupati, Inderjit Dhillon, and Prateek Jain. Ehi: End-to-end learning of hierarchical index for efficient dense retrieval, 2024.

[8] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting for retrieval-augmented large language models, 2023.

[9] Wenjun Peng, Guiyang Li, Yue Jiang, Zilong Wang, Dan Ou, Xiaoyi Zeng, Derong Xu, Tong Xu, and Enhong Chen. Large language model based long-tail query rewriting in taobao search, 2024.

[10] Nicholas Pipitone and Ghita Houir Alami. Legalbench-rag: A benchmark for retrieval-augmented generation in the legal domain, 2024.

[11] Mykhailo Poliakov and Nadiya Shvai. Multi-meta-rag: Improving rag for multi-hop queries using database filtering with llm-extracted metadata, 2024.

[12] Spurthi Setty, Harsh Thakkar, Alyssa Lee, Eden Chung, and Natan Vidra. Improving retrieval for rag based question answering models on financial documents, 2024.

[13] Zhiqing Sun, Xuezhi Wang, Yi Tay, Yiming Yang, and Denny Zhou. Recitation-augmented language models, 2023.

[14] Qiaoyu Tang, Jiawei Chen, Zhuoqun Li, Bowen Yu, Yaojie Lu, Cheng Fu, Haiyang Yu, Hongyu Lin, Fei Huang, Ben He, Xianpei Han, Le Sun, and Yongbin Li. Self-retrieval: End-to-end information retrieval with one large language model, 2024.

[15] Keheng Wang, Feiyu Duan, Peiguang Li, Sirui Wang, and Xunliang Cai. Llms know what they need: Leveraging a missing information guided framework to empower retrieval-augmented generation, 2024.

[16] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jiongkang Ni. Navigable proximity graph-driven native hybrid queries with structured and unstructured constraints, 2022.

[17] Xintao Wang, Qianwen Yang, Yongting Qiu, Jiaqing Liang, Qianyu He, Zhouhong Gu, Yanghua Xiao, and Wei Wang. Knowledgpt: Enhancing large language models with retrieval and storage access on knowledge bases, 2023.

[18] Guangzhi Xiong, Qiao Jin, Zhiyong Lu, and Aidong Zhang. Benchmarking retrieval-augmented generation for medicine, 2024.

[19] Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. Generate rather than retrieve: Large language models are strong context generators, 2023.

[20] Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. Generate rather than retrieve: Large language models are strong context generators, 2023.

[21] Xueli Yu, Weizhi Xu, Zeyu Cui, Shu Wu, and Liang Wang. Graph-based hierarchical relevance matching signals for ad-hoc retrieval. In *Proceedings of the Web Conference 2021*, WWW '21, page 778–787. ACM, Apr. 2021.

[22] Shengyao Zhuang, Houxing Ren, Linjun Shou, Jian Pei, Ming Gong, Guido Zuccon, and Daxin Jiang. Bridging the gap between indexing and retrieval for differentiable search index with query generation, 2023.

APPENDIX

A. Word cloud

Figure 3. Word cloud of most frequently mentioned terms in documents

B. Distribution of document length (in words)

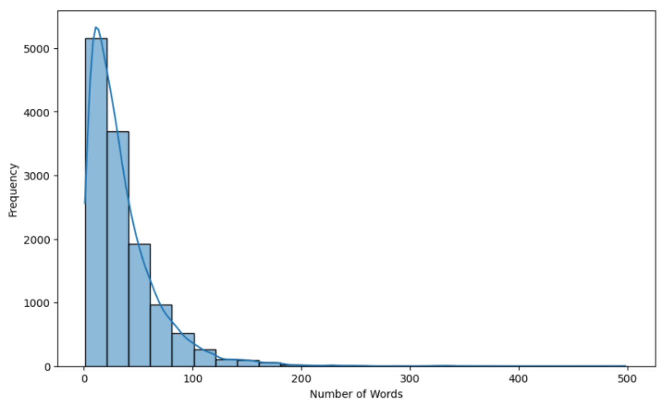

Figure 4. Distribution of document length in words: most texts fall below 100 words in length

C. Prompt used in the self-query approach

You are part of an information system that processes user queries about accommodations in Padua. Your job is to analyze the user's query and extract the relevant metadata fields. Do not output or add any extraneous field which is not part of the list below.
Here is the complete list of possible fields:

