Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Mats Finsås
Joachim Maksim

Optimizing RAG Systems for Technical Support with LLM-based Relevance Feedback and Multi-Agent Patterns

Master's thesis in Informatics and Computer Science
Supervisor: Jon Espen Ingvaldsen
June 2024

NTNU
Norwegian University of
Science and Technology

Mats Finsås
Joachim Maksim

Optimizing RAG Systems for Technical Support with LLM-based Relevance Feedback and Multi-Agent Patterns

Master's thesis in Informatics and Computer Science
Supervisor: Jon Espen Ingvaldsen
June 2024

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

This thesis explores the potential of large language models (LLMs) to enhance Retrieval-Augmented Generation (RAG) systems for complex technical product support. Two key areas are investigated: using LLMs for relevance feedback in information retrieval, and evaluating different multi-agent LLM communication patterns for final response generation.

For relevance feedback, experiments were conducted using various LLMs (GPT-4o, GPT-3.5-turbo, Llama3) and prompting techniques on standard information retrieval datasets. Results show LLMs can effectively mimic human relevance judgments, consistently outperforming traditional pseudo-relevance feedback methods. Larger models yielded the best performance, though the smaller Llama3 model achieved comparable results with appropriate prompting.

To evaluate multi-agent LLM patterns, experiments compared Tool-use, Chain-of-Thought, Reflection, and Planning communication patterns on a modified variant of the BioASQ-QA dataset. More complex patterns like Reflection and Planning demonstrated 55% higher correctness compared to simpler patterns, at the cost of significantly increased resource usage.

The findings suggest LLMs can substantially improve RAG systems for technical support, particularly when response quality outweighs speed. However, there are important trade-offs between performance and efficiency to consider when implementing these approaches. This research contributes to knowledge on practical applications of LLMs in information retrieval and multi-agent systems.

Table of Contents

List of Figures vi

List of Tables viii

1 Introduction 1

1.1 Background and Motivation 1

1.2 Research Objective and Questions 3

1.3 Thesis Outline 4

1.4 Results 5

1.5 Research Contribution 5

2 Background 6

2.1 The Information Retrieval (IR) Problem 6

2.2 Distance and Similarity Metrics 7

2.2.1 Minkowski Distance 7

2.2.2 Cosine Similarity 8

2.3 Proximity Search Algorithms 8

2.3.1 k Nearest Neighbors (kNN) 8

2.3.2 Approximate Nearest Neighbors (ANN) 9

2.4 Methods for Text Representation 10

2.4.1 Bag of Words (BOW) 10

2.4.2 Term Frequency - Inverse Document Frequency (TF-IDF) 11

2.4.3 GloVe 11

2.5 Query Expansion 12

2.5.1 Global Methods for Query Expansion 12
2.5.2 Relevance Feedback 13
2.5.3 Feedback Strategies for Relevance Feedback 14
2.5.4 The Rocchio Algorithm 14
2.5.5 Kullback-Leibler (KL) Query Expansion 15
2.5.6 Bose-Einstein 1 (Bo1) Query Expansion 16
2.5.7 Relevance-based Language Model (RM3) 17
2.5.8 Evaluation of Relevance Feedback Strategies 18
2.6 Nuances of Relevance 18
2.6.1 Defining Relevance 18
2.6.2 Maximal Marginal Relevance (MMR) 19
2.6.3 Working with Non-Binary Relevance Assessments 19
2.7 IR Evaluation Metrics 20
2.7.1 Precision 20
2.7.2 Recall 20
2.7.3 F-measure 20
2.7.4 Precision-recall Curves 21
2.7.5 Mean Average Precision (MAP) 21
2.7.6 Precision at k (P@k) 22
2.7.7 Normalized Discounted Cumulative Gain at k (NDCG@k) 23
2.7.8 Mean Reciprocal Rank (MRR) 24
2.8 Large Language Models 24
2.8.1 Prompting Techniques 25

2.8.2 Utilizing Custom Data in LLMs 26

2.9 Text Similarity Evaluation Metrics 27

2.9.1 Recall-Oriented Understudy for Gisting Evaluation (ROUGE) 28

2.9.2 Metric for Evaluation of Translation with Explicit ORdering (METEOR) 29

3 Related work **30**

3.1 Evaluation of IR Systems Employing Relevance Feedback 30

3.2 Query Expansion Using LLMs 30

3.3 Heuristics for Adapting Coefficients in Relevance Feedback 31

3.4 Mitigating Hallucination in LLMs 33

3.5 LLM-based Multi-agent Design Patterns 34

3.6 Evaluation of LLM-based Systems 39

3.6.1 Traditional Metrics 40

3.6.2 Benchmarks for LLM Evaluation 40

3.6.3 LLM-as-a-Judge 40

3.6.4 Evaluating Multi-agent LLM systems 41

3.6.5 Evaluating Retrieval-Augmented Generation LLM Systems 42

4 Methodology **42**

4.1 Exploratory Study 43

4.1.1 Data Gathering Module 44

4.1.2 Embedding and Storage 45

4.1.3 Search 47

4.1.4 Response Generation 47

4.1.5 Findings from Exploratory Study 48

4.2 Factorial Design Experiment for RQ1 - LLMs for Query Expansion 49
4.2.1 Technology 50
4.2.2 Planned Experiments 51
4.2.3 Baselines 51
4.2.4 Evaluation Metrics 52
4.2.5 Datasets 52
4.2.6 Experimental Setup 54
4.2.7 Evaluation 56
4.3 Single-Factor Experiment for RQ2 - LLM Communication Patterns 56
4.3.1 Technology 56
4.3.2 Experimental Setup 57
4.3.3 Patterns 58
4.3.4 Baselines 63
4.3.5 Evaluation Metrics 63

5 Results **64**
5.1 RQ1: Evaluating LLMs for Query Expansion 64
5.2 RQ2: Evaluating LLM Communication Patterns 68
5.2.1 Performance Metrics 68
5.2.2 Efficiency Metrics 70

6 Discussion **72**
6.1 RQ1: Relevance Feedback with LLMs 72
6.1.1 Outliers 72
6.1.2 Observation 1: Models are More Critical Towards VASWANI Documents . . . 73

6.1.3 Observation 2: LLMs are More Critical with Longer Outputs 73

6.1.4 Observation 3: Smaller Models Not Maximizing the Potential of Chain-of-Thought 74

6.1.5 Observation 4: Subjectivity in Human Relevance Judgments 75

6.1.6 Answering Research Question 1 76

6.2 RQ2: LLM Communication Patterns 78

6.2.1 Analysis of Performance Metrics 78

6.2.2 Analysis of Efficiency Metrics 81

6.2.3 Performance vs Efficiency Analysis 82

6.2.4 Suitability of Communication Patterns for Complex Technical Product Support 85

6.2.5 Answering Research Question 2 86

6.3 Thesis Relevance to UN Sustainable Development Goals (SDGs) 88

6.4 Limitations 88

7 Conclusion **91**

7.1 Future Work 91

Bibliography **93**

List of Figures

1 Illustration of the background theory relevance to Retrieval-Augmented Generation systems 6

2 Example of a typical IR scenario. A user has an information need, which they express as a query. An IR system uses the query to retrieve relevant documents from its document collection, and returns these to the user. 7

3 Example illustration of the kNN algorithm for $k = 3$. 5 documents (d1-5) and a query q are represented in vector space. The distance is calculated from q to all other points. The k (3) nearest points (here, d1, d3, and d5) are returned. 9

4 Example curve for the bo1 scoring function with $tf_{t,R} = 1$ and a variable $f_{avg}(t, C)$. $S(t)$ is lowest when $f_{avg}(t, C) = tf_{t,R}$ and rises as $f_{avg}(t, C)$ diverges from $tf_{t,R}$. . . 17

5 Example of a plotted precision/recall curve, retrieved from [6, Chapter 8.4]. The blue curve is created by plotting precision/recall values for each subsequent document in the ranked result list. The thinner red curve is an interpolation of the precision/recall-curve, showing the highest precision-value for each recall-value. . . 22

6 Overview of Retrieval Augmented Generation (RAG) 26

7 Query expansion method proposed by Jagerman et al. [45]. An LLM first answers the original query. The response is included in the query that is used by the retrieval system. 31

8 Tool-use illustrated by LangGraph. Retrieved from Meta AI [53] 35

9 Reflection illustrated by LangGraph. Retrieved from LangGraph [56] 36

10 Planning illustrated by LangGraph. Retrieved from LangGraph [58] 38

11 Overview of the RAG prototype developed in the exploratory study. The data gathering module scraped KSSO's and Atlassian's documentation, and extracted the core contents. An embedding script created 3 embeddings using different embedding models, and stored these together with a category tag and the original text in a PostgreSQL database. A support ticket was used as a query in the search module, which embedded it and fetched relevant KSSO documentation, as well as related Atlassian Community posts. Documents were sent along with the query to an LLM, which generated a final response. 44

12 Distribution of document token counts, measured with the tiktoken Python library. The x-axis represents the number of documents, while the y-axis is the number of tokens in a document on a logarithmic scale. The distribution shows that few documents exceed 16'000 tokens, and the vast majority is under 512 tokens. A cluster of concise documents, each under 100 tokens in length, is visible. 46

13 Tool-use and Chain-of-Thought pattern implementation 59

14 Reflection pattern implementation 61

15 Planning pattern implementation 63

16 Percentwise difference (from baseline PRF algorithm) in NDCG@k for different values of k with different language models and prompting methods on the MSMARCO dataset. 66

17 Percentwise difference (from baseline PRF algorithm) in NDCG@k for different values of k with different language models and prompting methods on the VASWANI dataset. 66

18 Average number of output tokens generated by language models on different datasets, categorized by prompt method. Standard deviation is shown with black error bars. 66

19 Fraction of documents deemed 'relevant' by each model for the MSMARCO and VASWANI datasets, categorized by prompting method. 67

20 Fraction of correct relevance judgments of each model for the MSMARCO and VASWANI datasets, categorized by prompting method. 67

21 Violin plot of performance scores by pattern 68

22 Average performance scores by pattern 69

23 Average correctness (%) by pattern 69

24 Average time spent and API calls by pattern 70

25 Average number of input and output tokens by pattern 71

26 Violin plot of time spent and API calls by pattern 71

27 Violin plot of input tokens and output tokens by pattern 71

28 Correctness and performance metrics 80

List of Tables

1 BOW representation of two documents, A and B. 10

2 Co-occurrence probabilities for target words ice and $steam$, with selected context words k. Retrieved from Pennington et al. [5] 11

3 Mean reciprocal rank (MRR) and percentwise difference in MRR from baseline (% Difference) for llama3-instruct (llama3) with 4-bit quantization, gpt-3.5-turbo-0125 (gpt-3.5-turbo) and gpt-4o-2024-05-13 (gpt-4o) on the VASWANI and MSMARCO datasets. Best relative result for each prompt method is highlighted with bold text for both datasets. Best result across all prompts is highlighted with an underline for both datasets. 65

4 Performance metrics by pattern 68

5 Efficiency metrics by pattern 70

6 Relative performance metrics by pattern 82

7 Relative efficiency metrics by pattern 82

1 Introduction

Since late 2022, Large Language Models (LLMs) have seen an explosion in popularity, as a new powerful tool that can improve various aspects of information processing and human-computer interaction. These models, trained on enormous amounts of textual data, show remarkable capabilities in understanding and generating human-like text. With new and more powerful LLMs being released as often as every week, both commercially and as open-source projects, businesses and organizations have growing interest in exploring their potential to assist, enhance or automate tasks in various domains.

One particular application of LLMs that can interest a broad range of organizations is querying their own internal knowledge bases in natural language, without being constrained to the query language of an Information Retrieval (IR) system. A popular method of querying organization or domain-specific data is called Retrieval-Augmented Generation (RAG). The method involves using an IR system to retrieve relevant pieces of information from a knowledge base, and providing them to an LLM in an instruction prompt to generate contextually accurate responses. This approach, often referred to as "grounding" an LLM, has shown potential in scenarios where access to up-to-date or domain-specific information is vital.

In this thesis, we examine Retrieval-Augmented Generation (RAG) by dissecting it into two key components: information retrieval, and response generation. In each of these areas, we explore ways to enhance traditional techniques. In doing this, we seek to expand the capabilities of RAG systems, improving their ability to access and utilize domain-specific knowledge and up-to-date information effectively.

The following sections will provide a detailed background and motivation for this research, and outline the specific research questions we will answer. The introduction concludes with an outline of the thesis, a summary of our results, and a description of our overall research contributions.

1.1 Background and Motivation

This thesis is motivated by findings from the specialization project conducted in the autumn of 2023. In the specialization project, we cooperated with Kantega Single-Sign-On (KSSO), and explored methods to enhance the efficiency of their customer support operations. KSSO is a software company based in Norway, that develops security plugins for Atlassian products such as Jira, Confluence, and Bamboo [1]. Our interviews with KSSO staff revealed that developers might spend as much as half their weekly work hours on customer support activities in extreme scenarios. Despite high customer satisfaction, this significant investment of time in resolving support tickets distracted technical staff from further developing new features for the company's core product, resulting in a long backlog of feature requests. KSSO staff hypothesised that a proportion of support tickets could likely be resolved automatically.

Motivated by recent advancements in LLMs, an idea emerged of a system that would automatically generate responses to support tickets. This system would allow KSSO employees to simply

review and approve these AI-generated responses before sending them to customers, potentially significantly reducing the time devoted to customer support tasks. The system would not directly communicate with customers due to the privacy-related risks involved. Given KSSO's relatively low volume of technical support tickets, the proposed system would not be under time pressure. Instead, its primary focus would be on producing higher quality responses. This approach sets it apart from typical chatbot applications, prioritizing accuracy and thoroughness over rapid response times.

A literature review was conducted in the specialization project, to learn about systems for automatic support ticket resolution. Through the literature review, retrieval-augmented generation (RAG) emerged as the most promising solution. This brings us to the present Master's thesis.

Due to the complex nature of KSSO's technical support, we expected that a traditional RAG application would fail to consistently resolve the company's support tickets, and that traditional RAG techniques would need to be extended. Due to little research available regarding the utilization of RAG in the context of technical customer support, we were motivated to begin our Master's thesis work by conducting an exploratory study to better understand the challenges that would arise from developing a RAG system for support ticket resolution.

Our initial exploratory study involved developing a prototype RAG application for support ticket resolution with KSSO's knowledge base and technical support tickets. A thorough description of this study is provided in the methodology section of this thesis, Section 4.1.

In our prototype, a KSSO support ticket would act as input to the RAG system, which would then attempt to generate an accurate response. Traditional RAG systems utilize vector search to retrieve relevant information for a query. This involved converting all documents from KSSO's knowledge base into vectors. The support ticket would be converted into the same vector representation. Then, to find documents relevant to the support ticket, a vector similarity search was conducted, where pairs of query-document vectors were compared, and documents whose vectors were most similar to the query were retrieved. The documents were provided to an LLM along with the original support ticket. Finally, the LLM would generate a response to the support ticket.

While testing, it was found that a simple vector similarity search, typically used in RAG applications, was insufficient for consistently retrieving relevant documents from a large and homogeneous dataset. This was because when the dataset is homogeneous, there is a significant similarity between relevant and non-relevant documents, making it more difficult to differentiate between them consistently. To us, as human evaluators, it was clear which documents in the collection were relevant for answering the query, regardless of their similarity. This motivated us to look into relevance feedback to improve search results, a technique in which a user gives feedback on document relevance to an IR system, which subsequently incorporates the feedback into a new query, and retrieves an updated set of documents. Believing that relevance feedback could mitigate our observed issue, and operating in a context without time constraints, where response quality is most important, we were motivated to explore the use of LLMs in relevance feedback.

Moreover, we observed that even when a proportion of the retrieved documents were relevant, a single language model often failed to filter out the remaining non-relevant information and instead

attempted to use all available documents, regardless of their relevance, in generating a response. This motivated us to look into ways to improve reasoning within the current space of LLMs, where one such method was to incorporate multiple LLM agents, combined with a RAG system, with the ultimate goal of improving the system's response generation capabilities.

This background sets the stage for the present Master's thesis, where we aim to further refine and optimize the use of RAG in customer support operations, addressing the identified challenges and exploring techniques to enhance the efficiency and accuracy of RAG for automated support ticket resolution.

1.2 Research Objective and Questions

The primary objective of this research is to enhance the effectiveness and efficiency of Retrieval-Augmented Generation (RAG) systems in the context of automated customer support. This objective is motivated by the challenges identified in our initial exploratory study, specifically the limitations of simple vector similarity search in retrieving relevant documents from homogeneous datasets, and the difficulties faced by single language models in effectively utilizing retrieved information.

The background motivated us to answer the following research questions in this Master's thesis:

Research Question 1 (RQ1) To what extent is an LLM able to mimic a human in a relevance feedback loop, and how does it affect second-pass retrieval results in an information retrieval (IR) system?

An information retrieval system is a system that, given a query and a document collection, aims to retrieve documents that are most relevant to the query. User relevance feedback is a technique used to improve the performance of such systems by having a user assess the relevance of each of the documents in the initial result set, adjusting the query according to the relevance feedback, and retrieving a new result set. Traditionally, a human user is asked to make relevance judgments. Techniques like pseudo-relevance feedback do not rely on users but instead, make a naive assumption that the top k retrieved documents are relevant. This question aims to explore whether LLMs can take the place of a human in relevance feedback to improve the performance of IR systems.

This research question will be addressed by answering the following sub-questions:

RQ 1.1 How closely aligned are the relevance judgments of human evaluators and LLMs?

RQ 1.2 Which factors influence the quality and efficiency of relevance judgments made by LLMs?

RQ 1.3 How does using LLMs for relevance feedback compare to traditional Pseudo-Relevance Feedback (PRF)?

Research Question 2 (RQ2) How do different multi-agent LLM conversation patterns compare in a RAG system designed to resolve technical support tickets?

Multi-agent conversation patterns are structured approaches to organizing interactions between multiple AI agents in a conversational system. The goal of this research question is to identify the most suitable conversational pattern that helps LLMs solve complex technical product support tickets. We will consider factors such as response quality, efficiency, consistency, and alignment with the specific requirements of technical support identified in our exploratory study.

Research question 2 will be addressed by answering the following sub-questions:

- RQ 2.1 How do the performance and efficiency metrics of different LLM communication patterns compare in the context of technical support ticket resolution?
- RQ 2.2 What are the strengths and weaknesses of each communication pattern in terms of response quality, consistency, and resource utilization?
- RQ 2.3 Which LLM communication pattern is most suitable for complex technical product support, considering the specific requirements of this domain?

1.3 Thesis Outline

This thesis is structured as follows:

Section 2 - Background Theory introduces the fundamental theoretical framework relevant to our research questions.

Section 3 - Related Work builds upon this foundation, presenting more recent research that extends the established theory.

Section 4 - Methodology details our research design. It begins with a description of our exploratory study, which informed the development of our research questions. We then outline the controlled experiments conducted to address these questions, providing justification for our choice of technology, variables, datasets, and evaluation metrics. This section also offers a comprehensive account of our experimental procedures.

Section 5 - Results presents our experimental findings.

Section 6 - Discussion analyzes these results in the context of our research questions, drawing insights and implications from our findings.

Section 7 - Conclusion summarizes our key findings and contributions, and proposes potential avenues for future research in this area.

1.4 Results

Our research yielded significant findings for both research questions. Regarding RQ1, we found that LLMs can effectively mimic human relevance feedback in information retrieval systems, with performance varying based on model size and prompting technique. Larger models like GPT-4o achieved consistently high performance, but the small Llama3-instruct model became highly competitive with the chain-of-thought prompting technique. Generally, LLM-based relevance feedback consistently outperformed traditional pseudo-relevance feedback, demonstrating its potential to enhance retrieval accuracy in RAG systems.

For RQ2, our evaluation of different LLM communication patterns revealed that more complex patterns, such as Reflection and Planning, achieved higher performance in terms of response quality and accuracy. These patterns showed a 55% increase in correctness compared to simpler, zero-shot (Tool-use) patterns. However, this improved performance came at the cost of significantly higher resource utilization, with Reflection and Planning patterns using 5-6 times more resources than simpler patterns. The Planning pattern demonstrated the highest consistency in performance metrics, suggesting reliable production of high-quality responses across various query types.

Overall, our results indicate that while LLMs can significantly enhance the performance of RAG systems in technical support scenarios, there are important trade-offs between response quality and resource efficiency that must be considered when implementing these systems in practice.

1.5 Research Contribution

This master's thesis makes several research contributions to the application and optimization of Retrieval-Augmented Generation (RAG) systems, especially within the field of complex technical product support where time is less of a constraint.

The thesis provides a comprehensive evaluation of several popular LLMs' capabilities in mimicking human relevance feedback for query expansion in information retrieval systems. The thesis offers insights on the potential of LLMs to enhance the accuracy of document retrieval in complex technical product support scenarios, based on experiments on known IR datasets.

The thesis also presents a comparison of different multi-agent LLM communication pattern within a pseudo RAG experimental setting. This investigation highlights some strengths and weaknesses of various patterns, and show that multiple iterations over a given task, either by self-refinement or decomposing into subtasks, can significantly increase reasoning capabilities and accuracy of the given response when provided with context of varying relevance.

This thesis also explores the potential of smaller and more resource-efficient LLMs competing with larger commercial models for specific tasks, contributing to the ongoing discussion on sustainable AI development.

2 Background

This section describes the theoretical framework upon which the research in this thesis is based. The theory presented here is well established, and is the basis of modern research in the fields of information retrieval and artificial intelligence. Our Background section focuses on the field of computer science, and therefore assumes that the reader has a basic understanding of concepts from linear algebra and calculus.

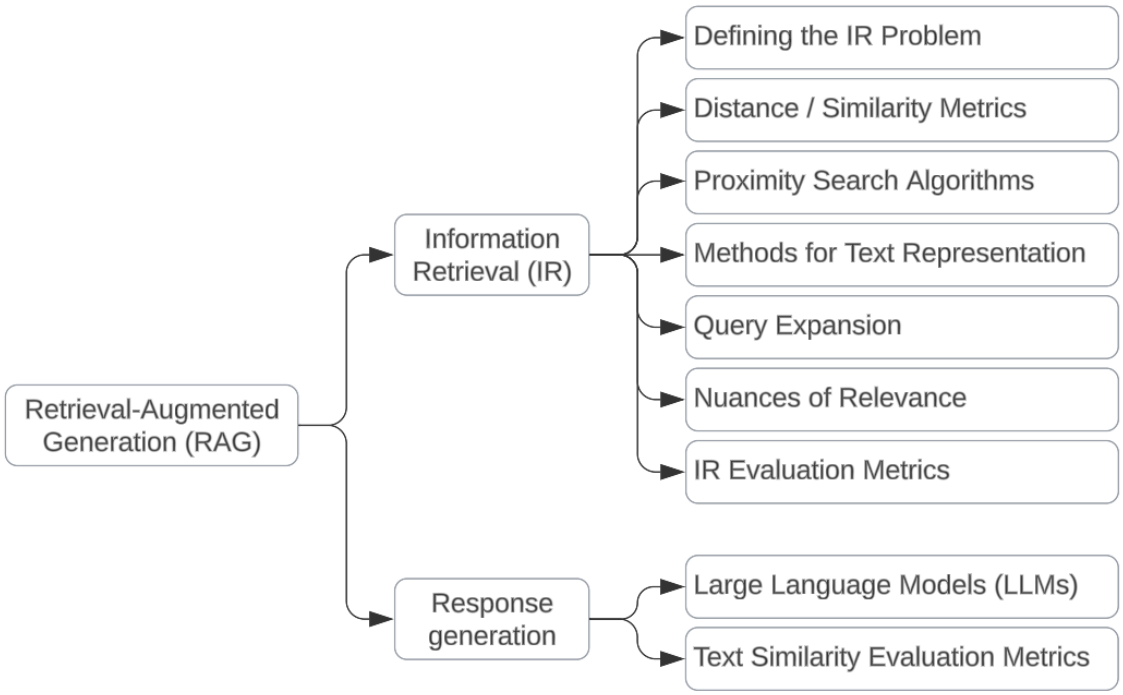

Figure 1: Illustration of the background theory relevance to Retrieval-Augmented Generation systems

Figure 1 illustrates the relevance of each background subsection to Retrieval-Augmented Generation (RAG) systems. We will discuss more recent and closely related work relevant to our research in Section 3.

2.1 The Information Retrieval (IR) Problem

Information retrieval addresses the challenge of obtaining relevant information from a vast collection of documents based on a user's expressed need. An IR scenario typically involves at least the following key components: A user with an information need, a query, and an IR system with a document collection. These components and their relationships are illustrated in Figure 2.

A user is an individual who seeks information. The user is said to have an information need, which denotes the specific information that the user seeks. Due to constraints in retrieval systems, the user must express this information need in the form of a query. The query is a representation of the user's information need, expressed in a syntax supported by the IR system, such as using

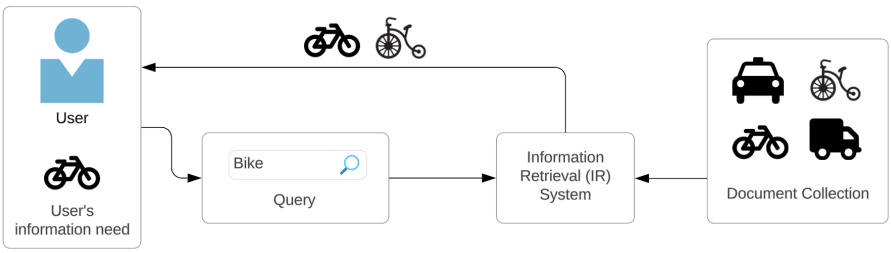

Figure 2: Example of a typical IR scenario. A user has an information need, which they express as a query. An IR system uses the query to retrieve relevant documents from its document collection, and returns these to the user.

keywords or natural language. The IR system uses the submitted query to retrieve documents that are the best match for the user's query. Documents can be any form of multimedia content with different representations. Documents that are deemed relevant to the query by the IR system are returned to the user in a result set R which can contain many documents, some of which might not exactly match the user's information need.

Various IR models have been developed, each with unique approaches to representing and matching queries and documents. An example of this is the vector space model which represents both documents and queries as vectors in multi-dimensional space. There exist various methods of measuring the similarity between a document and a query, one of which is the cosine similarity described in Section 2.2.2.