- roomType
- bathroomLabel
- bedroomLabel
- allowsChildren
- allowsInfant
- allowsEvent
- allowsPets
- allowsSmoking
- personCapacity
- isHostedBySuperhost

Answer only to the example 5.
Example 1:
Query: "I would like to stay in a room in a shared apartment with private bathroom in Padua. I also have a dog, so pets should be allowed. What is the best accommodation for me?"
Extracted metadata fields: {"roomType": "Private room", "bathroomLabel": "1 private bathroom", "allowsPets": True}

Example 2:
Query: "I am planning a trip near Padua. We're in 4 and we would like to have 2 bedrooms, since there are two children. Some of us also smoke, so it would be better to have a place where it is possible to do so. What is the best apartment suggested?"
Extracted metadata fields: {"personCapacity": 4, "bedroomLabel": "2 bedrooms", "allowsChildren": True, "allowsSmoking": True}

Example 3:
Query: "We are a family with a toddler. We would like to stay in an entire apartment with 2 bathrooms in the centre of Padua. I prefer apartments hosted by a superhost since I think they are more reliable. We would like to invite some friends for Christmas and do a small party in the house. Which is the best solution to our needs?"
Extracted metadata fields: {"roomType": "Entire accommodation", "bathroomLabel": "2 bathrooms", "isHostedBySuperhost": True, "allowsInfant": True, "allowsEvent": True}

Example 4:
Query: "I need to do a medical visit. Which apartment is closer to the hospital? I also need 2 rooms since I have children."
Extracted metadata fields: {"bedroomLabel": "2 bedrooms", "allowsChildren": True}

Example 5:
Query: {query}
Extracted metadata fields:

D. Examples of questions and generated answers

D.1. Examples of general questions

QUESTION: How are properties hosted by superhosts compared to other listings in Padua?

MODEL ANSWER: Based on the provided context, it appears that properties hosted by superhosts in Padua have several advantages. Firstly, they are more likely to have a higher rating and more reviews, indicating a better guest experience. Secondly, they are more likely to have a well-maintained house space, with cleanliness being a top priority. Additionally, superhosts are more likely to provide detailed and timely information about everything guests need, making the check-in process easier and more convenient. Finally, superhosts are more likely to have a more central location, with some listings being within walking distance to downtown Padua. Overall, properties hosted by superhosts in Padua appear to offer a more comfortable, convenient, and high-quality stay compared to other listings.

QUESTION: What are the recurring complaints in Airbnb reviews of properties in Padua?

MODEL ANSWER: Based on the reviews provided, the following are recurring complaints in Airbnb reviews of properties in Padua:
1. Lack of smoking: Many reviews mention that the property does not allow smoking, which may be a concern for smokers.
2. Limited parking: Some reviews mention limited parking options, which may be a challenge for guests with cars.
3. Noise: A few reviews mention noise from the street or other guests, which may be a concern for those who prefer a quieter environment.
4. Cleanliness: One review mentions that the property was not as clean as expected, which may be a concern for guests who prioritize cleanliness.
5. Infant policy: Some reviews mention that the property has a no-infant policy, which may be a challenge for families with infants.
6. Limited space: A few reviews mention that the property has limited space, which may be a concern for guests who prefer more spacious accommodations.
7. Lack of amenities: Some reviews mention that the property lacks certain amenities, such as a kitchen or laundry facilities, which may be a challenge for guests who prefer to have these amenities available.

It's important to note that these are just recurring complaints and not all properties in Padua will have these issues. It's always a good idea to read reviews carefully and ask the host about any specific concerns you may have before booking an Airbnb property.

D.2. Examples of user specific questions

QUESTION: We are a group of 6 people that are going to stay in Padua for a weekend. We would like to stay in an entire apartment, with two rooms and two bathrooms. We trust only superhosts because they are more relialable. What accommodation would you recommend?

Naive retrieval approach

MODEL ANSWER: Based on the provided context, I would recommend ”Elegant design apartment in Padua” or ”Patavium Apartment”. Both of these accommodations meet your requirements of an entire apartment with two rooms and two bathrooms, and they are hosted by superhosts.

* ”Elegant design apartment in Padua” is a modern and spacious apartment located in the city center, within walking distance to all major attractions. It has two bedrooms, two bathrooms, and a fully equipped kitchen. The host is very responsive and helpful.