2.2 Distance and Similarity Metrics

This section describes various metrics used to measure distance and similarity between points in vector space. The Minkowski distance metric, including its special cases of the Euclidean and Manhattan distances, is described in Section 2.2.1. Section 2.2.2 describes the cosine similarity metric.

2.2.1 Minkowski Distance

The Minkowski distance metric [2, Chapter 18.8.1] measures the distance between two arbitrary points in vector space. Let a and b be two points, each with n coordinates, $a = (a_0, ..., a_n)$ and $b = (b_0, ..., b_n)$. The Minkowski distance, also known as the L^p norm for points a and b is defined as

$$L^p(a, b) = (\sum_i |a_i - b_i|^p)^{1/p} \tag{1}$$

Setting $p = 1$ gives the Manhattan distance:

$$L^1(a, b) = \sum_i |a_i - b_i| \tag{2}$$

Setting $p = 2$ gives the Euclidean distance:

$$L^2(a, b) = \sqrt{\sum_i |a_i - b_i|^2} \tag{3}$$

2.2.2 Cosine Similarity

Another widely used similarity metric is the cosine similarity, which measures the cosine of the angle between two non-zero vectors a and b. The cosine similarity is defined as:

$$sim(a, b) = \frac{a \cdot b}{|a||b|} \tag{4}$$

where $a \cdot b$ is the dot product of the vectors a and b, and $|a|$ and $|b|$ are the magnitudes (or Euclidean norms) of the vectors a and b, respectively. Specifically,

$$a \cdot b = \sum_i a_i b_i$$

$$|a| = \sqrt{\sum_i a_i^2}$$

$$|b| = \sqrt{\sum_i b_i^2}$$

Cosine similarity ranges from -1 to 1, where 1 indicates that the vectors are identical, 0 indicates orthogonality, and -1 indicates that the vectors face in opposite directions.

2.3 Proximity Search Algorithms

Assume a set S of points in a d-dimensional vector space. Given a point q, proximity search algorithms aim to find the set R of data points that are in closest proximity to q. This section describes two popular proximity search algorithms that are relevant to this thesis: k nearest neighbors (kNN) in Section 2.3.1 and its more computationally efficient variant, approximate nearest neighbors (ANN) in Section 2.3.2.

2.3.1 k Nearest Neighbors (kNN)

k Nearest Neighbors is a simple proximity search algorithm. An integer $k \in \mathbb{N}$ defines how many points closest to q should be retrieved by the algorithm. Using an arbitrary distance metric such as one the Minkowski norms described in Section 2.2.1, the algorithm calculates the distance between q and all other points. The distances are sorted, and k points with the smallest distances (the k nearest neighbors of q) are returned.

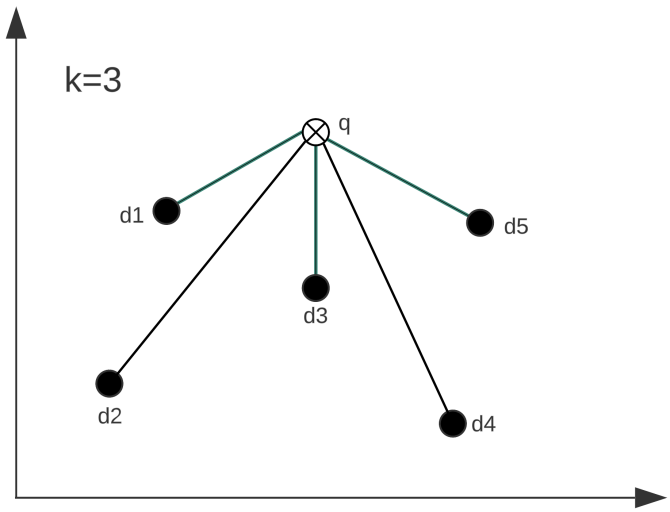

Figure 3: Example illustration of the kNN algorithm for $k = 3$. 5 documents (d1-5) and a query q are represented in vector space. The distance is calculated from q to all other points. The k (3) nearest points (here, d1, d3, and d5) are returned.

The algorithm's time complexity for a single data point is $O(nd)$, where n is the number of candidate points and d is the number of dimensions in the vector space.

2.3.2 Approximate Nearest Neighbors (ANN)

Naive KNN which calculates the distance to all documents is computationally expensive, and does not scale well in high-dimensional vector spaces with many points. Approximate Nearest Neighbors (ANN) is a group of algorithms designed to speed up the search for nearest neighbors in large and high-dimensional datasets by providing approximate results that are close to the true nearest neighbors. These algorithms trade off some accuracy for significantly improved query performance and scalability. The idea is that a small error in the distance calculation can lead to a significant reduction in computation time.

A variant of ANN uses Locality-Sensitive Hashing. The method utilizes a hash function $g(x)$ with the property that the probability of two points a and b being assigned the same hash value depends on their distance. If the points are close together, they are more likely to have the same hash value. All points in S are hashed using multiple hash functions with this property, each with a corresponding hash table. The data point q is hashed with the same functions. Each hash function ends up in some hash bucket, which contains a set of candidate points that are likely to be in close proximity to q. Finally, the distance is calculated between q and all candidate points from within the same hash buckets, and the k nearest candidate points are returned. [2, Chapter 18.8.3]

LSH requires a training phase where all training points are hashed. Depending on the number of hash functions, there is a non-0 probability that q never ends up in the same hash bucket as its actual nearest neighbor. However, the method only requires calculating the distance between a small subset of the points in the dataset, giving a significant improvement over kNN for high-dimensional vector spaces with many data points.

2.4 Methods for Text Representation

The vector space model relies on vector representations of queries and documents. Commonly, queries and documents contain natural text. Representing natural text as a vector is referred to as creating an embedding, and comes with various advantages. Firstly, embedding text increases storage efficiency, because vectors are compact representations of the original text data. Secondly, vectors are faster to perform calculations with than natural text, as they enable efficient mathematical operations that, in the context of IR, are needed to quickly determine the similarity between queries and documents.

There exist multiple methods for creating embeddings. Sparse embeddings are text representations with *sparse* vectors, vectors in which only a small number of dimensions have non-zero values. Sparse embeddings include the bag-of-words (BOW) model and TF-IDF, described in Section 2.4.1 and Section 2.4.2, respectively. Storing millions of sparse embeddings is inefficient because many vector dimensions remain unused. Dense embeddings have been developed to mitigate this problem. A popular dense embedding method is GloVe, described in Section 2.4.3. Sections 2.4.1, 2.4.2, and 2.4.3 are entirely copied from our specialization project.

2.4.1 Bag of Words (BOW)

Qader et al. [3] provide an overview of the BOW method for text representation. It is one of the simplest text representation methods in NLP. A textual document is represented as the set of its words, ignoring grammar and word ordering. Given the sentence ”Dara likes to go to the cinema”, the cleartext is first tokenized as such: ”Dara”, ”likes”, ”to”, ”go”, ”to”, ”the”, ”cinema”. The BOW-representation is a dictionary of words, and the number of times each word occurred in the document: { ”Dara”: 1, ”likes”: 1, ”to”: 2, ”go”: 1, ”the”: 1”, ”cinema”: 1 }.

In practical applications, a vocabulary is generated out of all words found in the document. Each word is a dimension in the vector space. A document is represented as a vector whose value in a dimension is the number of occurrences of the corresponding word in the document. Given two documents, A: ”Dara likes to go to the cinema” and B: ”Dara likes the cinema”, the BOW representations would be as shown in Table 1.

Table 1: BOW representation of two documents, A and B.

Document Term	Dara	likes	to	go	the	cinema
A	1	1	2	1	1	1
B	1	1	0	0	1	1

Table 1 shows an example of a term-document matrix. BOW uses the raw term frequency. It is popular due to its simplicity and low computational cost. However, the method fails to capture the meaning of the original text. The representation also tends to generate high-dimensional vectors, as vocabularies tend to be large. The method can be combined with stopword- and punctuation removal, as well as lemmatization, to reduce the vocabulary size.

2.4.2 Term Frequency - Inverse Document Frequency (TF-IDF)

Instead of solely using the raw term frequency of a word in a document, TF-IDF [4] can be used to capture the importance of words in a document. The term frequency (TF) represents how often a term appears in a document. The inverse document frequency (IDF) quantifies how rare a term is across all documents in a dataset or corpus. The rationale is that words that appear frequently across many documents might not be as useful for distinguishing between documents. Thus, they should get a low IDF value.

The highest TF-IDF value is achieved when a term $t_i \in V$ from a global vocabulary V occurs many times in a document $d_j \in D$, but few times in the entire dataset D. Thus, t_i is very representative for d_j. If a query q contains t_i, d_j is likely to appear relevant and be suggested in the process of information retrieval.

The standard formula for TF-IDF is the product of TF and IDF:

$$TFIDF(t,d) = TF \cdot IDF = f_{t,d} \cdot log(\frac{N}{n_t})$$

where $f_{t,d}$ is the number of times term t appears in document d, N is the number of documents in the dataset D, and $n_{t,d}$ is the number of documents that include term t. Notice that a term that occurs in all documents has IDF-value $log(\frac{N}{N}) = log(1) = 0$.

There exist variants of both TF and IDF that can be combined. For example, it is not uncommon to log-normalize TF to assign diminishing increases to the term weight. Setting IDF to 1 for all terms results in a standard BOW-representation.

2.4.3 GloVe

GloVe utilizes the ratio of word co-occurrence probabilities to find similar words. Given two words, $i = ice$ and $j = steam$, a set of probe words is gathered from the corpus. Assume that one such probe word is $k = solid$. Let $P_{ik} = P(k|ice)$ denote the probability that k co-occurs with (occurs within some context window of) the word ice. Similarly, let $P_{jk} = P(k|steam)$ denote the probability that k co-occurs with $steam$.The ratio $P_{ik}/P_{jk} = P(k|ice)/P(k|steam)$ can show which of the words ice and $steam$ is more likely to co-occur with $k = solid$. If the ratio is a large number, then ice is more connected to $solid$. Contrarily, if the ratio approaches 0, then $steam$ is more likely to co-occur with $solid$. If the ratio is close to 1, then both words tend to co-occur with $k = solid$ equally often. Table 2 shows the co-occurrence probabilities and the probability ratio, retrieved from [5].

Table 2: Co-occurrence probabilities for target words ice and $steam$, with selected context words k. Retrieved from Pennington et al. [5]

Probability and Ratio	k = solid	k = gas	k = water	k = fashion
$P(k\|ice)$	$1.9 \cdot 10^{-4}$	$6.6 \cdot 10^{-5}$	$3.0 \cdot 10^{-3}$	$1.7 \cdot 10^{-5}$
$P(k\|steam)$	$2.2 \cdot 10^{-5}$	$7.8 \cdot 10^{-4}$	$2.2 \cdot 10^{-3}$	$1.8 \cdot 10^{-5}$
$P(k\|ice)/P(k\|steam)$	8.9	$8.5 \cdot 10^{-2}$	1.36	0.96

From Table 2, it is clear that ice is more related than $steam$ to the probe word $k = solid$. Contrarily, $steam$ is more related than ice to $k = gas$. Irrelevant noise-words such as $k = water$ and $k = fashion$ are not decisively more related to either ice or $steam$. Compared to the raw probabilities, the ratio much more clearly distinguishes relevant words such as $solid$ and gas from irrelevant words such as $water$ and $fashion$.

GloVe subsequently learns the probability ratios, and aims to optimize the word embeddings such that the dot product of pairs of words matches their probability ratio as closely as possible.

The benefit of this approach, is that words that do not tend to appear together, such as ”ice” and ”steam”, can end up with similar embeddings. The reason is that they tend to occur in similar contexts, for example in sentences related to water or other liquids.

2.5 Query Expansion

Query expansion is a popular technique for improving recall in information retrieval by adjusting the original user-defined query q_0. For example, a query such as ”thermodynamics” should match documents related to heat, hence "thermodynamics" could be expanded to include the term "heat". There exist global and local methods for query expansion. Global methods - described in Section 2.5.1 - expand or reformulate query terms independent of the results returned from q_0. Meanwhile, local methods utilize the first-pass result set R_0 to optimize the query. One of the most used and most successful approaches for local query expansion is relevance feedback [6, Chapter 9]. Section 2.5.2 introduces the concept of relevance feedback, and describes its intuitiveness and limitations. Section 2.5.3 describes various strategies of collecting relevance feedback. Sections 2.5.4 to 2.5.7 describe different algorithms for incorporating the relevance feedback into the query vector, namely the Rocchio algorithm, Kullback-Leibler (KL) query expansion, Bose-Einstein 1 (Bo1) query expansion, and the Relevance-based Language Model (RM3). Finally, Section 2.5.8 concludes the section by discussing how to evaluate relevance feedback strategies.

2.5.1 Global Methods for Query Expansion

Global query expansion methods do not rely on the first-pass result set R_0. For this reason, they can be applied to any query before executing a search. Manning et al. [6, Chapter 9] mention expansion and reformulation of queries with a thesaurus, and spelling correction as examples of global query expansion methods.