* ”Patavium Apartment” is a cozy and well-equipped apartment located in the heart of Padua, close to the main attractions and public transportation. It has one bedroom, one bathroom, and a fully equipped kitchen. The host is a superhost and has received excellent reviews from previous guests.

Both of these accommodations are excellent options for your group of six, and I recommend checking their availability and booking them as soon as possible to ensure a smooth and enjoyable stay in Padua.

SOME RETRIEVAL DOCUMENTS:

Document n. 6:

Content: Absolutely recommended apartment for staying in Padua.

Metadata: ’allowsChildren’: True, ’allowsEvents’: False, ’allowsInfant’: True, ’allowsPets’: False, ’allowsSmoking’: False, ’bathroomLabel’: ’1 bathroom’, ’bedroomLabel’: ’1 bedroom’, ’city’: ’Padua, Veneto, Italy’, ’isHostedBySuperhost’: True, ’name’: ’Elegant design apartment in Padua’, ’personCapacity’: 4, ’roomType’: ’Entire accommodation’, ’url’: ’https://www.airbnb.com/rooms/966575805894807004’

Document n. 4:

Content: Perfect accommodation, perfect location near the center and the polyclinic, super bed.¡br/¿You could improve the supply of pans in the kitchen and some furniture, but in general the apartment has everything you need to enjoy a few days in Padua.

Metadata: ’allowsChildren’: True, ’allowsEvents’:

False, 'allowsInfant': True, 'allowsPets': False, 'allowsSmoking': False, 'bathroomLabel': '1 bathroom', 'bedroomLabel': '1 bedroom', 'city': 'Padua, Veneto, Italy', 'isHostedBySuperhost': True, 'name': 'Patavium Apartment', 'personCapacity': 4, 'roomType': 'Entire accommodation', 'url': 'https://www.airbnb.com/rooms/14117139'

Self-query retrieval approach

MODEL ANSWER: Based on the retrieved context, I would recommend "Dolce vita Padova centro" as it is a fully equipped apartment with two bedrooms, two bathrooms, and a spacious living area. It is centrally located, close to restaurants, shops, and a tram station, making it an ideal choice for a group of six. Additionally, the host is a superhost, ensuring a reliable and high-quality stay.

Another option could be "Foresteria Conti, sulle tracce di Shakespeare" which is also a fully equipped apartment with two bedrooms and two bathrooms. It is located in a quiet corner of the city, but within walking distance to all the attractions. The host is also a superhost, known for their excellent hospitality and cleanliness.

Both of these accommodations have received excellent reviews and are well-equipped to meet your needs. However, if you have any specific preferences or requirements, please let me know and I can provide more tailored recommendations.

SOME RETRIEVAL DOCUMENTS:

Document n. 1:

Content: Conveniently and centrally located to all the best Padua has to offer. Lorenzo was a very gracious host. The apartment is spacious and has everything you need. Two bathrooms and laundry were an added bonus for our lengthier stay. Grazie mille!

Metadata: 'allowsChildren': True, 'allowsEvents': False, 'allowsInfant': True, 'allowsPets': False, 'allowsSmoking': False, 'bathroomLabel': '2 bathrooms', 'bedroomLabel': '2 bedrooms', 'city': 'Padua, Veneto, Italy', 'isHostedBySuperhost': True, 'name': 'Dolce vita Padova centro', 'personCapacity': 6, 'roomType': 'Entire accommodation', 'url': 'https://www.airbnb.com/rooms/43307359'

Document n. 2:

Content: Beyond fabulous. The apartment is spacious, light and airy, the kitchen well equipped and the bathrooms fully modernized. The location, tucked away in a quiet corner of the city but within easy walking distance to all of the attractions is perfect. If ever planning another trip to Padua, I would arrange it according to the availability of this wonderful accommodation.

Metadata: 'allowsChildren': False, 'allowsEvents': False, 'allowsInfant': False, 'allowsPets': False, 'allowsSmoking': False, 'bathroomLabel': '2 bathrooms', 'bedroomLabel': '2 bedrooms', 'city': 'Padua, Veneto, Italy', 'isHostedBySuperhost': True, 'name': 'Foresteria Conti, sulle tracce di Shakespeare', 'personCapacity': 6, 'roomType': 'Entire accommodation', 'url': 'https://www.airbnb.com/rooms/43275545'

E. Task-time distribution

- Literature overview: 30 h
- Code: 72 h
- Report: 40 h