Query reformulation and expansion with a thesaurus involves utilizing a thesaurus to generate a set of reformulated queries for the user. The user is then able to select the most relevant query from the set. A common method is to look up each query term t in a thesaurus to find a new term t_{alt}, and either replace t with t_{alt}, or add t_{alt} to the query q.

Humans have manually created thesauruses for many purposes. These thesauruses can be used

in query reformulation and expansion. Alternatively, Manning et al. [6, Chapter 9] suggest that a thesaurus can be generated automatically by calculating word co-occurrence statistics in the document collection. This involves starting with a term-document matrix A, where each cell $A_{t,d}$ is a weighted count $w_{t,d}$ indicating the relative number of times term t appeared in document d. Consequently, if $C = AA^T$, then $C_{u,v}$ denotes the similarity between terms u and v. The intuition behind this is that the equation is the dot product between pairs of rows of the matrix A. The dot product indicates the similarity between the two rows. A high similarity between rows A_u and A_v of the term-document matrix A indicates that the terms u and v tend to appear a similar number of times in the same documents, and are therefore likely to be semantically similar.

Manning et al. [6, Chapter 9] mention that term ambiguity can cause misleading term co-occurrences, such as in the hypothetical case that 'Apple computer' is expanded to 'Apple red fruit computer'. Moreover, since the terms in the automatic thesaurus tend to commonly co-occur in the document collection, such query expansion may fail to yield many additional documents. False positives may also add non-relevant terms to the query, negatively impacting precision.

2.5.2 Relevance Feedback

Relevance feedback is a type of local query expansion which involves identifying which documents from the first-pass result set R_0 are relevant. The query is automatically reformulated using a query expansion algorithm which incorporates this relevance feedback, and the user receives a second-pass result set R_1. This can continue for a number of iterations.

According to Manning et al. [6, Chapter 9.1.5], one round of relevance feedback is often very useful, but each subsequent iteration of relevance feedback has diminishing returns. It is also important that enough documents are judged in each iteration. Otherwise, the process may drift away from the user's information need. Therefore, the authors recommend judging at least five documents.

Manning et al. [6, Chapter 9.1.3] describe a number of cases in which relevance feedback is unlikely to be effective. These are briefly described in the remainder of this subsection.

Misspellings in either the query or the parsed document may alter the vector embeddings such that the similarity between the query-vector and the vector representations of relevant documents is too low to be retrieved. This may be alleviated using spelling correction techniques.

Documents written in other languages than the query will not reside near the query in vector space, and are unlikely to be retrieved regardless of the number of relevance feedback iterations.

Differences in the vocabulary used in the query and document may not be captured by the embedding algorithm. This would result in a low vector similarity, making relevance feedback ineffective. For example, if the user searches for 'laptop', documents containing 'notebook computer' may not be deemed similar if the embedding algorithm fails to associate the two keywords with each other.

If relevant documents are dissimilar, they will not cluster. Some relevance feedback algorithms treat relevant documents as a single cluster, which does not work well if there are multiple clusters

of relevant documents. A solution to this could be to introduce document that link clusters of documents by combining their topics. If a linking document appears in one of the result sets, the user can mark it as relevant, and the query will be adjusted towards the second cluster of relevant documents.

2.5.3 Feedback Strategies for Relevance Feedback

There are various ways of obtaining relevance feedback. Explicit, implicit and pseudo-relevance feedback are briefly introduced in this subsection.

Explicit Relevance Feedback Explicit relevance feedback is a feedback strategy in which the user of the IR system manually specifies which documents from the first-pass result set R_0 are relevant and not. The intuitiveness is that it may be difficult to formulate an appropriate query without knowing the document set, but it is easy to judge the relevance of documents to specific information needs. [6, Chapter 9.1]

Implicit User Relevance Feedback Users are often reluctant to provide explicit feedback. Therefore, some search engines employ implicit methods for gathering relevance feedback by collecting user data. For example, the DirectHit search engine assumes that a web page p is likely to be relevant to a query q if a user clicks on p when searching q. Users decide which results to click on based on their title and short summary. This approach therefore assumes that the document summaries are representative of the main content of each article [6, Chapter 9.1.7].

Pseudo-relevance Feedback Pseudo-relevance feedback automates the relevance feedback such that the user is not bothered while searching. The approach does relevance feedback based on the assumption that the top-k ranked documents are relevant. The danger of this approach is that if the top-k documents are non-relevant, it can drift the result set towards non-relevant content, effectively giving a worse user experience [6, Chapter 9.1.6].

2.5.4 The Rocchio Algorithm

The Rocchio algorithm provides a method for incorporating relevance feedback information into the query vector $\vec{q}$. The set of retrieved documents R contains a subset of relevant documents D_r and a subset of non-relevant documents $D_{nr} = R - D_r$. The goal is to adjust the query vector $\vec{q}$ to maximize its similarity to documents in D_r while minimizing its similarity to documents in D_{nr}. In other words, we wish to maximize the difference between $sim(\vec{q}, D_r) - sim(\vec{q}, D_{nr})$. Assuming that D_r and D_{nr} are two separate clusters, the optimal query vector $\vec{q}_{opt}$ should be the difference between the two cluster centroids. The centroid of a cluster consisting of a set of documents D is

just the average of all document vectors within D:

$$centroid(D) = \frac{1}{|D|} \sum_{\vec{d_j} \in D} \vec{d_j}$$

Thus, the optimal query vector $\vec{q}_{opt}$ satisfies

$$\vec{q}_{opt} = centroid(D_r) - centroid(D_{nr}) = \frac{1}{|D_r|} \sum_{\vec{d_j} \in D_r} \vec{d_j} - \frac{1}{|D_{nr}|} \sum_{\vec{d_j} \in D_{nr}} \vec{d_j}$$

Given an initial query vector $\vec{q}_0$, a set of relevant documents D_r and a set of non-relevant documents D_{nr}, the Rocchio algorithm can be expressed as

$$\vec{q}_m = \alpha \vec{q}_0 + \beta \frac{1}{|D_r|} \sum_{\vec{d_j} \in D_r} \vec{d_j} - \gamma \frac{1}{|D_{nr}|} \sum_{\vec{d_j} \in D_{nr}} \vec{d_j} \tag{5}$$

α, β and γ are weights attached to each term, controlling the balance between trusting the judged document set or the original query. Intuitively, the new query starts from the original query $\vec{q}_0$, moves some distance towards the centroid of relevant documents, and finally moves some distance away from the centroid of non-relevant documents [6, Chapter 9.1.1]. Higher values of α emphasize the initial query, while higher values of β and γ emphasize relevance judgements. According to [6], positive feedback is shown to be more valuable than negative feedback, so most IR systems set $\gamma < \beta$. The authors of [6] claim that $\alpha = 1$, $\beta = 0.75$ and $\gamma = 0.15$ are reasonable values. Many systems only allow positive feedback, effectively setting $\gamma = 0$. Some systems only use the single non-relevant document $\vec{d_j} \in D_{nr}$ which was most similar to the original query $\vec{q}_0$, effectively setting $|D_{nr}| = 1$. This variant of the Rocchio algorithm is called *Ide dec-hi*, and tends to give consistent results [6].

2.5.5 Kullback-Leibler (KL) Query Expansion

KL query expansion is based on the Kullback-Leibler divergence between term distributions. First, a term probability distribution is calculated both for the set of pseudo-relevant documents R and for the entire corpus C. All terms in the pseudo-relevant set are treated as candidate expansion terms. When selecting terms for query expansion, the intuition is to use terms that contribute the most to the divergence between the term probability distribution in relevant documents and that of the entire corpus [7]. For example, selecting a term that is common in a pseudo-relevant document but uncommon in the general corpus for query expansion, would most efficiently shift the query towards the characteristic features of relevant documents.

Assume a set of pseudo-relevant documents R and a corpus C. For a specific term t, p_r is a probability distribution of terms in the set of pseudo-relevant documents, and p_c is the probability distribution of terms in the corpus. Thus, $p_r(t)$ and $p_c(t)$ are the probabilities of finding term t in the set of relevant documents and the corpus, respectively:

$$p_r(t) = \frac{\text{Term frequency of } t \text{ in } R}{\text{Total number of terms in } R} = \frac{\sum_{d \in R} tf(t,d)}{\sum_{d \in R} \sum_{t' \in d} tf(t',d)}$$

$$p_c(t) = \frac{\text{Term frequency of } t \text{ in } C}{\text{Total number of terms in } C} = \frac{\sum_{d \in C} tf(t,d)}{\sum_{d \in C} \sum_{t' \in d} tf(t',d)}$$

where $tf(t,d)$ is the term frequency of term t in a document d. The contribution of a term to the divergence between p_r and p_c can be expressed as

$$S(t) = p_r(t) \cdot log \frac{p_r(t)}{p_c(t)}$$

Terms with the highest contribution are selected for query expansion. [7]

Notice that if $p_r(t) = p_c(t)$, then $log(\frac{p_r(t)}{p_c(t)}) = 0$ and $S(t) = 0$. If $p_r(t) < p_c(t)$, then $S(t) < 0$, because it is undesirable to add terms that are less common in the set of relevant documents than in the entire corpus C. Contrarily, when $p_r(t) > p_c(t)$, then $S(t) > 0$, because terms that more commonly occur in R than in C are deemed representative of the pseudo-relevant documents, and should therefore be used for query expansion.

2.5.6 Bose-Einstein 1 (Bo1) Query Expansion

In bo1, the informativeness of a term t is measured by the divergence between its distribution in pseudo-relevant documents R and a random distribution. The score of a candidate expansion term t is expressed as

$$S(t) = tf_{t,R} \cdot log_2(\frac{1 + f_{avg}(t,C)}{f_{avg}(t,C)}) + log_2(1 + f_{avg}(t,C))$$

where $tf_{t,R}$ is the term frequency of term t in the set of pseudo-relevant documents R

$$tf_{t,R} = \sum_{d \in R} tf(t,d)$$

and $f_{avg}(t,C)$ is the average term frequency of term t in a random document from the corpus C, expressed as

$$f_{avg}(t,C) = \sum_{d \in C} tf(t,d)/N$$

where $tf(t,d)$ is the term frequency of term t in document d, and N is the number of documents in the collection.[7]

Notice that f_{avg} can never be 0, because if it were, then t would not be considered to be a candidate expansion term.

Figure 4 shows an example curve of $S(t)$ where a term t is found once in the set of pseudo-relevant documents ($tf_{t,R} = 1$). The x-axis represents the $f_{avg}(t,C)$ value. Notice that the lowest score for a term t is when $tf_{t,R} = f_{avg}(t,C)$, meaning that the pseudo-relevant documents contain term t exactly as many times as would be expected based on the corpus average. Contrarily, if $tf_{t,R} > f_{avg}(t,C)$, then t becomes a clear candidate for query expansion.

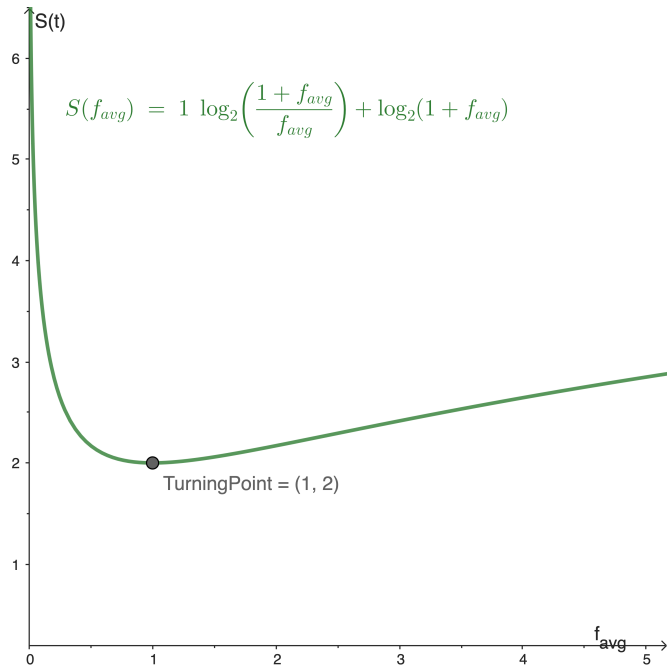

Figure 4: Example curve for the bo1 scoring function with $tf_{t,R} = 1$ and a variable $f_{avg}(t, C)$. $S(t)$ is lowest when $f_{avg}(t, C) = tf_{t,R}$ and rises as $f_{avg}(t, C)$ diverges from $tf_{t,R}$.

2.5.7 Relevance-based Language Model (RM3)

RM3 is an association-based approach for query expansion. Assume a query Q consisting of k query-terms such that $Q = q_1, q_2, ..., q_k$. In RM3, the association $S(t)$ between a term t and a query is measured by the joint probability of observing t together with the query terms, $P(t, q_1, ..., q_k)$. Each of these terms are sampled from an unknown relevance model. The relevance model consists of a finite universe $\mathcal{M}$ of term distributions, each of which derives from a relevant document. Since the relevant documents are unknown, an assumption is made that the pseudo-relevant documents are indeed relevant, and their term distributions are used instead. Under the assumption that $t, q_1, ..., q_k$ are independently and identically sampled from $M \in \mathcal{M}$,

$$\begin{aligned} S(t) &= P(t, q_1, ..., q_k) \\ &= \sum_{M \in \mathcal{M}} P(M)P(t|M) \prod_{i=1}^{k} P(q_i|M) \\ &= \frac{1}{|R|} \sum_{d \in R} \left(\frac{tf(t,d)}{|d|} \cdot \prod_{i=1}^{k} \frac{tf(q_i,d) + \mu P(q_i|C)}{|d| + \mu} \right) \end{aligned}$$

Here, $\frac{tf(t,d)}{|d|}$ is the average term frequency of t in the set of pseudo-relevant documents. $\frac{tf(q_i,d)+\mu P(q_i|C)}{|d|+\mu}$ is the average term frequency of a query term q_i, smoothed by the probability of finding t in the corpus. The smoothing is applied to avoid solely relying on term frequency. This due to the fact that term frequency can be 0, nullifying the impact of an entire pseudo-relevant document on $S(t)$.

2.5.8 Evaluation of Relevance Feedback Strategies

A possible strategy to evaluate the effectiveness of relevance feedback could be to start with an initial query q_0, retrieve an initial result set R_0, apply relevance feedback to obtain a refined search query q_1, retrieve a new result set R_1, and compare precision and recall between R_0 and R_1. According to Manning et al. [6, Chapter 9.1.5], the mean average precision for R_1 tends to be up to 50% higher than that of R_0. However, the authors argue that this comparison is unfair, because known relevant documents judged by the user are ranked higher in R_1, and fairness would demand that each result set should only include documents not seen before by the user.

A second strategy is therefore to exclude all documents deemed relevant in R_0 from the dataset before retrieving documents for R_1. However, this can lead to an unrealistically low performance, especially if there are few relevant documents in the dataset. In a worst-case scenario, all relevant documents might have appeared in R_0, leaving no relevant documents for R_1.

A third method mentioned by the authors of [6] is to use two collections in which one is used for the initial query and relevance judgments, and the second is used for comparative evaluation.

A time-based evaluation may also be used, for example by measuring the time until a user finds relevant documents with relevance feedback compared to other strategies, or counting the number of relevant documents that are found after a predetermined amount of time [6].

2.6 Nuances of Relevance

Until the present moment, this thesis has very loosely used the term ”relevance”. In reality, relevance is a complex, subjective and dynamic term. This section aims to illustrate why defining relevance is difficult. Section 2.6.1 distinguishes between the query and the user’s information need and reiterates the relevance definition used by the Text REtrieval Conference (TREC). Section 2.6.2 describes Maximal Marginal Relevance (MMR); a definition of relevance that accounts for the term’s dynamic nature. Finally, section 2.6.3 shows examples of how to work with non-binary relevance assessments.

2.6.1 Defining Relevance

Evaluation of IR systems often revolves around the notion of relevant and non-relevant documents. There exist many definitions of relevance.

Manning et al. [6, Chapter 8.1] emphasize that the relevance of a document is based on the user’s information need, not the query. Hamid [8] states that relevance defines how much the retrieved information meets the user requirements, reiterating that it is the information need that is of importance, not the query. If a user types ’python’ into a search engine, they may wish to learn where to buy a pet python. Returning a set of articles about Python programming would be relevant to the query, but not to the information need.

According to The National Institute of Standards and Technology (NIST), TREC uses the following definition of relevance: If you were writing a report on the subject of the topic and would use the information contained in the document in the report, then the document is relevant. Importantly, a document is judged relevant if any piece of it is relevant, regardless of how small the piece is in relation to the rest of the document. [9]

2.6.2 Maximal Marginal Relevance (MMR)

The relevance of a document tends to be treated as independent of the relevance of other documents in the collection. This is because documents are usually compared to the query q and not to each other when assessing their relevance. Independence of relevance implies that the users' information needs do not change as they start looking at retrieval results [6, Chapter 8.5.1]. This is unrealistic due to overlap in information between documents returned in the same query. For example, if a result set only includes duplicates, it is possible that each returned document is individually relevant to the query q, but in reality, only the first document that the user reads is relevant to the user's information need. A document is said to have high *marginal relevance* if it is both relevant to the query and contains minimal similarity to previously selected documents. The goal should therefore be to maximize the marginal relevance of documents in the result set; i.e. maximize the maximal marginal relevance (MMR), initially introduced by Carbonell and Stewart [10]:

$$MMR = \text{Arg} \max_{D_i \in R \setminus S} [\lambda \cdot sim(D_i, Q) - (1 - \lambda) \max_{D_j \in S} sim(D_i, D_j)]$$

Here, $R = IR(C, Q, \theta)$ is the ranked list of documents retrieved by an IR system given the collection of documents C, a query Q and a relevance threshold θ. S is the subset of documents in R already selected, meaning that $R \setminus S$ is the set of yet unselected documents in R. $\lambda \in [0, 1]$ is a parameter controlling the balance between rewarding document-query-similarity and penalizing document-document-similarity. When $\lambda = 1$, MMR computes the standard relevance-ranked list. When $\lambda = 0$, MMR computes a maximal diversity ranking among the documents in R [10].

2.6.3 Working with Non-Binary Relevance Assessments

Relevance assessments tend to be binary; a document is either relevant or non-relevant. In reality, the degree of relevance of a document for a specific query is continuous or at the very least categorical. There have been developed evaluations with an ordinal notion of relevance divided into 3 or 4 classes, distinguishing slightly relevant documents from highly relevant documents [6, Chapter 8.5.1]. The INEX reference collections are a good example of such evaluations.

The INEX (**IN**itiative for the **E**valuation of **X**ML retrieval) program produces reference collections containing sets of queries and relevance judgments, pairing queries with relevant XML documents. Relevance judgments are not binary, but ordinal. The relevance of documents is judged by human assessors, and is based on the definition of relevance described in Section 2.6.1. INEX 2002 defines *component coverage* and *topical relevance* as dimensions of relevance. The component coverage dimension evaluates whether the XML element retrieved is "structurally correct", i.e. to which degree this part of the full document XML tree includes the information sought by the user as its

main component. The component coverage is evaluated on four levels ranging from no coverage to exact coverage. The topical relevance also evaluated on four levels, and lastly, relevance and coverage for an XML component c are combined into a single ordinal metric $Q(c)$ ranging from 0 (not at all relevant) to 1 (very relevant). Then, the number of relevant components in a retrieved set A of components can be computed as

$$\#(\text{relevant items retrieved}) = \sum_{c \in A} Q(c)$$

Finally, standard evaluation metrics such as precision, recall and the F-measure can be applied using this modified definition of relevant items retrieved [6, Chapter 10.4].

2.7 IR Evaluation Metrics

This section describes a number of evaluation metrics commonly used to evaluate the performance of IR systems. Subsections 2.7.1-2.7.3 describe set-based measures that are computed regardless of document ordering in the result set. Subsection 2.7.4 introduces precision-recall curves, which are used to derive rank-based evaluation metrics such as mean average precision (MAP), precision at k ($P@k$), normalized discounted cumulative gain at k ($NDCG@k$) and mean reciprocal rank (MRR), described in subsections 2.7.5-2.7.8.

2.7.1 Precision

Precision is the fraction of retrieved documents that are relevant [6, Chapter 8.3].

$$Precision = \frac{\text{\#(relevant items retrieved)}}{\text{\#(retrieved items)}} \tag{6}$$

2.7.2 Recall

Recall is the fraction of relevant documents that are retrieved [6, Chapter 8.3].

$$Recall = \frac{\text{\#(relevant items retrieved)}}{\text{\#(relevant items)}} \tag{7}$$

Recall is a non-decreasing function of the number of retrieved documents. Notice that any IR system may have 100% recall if it returns all documents. Therefore, recall by itself is an incomplete measure of IR performance.

2.7.3 F-measure

Which is most important between precision and recall depends on the circumstances. In web browsing, having relevant documents at the top of the search results is valuable, even if not all

relevant documents are present. In that case, precision is more important than recall. Contrarily, professional searchers such as doctors and lawyers may prefer to find all relevant documents, even if that means having to go through a number of non-relevant documents first. For them, recall is more important. Precision and recall trade off against each other. Achieving 100% recall is possible by retrieving more documents, but that is likely to negatively affect precision. [6, Chapter 8.3]

The F-measure trades off precision versus recall. It is the weighted harmonic mean of precision and recall.

$$F = \frac{1}{\alpha \frac{1}{precision} + (1-\alpha)\frac{1}{recall}} \tag{8}$$

Here, $\alpha \in [0, 1]$ defines the weights of precision and recall. The default balanced F-measure equally weighs precision and recall, setting $\alpha = \frac{1}{2}$, changing the expression to

$$F_{\alpha=\frac{1}{2}} = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{9}$$

When evaluating an IR system where recall is to be emphasized, α could be set to a lower value. Contrarily, in an IR system where precision is important, α could be increased. [6, Chapter 8.3]

2.7.4 Precision-recall Curves

In ranked retrieval, sets of retrieved documents are given by the top-k retrieved documents, for a predefined value k. Retrieving one document at a time, precision and recall is calculated, and a point is plotted in a graph with recall on the x-axis, and precision on the y-axis. The result is the blue-colored curve in Figure 5. If a document is relevant, both precision and recall increase, leading to a jump upwards and to the right. Contrarily, if the document is non-relevant, recall remains the same, while the precision decreases. It is common to interpolate the precision-recall-curve by simply plotting the highest precision value for each recall-value. This is depicted by a thinner red curve in Figure 5. [6, Chapter 8.4]

2.7.5 Mean Average Precision (MAP)

Although viewing the precision/recall graph yields a lot of useful information, it is often preferred to boil this information down into a single numeric value. For a query q, documents are first ranked, and a top-k ranked result list R_j is obtained. For each *relevant* document $\{d_1, ..., d_{m_j}\} \in R_j$, the precision is calculated at that rank position. These precision values are subsequently averaged to obtain the Average Precision (AP):

$$AP(q_j, R_j) = \frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk}) \tag{10}$$

where R_{jk} is the set of ranked retrieval results from the top result until you get to the relevant document d_k. If no relevant documents have been retrieved in R_{jk}, $Precision(R_{jk}) = 0$. Mean

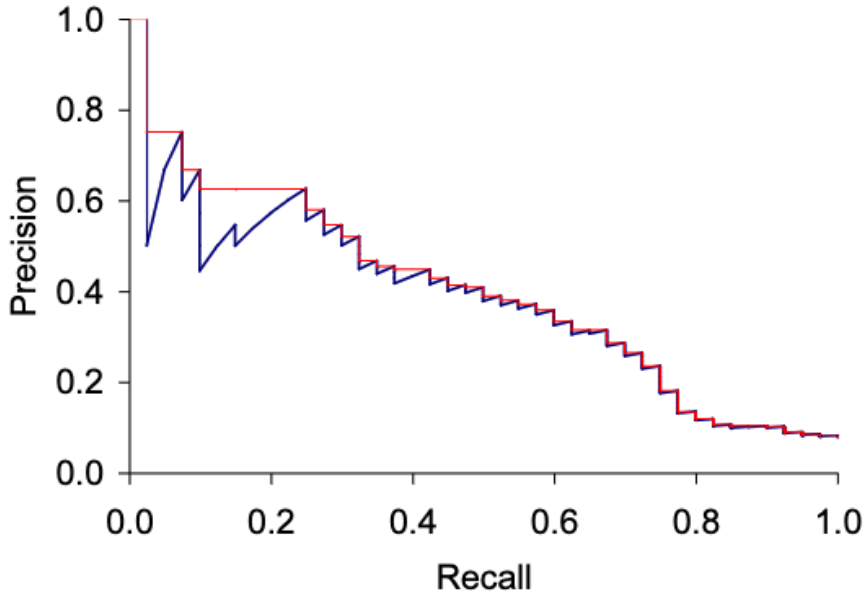

Figure 5: Example of a plotted precision/recall curve, retrieved from [6, Chapter 8.4]. The blue curve is created by plotting precision/recall values for each subsequent document in the ranked result list. The thinner red curve is an interpolation of the precision/recall-curve, showing the highest precision-value for each recall-value.

average precision is calculated by averaging AP for each query $q_j \in Q$, Q being the set of queries [6, Chapter 8.4]:

$$MAP(Q, R) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} AP(q_j, R_j) \tag{11}$$

The MAP value for a test collection is calculated as the arithmetic mean of the average precision scores for individual queries. This approach ensures that each query is given equal weight in the final MAP score, regardless of the number of relevant documents associated with each query. MAP scores for different queries can vary significantly within a single system, for example, ranging from 0.1 to 0.7. Notably, there tends to be greater consistency in MAP scores for a specific query across different systems than for different queries within the same system. Therefore, it is important to use a large and diverse set of test queries to ensure the MAP score accurately represents the system's effectiveness across various queries. [6, Chapter 8.4]

2.7.6 Precision at k (P@k)

For applications such as web search, the most important thing is the number of relevant results on the first result page. This can be evaluated by measuring precision at fixed low levels of retrieved results, such as 10 or 30 documents, referred to as "Precision at k", or $P@k$ in short. For example, $P@10$ would measure the precision for a set of 10 retrieved documents. Unfortunately, $P@k$ is rather unstable and does not average well because the total number of relevant documents for a query strongly influences it. [6, Chapter 8.4]

2.7.7 Normalized Discounted Cumulative Gain at k (NDCG@k)

The following description of NDCG is mainly based on Aggarwal [11, Chapter 7.5], but adapts the equations to measure NDCG at specific values of k, as proposed in Manning et al. [6, Chapter 8.4].

Discounted cumulative gain (DCG) measures the usefulness (gain) of a document to the user based on the document's position in the result list. The gain is discounted (reduced) logarithmically, because items lower in the ranking contribute less to the overall gain. The DCG at position k in the ranked result list for a query q is defined as

$$DCG(q,k) = \sum_{i=1}^{k} \frac{g_i}{log_2(i+1)} = \sum_{i=1}^{k} \frac{2^{relevance_{i,q}} - 1}{log_2(i+1)} \tag{12}$$

where g_i is the gain of the result at position i, typically set to $2^{relevance_{i,q}} - 1$ with $relevance_{i,q}$ being the relevance of the document at position i to a query q.

Ideal DCG (IDCG) is the DCG of the ideal ranking, which is obtained by sorting items by their relevance in descending order.

Normalized DCG (NDCG) is the ratio between DCG and IDCG, defined as

$$NDCG = \frac{DCG}{IDCG} \tag{13}$$

The ratio compares how close the actual DCG is to the DCG of an ideal ranking, and ensures that the NDCG score ranges from 0 to 1, where 1 indicates a perfect ranking where all of the most relevant documents are on the top of the result list.

NDCG can be calculated at a specific rank position k. It is then referred to as $NDCG@k$. For a set of queries Q, it is expressed as

$$NDCG(Q,k) = \frac{1}{|Q|} \sum_{q \in Q} \frac{DCG(q,k)}{IDCG(q,k)} \tag{14}$$

NDCG is designed for non-binary notions of relevance [6, Chapter 8.4], but can be used with binary relevance judgments where the relevance of each document is either 1 or 0. When calculating the DCG, $2^{relevance_{i,q}} - 1$ will be 1 for relevant documents and 0 for non-relevant documents. For the IDCG, as long as all relevant documents are in the top positions of the ranked list, their relative ordering is irrelevant. For example, if the IR system returns a document list with the following document relevances:

$$[1, 0, 0, 1, 0]$$

then the list can be adapted to calculate IDCG by moving all relevant documents to the beginning ("top") of the list:

$$[1, 1, 0, 0, 0]$$

Thus, NDCG can be calculated even with binary relevance judgments.

It is evident, based on Equation 14, that NDCG alleviates some of the issues with other precision-based metrics. Unlike $P@k$ or MAP, NDCG will not be negatively affected if a query is associated with few relevant documents, due to the normalization with IDCG; NDCG can still be high even if DCG is low, as long as DCG is close or equal to the IDCG.

2.7.8 Mean Reciprocal Rank (MRR)

The reciprocal rank (RR) for a query is the reciprocal of the rank of the first relevant document. For example, if the first relevant document is found at rank 3, RR is $\frac{1}{3}$. MRR is the average of the reciprocal ranks over a set of queries and is expressed as

$$MRR(Q) = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{S_{correct}(R_i)} \tag{15}$$

where Q is the set of queries that the evaluation is run on, R_i is the result set for the i-th query, and $S_{correct}$ is a function that returns the position of the first correct answer in the result set. [12, Chapter 4.3.2]

MRR is high when the first relevant document appears near the top. It is bounded between 0 and 1. A drawback is that only the first relevant result is considered, but MRR can still be useful for evaluating instances in which the first correct result is what matters the most, such as in question-answering systems. [12, Chapter 4.3.2]

2.8 Large Language Models

With the introduction of ChatGPT by OpenAI [13], Generative AI and particularly Large Language Models (LLMs) has exploded in popularity [14, 15, 16], and has now become one of the most popular fields within natural language processing (NLP). Today, we see companies rushing to integrate LLMs into a variation of products in our daily life [17, 18, 19], competing and continuously pushing the boundaries of what machines can understand and generate.

LLMs are trained on vast datasets comprised of text, enabling them to predict the probability distribution of words in a sequence based on the context provided by the prior words. The training process involves adjusting the model's weights to minimize the difference between its predictions and the actual next words in the sequences. [20, 21].

The architecture of LLMs has evolved significantly over time to enhance their capabilities in capturing linguistic patterns and dependencies. Early models relied on Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, which processed text sequentially [22, 23]. However, Vaswani et al. [24] introduced the Transformer architecture in 2017, marking the most recent significant breakthrough, using self-attention mechanisms to capture long-range dependencies more effectively and enable parallel processing of sequences.

Transformer-based models, such as the early BERT [25], the GPT series [26, 27, 28] and most notable other recent models, employ a pre-training and fine-tuning approach. They are first pre-trained on large corpora to learn general language representations and then fine-tuned on specific tasks, demonstrating impressive performance across a wide range of NLP tasks and applications.

The performance and capabilities of LLMs have improved dramatically in recent years, with Generative AI's explosion in popularity, now driven by increases in model size, training data, and computational resources [28]. The success story of OpenAI's ChatGPT partnered with Microsoft triggered a competitive drive among multiple global tech giants. Companies including Amazon-backed Anthropic [29, 30], Google DeepMind [31], Meta AI [32], and Apple [33, 34] are investing heavily to compete. However, LLMs still have clear limitations such as biases and inconsistencies that require careful consideration when deploying them in real-world scenarios. As LLMs continue to advance, understanding their inner workings, capabilities, and potential implications becomes increasingly important. We will investigate some recent work around limitations and their mitigation techniques in Section 3.4 and Section 3.5.

2.8.1 Prompting Techniques

Few-shot Learning Few-shot learning is a technique for in-context tuning of language models. It involves providing a number K of question-answer examples in the input prompt. The difference between few-shot learning and fine-tuning described which we will describe below, is that the weights of the model are not adjusted in few-shot learning. Although few-shot learning generally gives better results than zero- and one-shot learning ($K = 0$ and $K = 1$, respectively), pure few-shot learning is still unable to beat models that utilize state-of-the-art fine-tuning [27]. However, instruction tuning can be combined with few-shot learning to optimize the performance of the language model.

Chain-of-Thought The Chain-of-Thought technique was initially introduced by Wei et al. [35], and involves instructing an LLM to generate a number of intermediate reasoning steps before answering the original question, given a prompt which consists of (`input`, `chain of thought`, `output`) triples. Here, `chain of thought` denotes the set of reasoning steps required to arrive at the answer in the output. Wei et al. [35] showed results that proved chain-of-thought could outperform standard few-shot prompting in reasoning tasks, then moved on to conclude that chain-of-thought does not improve performance on smaller models, but only yield performance gains in models with more than ∼100B parameters. Contrarily, smaller models tends to produce fluent but illogical chains of thought, consequently performing worse than few-shot prompting. However, chain-of-thought did seem to improve performance of larger models, especially in more complicated reasoning tasks. It is important to note that there is no guarantee for correct reasoning paths.

2.8.2 Utilizing Custom Data in LLMs

Retrieval Augmented Generation (RAG) Retrieval Augmented Generation (RAG) is a technique first introduced by Lewis et al. [36] in 2021. Traditional LLMs primarily rely on vast amounts of training data and their internal algorithms to generate responses. They are typically pre-trained on a large corpus of text, enabling them to replicate information which is present in that data. However, they often struggle with providing up-to-date or highly specific information because their knowledge is limited to what was available during training. RAG is a relatively new and innovative approach in the field of LLMs that combines the strengths of both retrieval-based and generative models. This hybrid technique pairs any LLM with a retrieval system which searches a separate database for information relevant to the query at hand. The separate database can be updated regularly, allows private data such as company proprietary information to be included, and allows the LLM to be "grounded" in real-world information. RAG is a noteable departure from how LLMs traditionally operate, and represents a significant step forward in making LLMs more reliable and versatile for various applications.

A simple overview of RAG using a vector database is shown in Figure 6. Using RAG, documents are processed through an embedding model to create vector representations of their content. These vector representations are stored in a vectorstore, which is a database designed to handle high-dimensional data. When a user submits a query, it is embedded using the same embedding model. The query embeddings are used to retrieve relevant information from the vectorstore. This information is subsequently added to the original query, before being sent to the LLM. Finally, the LLM generates a response that is delivered back to the user.

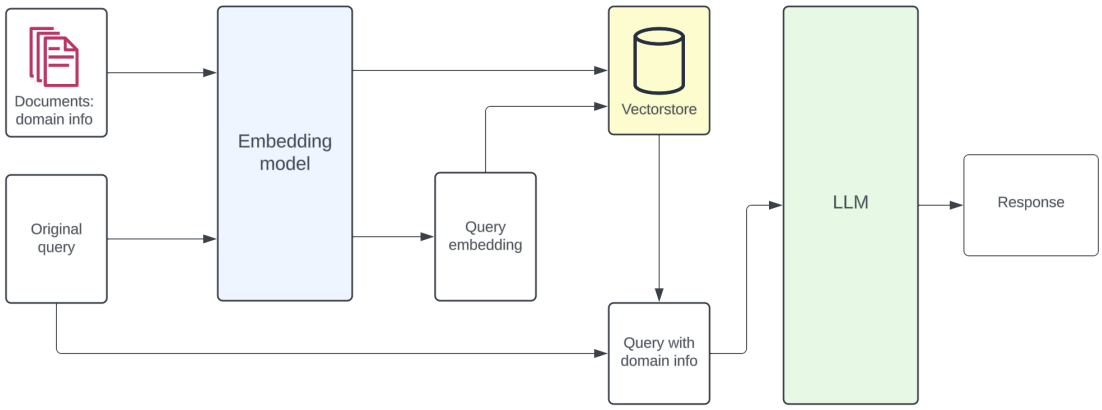

Figure 6: Overview of Retrieval Augmented Generation (RAG)

Since 2021, the concept of RAG has seen significant further innovation. Gao et al. [37] investigated the state of RAG in 2024 and categorized it into three stages: Naive RAG, Advanced RAG and Modular RAG.

Naive RAG represents the initial stage, characterized by a straightforward retrieval and generation process. It involves indexing documents, retrieving relevant chunks based on semantic similarity, and generating responses using these retrieved chunks. While effective, this method often struggles with issues such as retrieval precision, hallucination, and redundancy in generated responses. [37]

Advanced RAG introduces enhancements to address the limitations of Naive RAG. It includes pre-retrieval and post-retrieval optimization strategies. Pre-retrieval optimizations include refining the indexing process and query transformation techniques like query rewriting and expansion to improve retrieval accuracy. Post-retrieval optimizations involve re-ranking and compressing the retrieved chunks to ensure the most relevant information is used in the generation process. [37]

Modular RAG further evolves the concept by adding specialized modules and introducing iterative and adaptive retrieval processes. This approach allows for greater flexibility and adaptability, enabling the system to handle more complex queries and scenarios. Modular RAG can integrate various functional components, such as memory modules and routing mechanisms, to enhance the retrieval and generation efficiency. [37]

Islam et al. [38] also made an interesting finding related to RAG, namely how to structure the prompt with context to different LLMs. Their study revealed that structuring the prompt with the context presented first rather than last resulted in significant performance improvements with both Meta AI's Llama2 and OpenAI's GPT-4 models.

Fine-Tuning Fine tuning is a technique for adapting pre-trained language models to specific domains or new tasks by further training on an additional, often smaller dataset. This process updates and optimizes the model's parameters for the given task or domain. [39]

Instruction tuning is a specialized form of fine tuning. The technique adjusts the weights of language models after the initial pre-training by training the model using `(instruction, output)` tuples. `Instruction` is the input given as a query to the language model, while `output` is the target response that the language model should generate. The main difference between instruction tuning and standard supervised fine-tuning lies in the data used for training. While supervised fine-tuning train models on input examples and their corresponding outputs, instruction tuning augments these examples with instructions, enabling the models to generalize more easily to new tasks. This can make the model more predictable by aligning its output with the linguistic characteristics in the training data. [40]

Regardless of benefits such as improved performance on trained instructions and better alignment with users' intentions, there are several challenges with fine- and instruction tuning. Firstly, it is difficult to craft high-quality tuning datasets. Secondly, Kung and Peng [41] concluded that although instruction tuning improves the performance of language models, the gains are likely due to learning superficial patterns like output format, rather than fully utilizing the provided instructions. Therefore, it is unlikely that instruction tuning is an efficient way to feed domain-specific information into a language model, and the reliability of the model in utilizing information from the tuning dataset remains uncertain.

2.9 Text Similarity Evaluation Metrics

This section describes metrics commonly used to evaluate the similarity between texts, which in turn can help evaluate the quality of responses generated by large language models. Section 2.9.2

describes the METEOR score, which incorporates precision, recall, and stemming, to produce a more nuanced measure of text similarity. Section 2.9.1 describes the ROUGE score, a set of metrics that primarily focuses on the overlap of n-grams and longer sequences to evaluate the content similarity between texts. Together, these metrics can robustly assess the performance of language models in generating contextually relevant responses.

2.9.1 Recall-Oriented Understudy for Gisting Evaluation (ROUGE)

ROUGE, introduced by Lin [42], is a set of metrics used for evaluating automatic summarization and machine translation. This section will describe ROUGE-N and ROUGE-L.

ROUGE-N measures the overlap of n-grams between a candidate summary and reference summaries. It is expressed as

$$\text{ROUGE-N} = \frac{\sum\limits_{S \in \{ReferenceSummaries\}} \sum\limits_{gram_n \in S} Count_{match}(gram_n)}{\sum\limits_{S \in \{ReferenceSummaries\}} \sum\limits_{gram_n \in S} Count(gram_n)} \tag{16}$$

where n is the length of the n-gram $gram_n$, $Count_{match}(gram_n)$ is the number of times the n-gram is found in both the candidate summary and the reference summary, and $Count(gram_n)$ is the number of times the n-gram appears in the reference summary.

Notice that the numerator sums over all reference summaries, effectively giving more weight to matching n-grams occurring in multiple references. Thus, a candidate summary that contains words shared by more references is favored by the ROUGE-N measure. The author of [42] states that this is intuitive because we normally prefer a candidate summary that is more similar to consensus among reference summaries.

ROUGE-L measures the longest common subsequence (LCS) between two summaries. To apply LCS, a summary sentences is viewed as a sequence of words. Intuitively, the longer the LCS of two summary sentences is, the more similar the two summaries are.

The author of [42] proposes using LCS-based F-measure to estimate the similarity between two summaries X and Y. Assume that X is a reference summary sentence, and Y is a candidate summary sentence. We define

$$R_{lcs} = \frac{LCS(X,Y)}{length(X)}$$

$$P_{lcs} = \frac{LCS(X,Y)}{length(Y)}$$

$$ROUGE\text{-}L = F_{lcs} = \frac{(1+\beta^2)P_{lcs}R_{lcs}}{R_{lcs} + \beta^2 P_{lcs}}$$

where $LCS(X,Y)$ is the length of the longest common subsequence of X and Y, and β is an

F-measure coefficient used to balance the importance between P_{lcs} and R_{lcs}. Notice that when $X = Y$, $R_{lcs} = P_{lcs} = 1$ and $ROUGE\text{-}L = 1$. Contrarily, $ROUGE\text{-}L = 0$ when $LCS(X, Y) = 0$.

ROUGE-L comes with a number of advantages. Since ROUGE-L captures the longest in-sequence common n-grams, it automatically includes n-grams of varying lengths. Unlike the recall-based ROUGE-N and precision-based BLEU metric, ROUGE-L combines both precision and recall using the F-measure. In the following example used by [42]:

Reference (R):	police killed the gunman
Generated text 1 (G1):	police kill the gunman
Generated text 2 (G2):	the gunman kill police
Generated text 3 (G3):	the gunman police killed

$G1 > G3 > G2$ in terms of summary quality. $LCS(G1, R) = 3$ while $LCS(G2, R) = 2$, meaning that G1 would rightfully obtain a higher ROUGE-L score. Meanwhile, the ROUGE-2 scores would be equal for G1 and G2. However, even though $G3 > G2$, they would obtain the same ROUGE-L scores, because ROUGE-L would either consider '*the gunman*' or '*police killed*' in G3 as the LCS, but not both. Meanwhile, ROUGE-2 would prefer G3 over G2. [42]

2.9.2 Metric for Evaluation of Translation with Explicit ORdering (METEOR)

METEOR is an evaluation metric for machine translation output developed by Denkowski and Lavie [43]. It is based on unigram (word) matching between machine-generated translations and human reference translations. What sets METEOR apart from other metrics is that it considers not just exact word matches, but also matches between word stems and synonyms.

The METEOR scoring process typically follows three stages. First, the metric identifies exact matches between unigrams in the machine and reference texts. Next, it matches unigrams that share the same word stem, even if the surface forms differ. Finally, it uses a synonym dictionary to match unigrams that have the same meaning but different surface forms and stems.

By considering stemmed and synonym matches in addition to exact matches, METEOR aims to better capture the semantic similarity between the machine translation and the reference, even if the wording is not identical. This is intended to align the metric more closely with human judgments of translation quality.

In their evaluation, Denkowski and Lavie [43] found that METEOR scores had a higher correlation with human judgments compared to several other commonly used metrics, including BLEU, precision, recall, and the F measure, suggesting that by incorporating stemming and synonymy, METEOR may provide a more accurate automatic assessment of translation quality that better reflects human perceptions.

3 Related work

This section presents an overview of recent research that builds upon the foundational theories and concepts introduced in Section 2. The aim is to investigate and show recent advancements and trends directly relevant to our research questions on enhancing RAG systems.

Section 3.1 investigates recent work within evaluating IR systems with relevance feedback. In Section 3.2, we explore recent developments in query expansion techniques using large language models. Section 3.3 describes interesting findings of how query and document characteristics can impact the effectiveness of relevance feedback. In Section 3.4 and Section 3.5 we examine studies on mitigating hallucination in LLMs and multi-agent LLM design patterns, addressing the generation aspect and potential improvements through collaborative AI approaches. Finally, in Section 3.6 we review current methods for evaluating LLM-based systems, including RAG-specific evaluation frameworks, to inform our approach to assessing the performance of our proposed enhancements.

3.1 Evaluation of IR Systems Employing Relevance Feedback

Huiskes and Lew [44] review strategies for evaluating IR systems with relevance feedback.

Two characteristics define the standard approach for evaluating IR systems. First, the search problems are specified. The IR systems are tested as a whole, and the relevance feedback algorithms are evaluated by comparing the quality of ranking of the IR system when all conditions are kept equal except the RF algorithm itself. The most common way to evaluate ranking quality is by using precision-recall graphs. Additional performance measures derived from the relevance rankings such as mean average precision (MAP) are often used as a summarizing evaluation measure. [44]

3.2 Query Expansion Using LLMs

Jagerman et al. [45] leverage LLMs for query expansion by directly updating the textual query with generated LLM responses. The initial query q_0 is presented to an LLM in a prompt, asking it to generate a hypothetical response. This response is concatenated to q_0 to obtain a new query, which is used by the IR system. The method is illustrated in Figure 7.

The authors of [45] experimented with different prompts, including zero-shot, few-shot and chain-of-thought (CoT). They found that chain-of-thought prompts yielded the best results, likely because they ask the language model to break down its answer step-by-step, thus including many keywords that can aid in query expansion.

Similarly, Wang et al. [46] expanded queries with pseudo-documents generated by LLMs. The aim of the pseudo-documents was to answer the original query. Experimental results showed an improvement over standard BM25 by 3-15% on IR datasets like MS-MARCO and TREC DL without

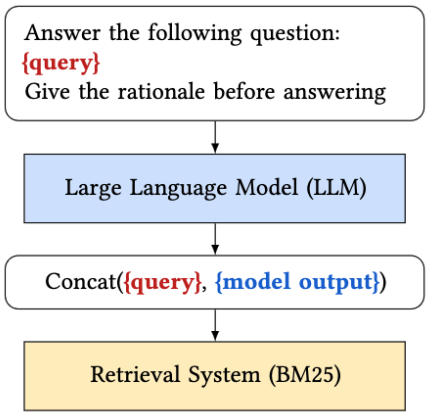

Figure 7: Query expansion method proposed by Jagerman et al. [45]. An LLM first answers the original query. The response is included in the query that is used by the retrieval system.

model finetuning. The results also illustrated the effect of model scale on the quality of query expansion, showing that performance steadily improves when going from the 1.3B model up to 175B models. Empirically, texts generated by smaller language models were shorter and contained more factual errors.

3.3 Heuristics for Adapting Coefficients in Relevance Feedback

An important challenge in relevance feedback is how to balance the original query and the relevance information given by a user. Over-trusting the user feedback may lead to bias towards a particular subset of relevant documents. Under-trusting would conversely not take advantage of the provided feedback. Relevance feedback algorithms typically use fixed parameters across all queries and collections. Ideally, such parameters should be optimized for each query and collection. Therefore, Lv and Zhai [47] introduce three heuristics to characterize the balance between query and feedback information.

(1) Discrimination of Query The authors argue that if a query is more discriminative, i.e. more effective at differentiating between relevant and non-relevant documents, it could be more drifting-tolerant, and thus it would be safe to utilize more feedback information.

A longer query is generally more discriminative than a shorter one. The first discrimination measure is therefore query length:

$$|Q| = \sum_{w \in Q} c(w, Q) \tag{17}$$

where $c(w, Q)$ is the number of times w appears in Q.

The entropy of a query measures the uncertainty or ambiguity associated with it. Queries with high entropy have unclear intent, leading to a wide range of possible results. For example, "band" could give results for music bands, rubber bands or wedding rings. Conversely, "blue rubber band" is much more specific. Queries with high entropy are more prone to drifting. The second discrim-

ination measure is therefore the entropy-score based on the top-k result documents F' in the initial retrieval:

$$Q_{entropy} = \sum_{w \in F'} -p(w|\theta_{F'})\ log_2 p(w|\theta_{F'})$$

Here, $p(w|\theta_{F'})$ is the probability of finding term w given a model of the term distribution in F'. The authors estimate it with

$$p(w|\theta_{F'}) = \frac{c(w, F')}{\sum_w c(w, F')}$$

where $c(w, F')$ is the number of times term w appears in the feedback document set F', and $\sum_w c(w, F')$ is the total count of all words in the feedback document set. This gives

$$Q_{entropy} = \sum_{w \in F'} -\frac{c(w, F')}{\sum_w c(w, F')}\ log_2 \frac{c(w, F')}{\sum_w c(w, F')} \tag{18}$$

Intuitively, given Equation 18, the entropy is high if the term distribution is uniform, meaning that many terms occur with a similar frequency. Contrarily, having a skewed term distribution with only a few words that dominate F' means that the feedback documents are focused on specific topics, and the entropy is low.

Lastly, query clarity is proposed as a query discrimination metric. High query clarity means that the query is effective in retrieving relevant documents.

(2) Discrimination of Feedback Documents If feedback documents are more discriminatory, they likely focus more on the relevant topic and contain less noise. Thus, they can be trusted more in the feedback process.

Firstly, the number of feedback documents plays a role, as a single document is less discriminatory than multiple documents.

Secondly, the authors define feedback radius as a measure of feedback document discrimination. If the feedback documents are within a small radius of the feedback document centroid, meaning that they are similar, they can be trusted more.

Thirdly, the term distribution within feedback documents is important. A feedback document with a skewed distribution of terms can be trusted more, as it is more specific. Contrarily, a uniform term distribution indicates that the feedback document covers multiple topics, and should not be trusted as much in a relevance feedback loop.

Lastly, the authors of [47] discuss feedback document clarity as a measure of feedback document discrimination. Whereas entropy measures the internal uncertainty of feedback documents, clarity estimates how distinct and focused the feedback documents are in relation to the entire collection. Documents with high clarity can be trusted more.

(3) Divergence Between Query and Feedback Documents Lv and Zhai [47] argue that we may rely more on feedback documents if the divergence between a query and its feedback documents

is large. Specifically, if the query diverges from the relevant documents in the feedback document set, it may be a poor representation of the user's information needs, and the relevance judgments should be trusted more.

Conclusively, the authors found that predicting the coefficient responsible for balancing query and feedback document emphasis using features derived from these heuristics made the relevance feedback more robust and effective, compared to a fixed-coefficient approach. This underlines the importance of these heuristics in determining the efficiency of relevance feedback.

3.4 Mitigating Hallucination in LLMs

Hallucination in LLMs is a significant challenge that hinders reliable deployment in real-world applications. "Hallucination" refers to instances where an LLM generates outputs that appear plausible but are not grounded in the provided data or real-world facts. This phenomenon, while metaphorically named after human perceptual experiences, arises due to the probabilistic nature of LLMs and the vast, unverified text corpora they are trained on. [48]

The issue of hallucination is particularly problematic in single-agent LLM systems, where the model relies solely on its own knowledge and generation capabilities. Without external validation or feedback mechanisms, a single-agent LLM may confidently produce hallucinated outputs, undermining its trustworthiness and usability in critical domains such as healthcare, finance, and legal advice.

To address hallucination, Tonmoy et al. [49] has investigated common techniques to mitigate it, aiming to improve the reliability of LLM outputs. They split it into two categories:

Prompt Engineering Crafting specific prompts that guide the LLM to generate more accurate and reliable outputs. The authors define Retrieval-Augmented Generation as described in Section 2.8.2 as one such technique, as it mitigates hallucination by grounding the LLM in verifiable information, reinforcing user confidence. Additionally, feedback mechanisms and iterative refinement processes can help improve the model's accuracy by providing structured feedback loops, which continuously enhance the reliability of its outputs.

Model Development The other category of mitigating hallucinations is focused on developing models. The authors describe going further than just fine-tuning models, but implementing advanced model architectures. This includes introducing new strategies during the generation phase that prioritize contextually relevant and accurate information, thereby reducing the likelihood of hallucinations. They also describe utilizing structured data sources, such as knowledge graphs, to provide a foundation of verifiable facts. This ensures that the generated outputs are grounded in well-defined relationships and factual information. By leveraging these structured data sources, models can produce more coherent and accurate content, significantly mitigating the risk of hallucinations.

3.5 LLM-based Multi-agent Design Patterns

LLMs have evolved to have remarkable capabilities in language generation, understanding, and few-shot learning since the explosion in popularity with the release of ChatGPT by OpenAI [13]. Despite these advancements, as described in Section 3.4, current models often struggle with issues such as hallucinating facts and making implausible and incorrect reasoning jumps due to the inherent limitations of training on massive, unverified text corpora. Traditional methods to improve LLM performance, such as few-shot prompting, self-consistency, and intermediate scratchpads, have typically focused on enhancing individual model instances.

Recent research has proposed improvements in reasoning using LLM multi-agent systems, where multiple LLMs work together in a coordinated manner to achieve better performance. This approach utilizes the diverse strengths and perspectives of different models, reducing the likelihood of errors and improving the overall quality of the generated outputs. A study by Li et al. [50] demonstrates that simply increasing the number of LLM agents and employing a sampling-and-voting method can lead to significant performance improvements across various tasks, including arithmetic reasoning, general reasoning, and code generation.

The study reveals that the performance of LLMs scales with the number of agents instantiated, and this method is complementary to existing complex methods for enhancing LLMs. Importantly, the degree of enhancement correlates with task difficulty, suggesting that multi-agent approaches are particularly beneficial for more challenging problems. This finding sparks an interest to investigate multi-agent design patterns in LLM applications, as it indicates that leveraging multiple agents can yield substantial improvements in task performance without the need for more complex prompt engineering or collaboration frameworks.

In the following sections, we will explore some LLM multi-agent design patterns in detail, illustrating how they enable LLMs to collaborate, dynamically adapt and improve through collaborative and iterative processes.

Tool Use As a design pattern, Tool-use enables LLMs to leverage external tools and APIs to expand their capabilities and adapt to new tasks without retraining. Tool-use is illustrated in Figure 8. This approach was suggested in "Gorilla: Large Language Model Connected with Massive APIs" by Patil et al. [51] and "MM-ReAct: Prompting ChatGPT for Multimodal Reasoning and Action" by Yang et al. [52], and it discusses methods for LLMs to access and utilize a wide range of specialized tools through API calls, which can potentially enhance their problem-solving abilities. Tool use involves the LLM serving as a high-level planner and router that determines which tools should be invoked for a given task and how to compose their outputs. The LLM is provided with knowledge about each tool's capabilities, input arguments, and output format through prompt engineering. When presented with a user query, the LLM generates "action requests" specifying which tool to call with what arguments. The tool's output is then fed back into the LLM, which then can iteratively reason and act using one or several tools until the task is solved to its own or the user's satisfaction.

Gorilla [51] focuses on enabling LLMs to accurately write API calls for a massive number of ML tools from popular sources and repositories. By retrieving the most relevant API documentation

and providing it as additional context in the prompt, Gorilla outperforms GPT-4 in API calling correctness while mitigating hallucinations as the latest API information is retrieved on the spot. This retrieval-augmented training also allows Gorilla to adapt to changes in API documentation without retraining or fine-tuning any machine learning model.

MM-ReAct [52] integrates OpenAI's gpt-3.5-turbo with predefined external vision API tools to enhance it with multimodal reasoning capabilities. Visual inputs like images and videos are provided to the LLM as file paths, which act as placeholders that require an external API to be analyzed. gpt-3.5-turbo is instructed via prompt to invoke the predefined specific vision API endpoints (e.g. captioning, OCR, object detection) to interpret the visual content as needed. This enables MM-ReAct to tackle complex visual understanding tasks like multi-image reasoning, document understanding, and video summarization in a few-shot (prompt engineering) manner without additional training.

Tool use is a flexible pattern. By connecting LLMs with a diverse set of tools through unified API interfaces, the Tool use pattern enables open-ended task solving and knowledge gathering beyond the LLM's training data. The modularity also provides flexibility to incorporate whatever the latest tool advancements might be at the time. However, challenges remain in systematically evaluating such systems' performance. The Tool use pattern also adds an additional layer where one might have to address tool limitations, and not just LLM-agent or pattern limitations. Still, it presents an extensible and customizable method for augmenting LLMs with specialized tools to expand their reasoning and acting capabilities. This opens up possibilities for LLMs to serve as general-purpose problem solvers that can continually adapt to new tasks and domains by adding and improving upon available tools.

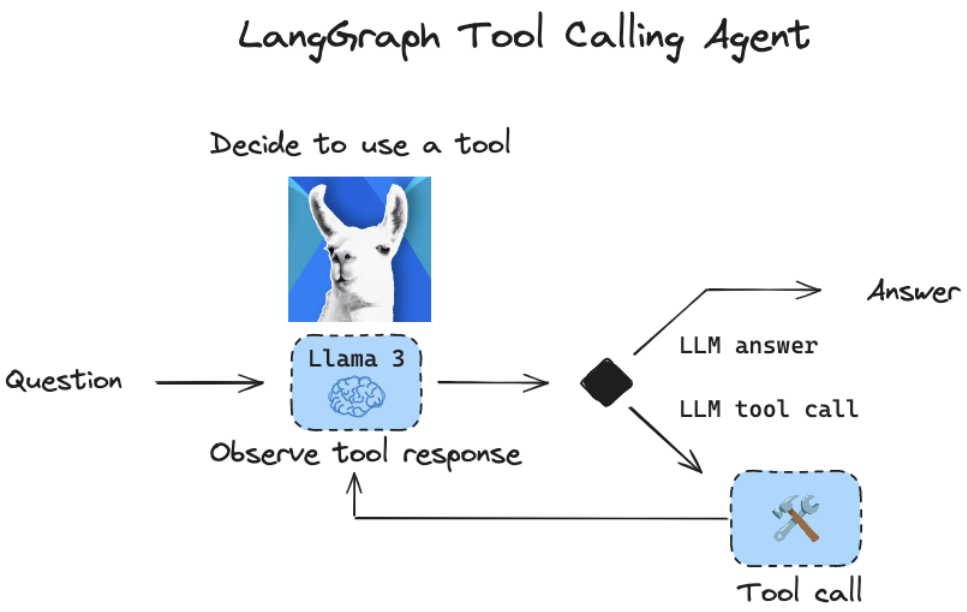

Figure 8: Tool-use illustrated by LangGraph. Retrieved from Meta AI [53]

Reflection is an agentic design pattern that enables LLMs to iteratively refine and improve their outputs through a process of self-feedback and revision. Two versions of this pattern are described in "Self-Refine: Iterative Refinement with Self-Feedback" by Madaan et al. [54] and "Reflexion: Language Agents with Verbal Reinforcement Learning" by Shinn et al. [55]. Reflection allows LLMs to autonomously enhance the quality of their generated responses without requiring additional training data or fine-tuning. The key idea behind Reflection is to have the LLM serve multiple roles - as the initial output generator, the feedback provider analyzing the initial output, and the refiner that incorporates the feedback to produce an improved output. This multi-agent setup, illustrated in Figure 9 is enacted over multiple feedback-refinement iterations until a satisfactory output is achieved according to the feedback provider, or a maximum number of iterations is reached.

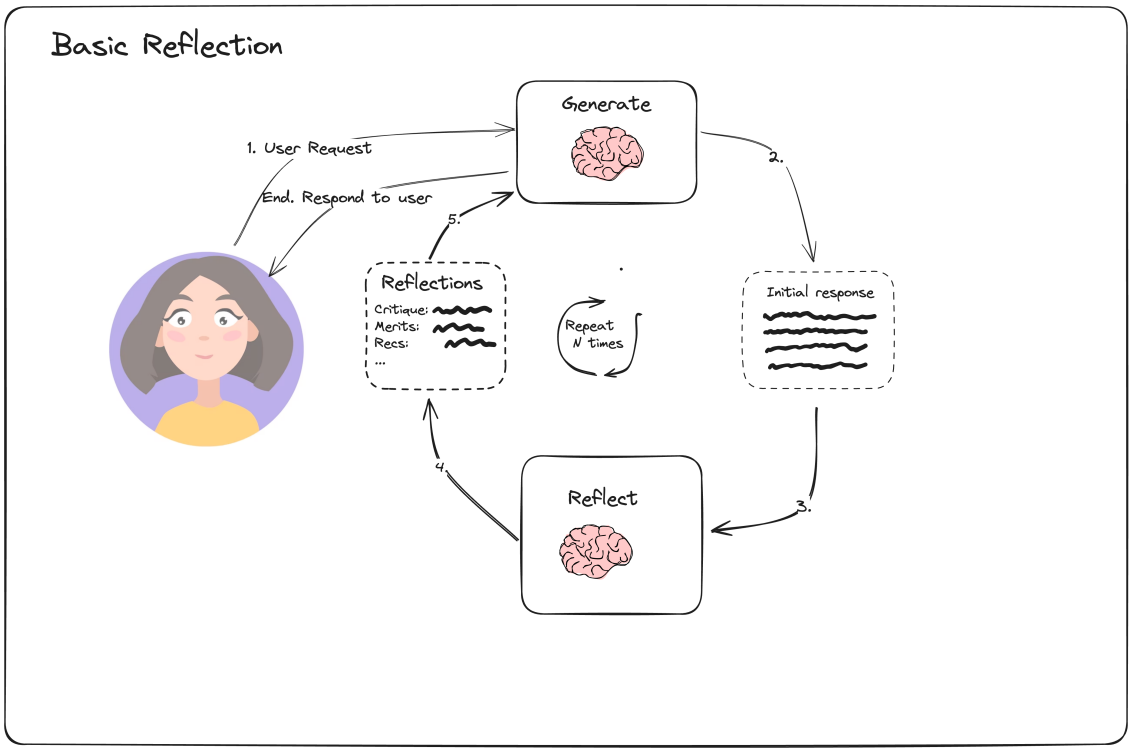

Figure 9: Reflection illustrated by LangGraph. Retrieved from LangGraph [56]

In Self-Refine [54], the LLM first generates an initial output given a prompt. It then analyzes this output and provides specific, actionable feedback on how to improve it. Finally, the feedback is fed back into the LLM-based agent along with the original prompt and previous output to generate a refined version. This process can repeat multiple times. Importantly, Self-Refine relies solely on the few-shot capabilities of the LLM using in-context examples, without any additional training.

Similarly, in Reflexion [55], an LLM-based agent called the Actor first generates an output by interacting with an environment. Another LLM agent, the Evaluator, then scores this output. The scores are turned into natural language feedback by a third Self-Reflection model. This feedback is stored in the Actor's memory to inform future decision making. The Reflexion agent continues this generate-evaluate-reflect loop until the Evaluator deems the output satisfactory.

Both papers demonstrate the effectiveness of the Reflection pattern across diverse tasks. Self-Refine achieves significant improvements over the base LLMs on tasks like dialog generation, code

optimization, math reasoning and more. Reflexion outperforms strong baselines on sequential decision making in virtual environments, question answering, and competitive programming. Reflexion also claim to achieve a remarkable 91% on the popular HumanEval code generation benchmark [57], surpassing the previous state-of-the-art 67% by GPT-4 [28].

The Reflection design pattern offers several advantages. It enables iterative improvement of LLM outputs without the computational cost of model fine-tuning. The natural language feedback provides interpretable and nuanced guidance compared to the numerical scalar reward ("score") that is traditionally used in reinforcement learning. Instead of relying on a single numerical value to capture the quality of the output, we get "verbal" feedback we can use to pinpoint specific areas to be changed and improved. Similarly to how humans learn, detailed constructive feedback is more helpful than just a score. And finally, storing the model's "thoughts" and self-feedback into some accessible memory makes the reasoning process more transparent. This memory serves as a record of the agent's decision making process and the key insights it has gathered through the reflection loops. It allows for easier analysis and debugging of the agent's behavior, as one can trace back the steps that led to a particular output.

The Reflection design pattern introduced by these two papers presents a powerful and flexible framework for LLMs to autonomously refine the quality of their outputs through multi-agent self-feedback loops. This paves the way for more capable and robust frameworks that enable language models to be able to dynamically adapt and improve through reflection on their own generations.

Planning Planning, depicted in Figure 10, is an agentic design pattern that enables LLMs to decompose complex tasks into multi-step procedures and reason through them to arrive at solutions. Several recent papers have demonstrated the effectiveness of equipping LLMs with planning capabilities to tackle challenging reasoning problems.

Building on the work by Wei et al. [35] described in Section 2.8.1, Wang et al. [59] propose Plan-and-Solve (PS) prompting to further improve zero-shot CoT reasoning in large language models. They find that Zero-shot-CoT still suffers from three main pitfalls: calculation errors, missing reasoning steps, and semantic misunderstanding. To address these issues, PS prompting guides the LLM to first devise an explicit plan that breaks down the problem into subtasks, and then execute that plan step-by-step. The authors also introduce PS+ prompting, which extends PS with more detailed instructions to reduce errors and improve the quality of the generated reasoning. Across ten datasets spanning arithmetic, commonsense, and symbolic reasoning, Wang et al. [59] shows that PS+ consistently outperforms Zero-shot-CoT, matches the few-shot CoT performance on math problems, and even exceeds the more complex Zero-shot Program-of-Thought approach on several tasks.

Shen et al. [60] take planning a step further with HuggingGPT, an LLM-powered agent that uses planning to connect different expert models for solving complex AI tasks. HuggingGPT uses an LLM to analyze user requests, break them down into solvable sub-tasks, determine the execution order based on dependencies, and assign appropriate expert models to each sub-task. An LLM acts as a central controller that coordinates the planning and execution of multiple expert models to collectively fulfill the user's high-level objective. This global planning strategy allows HuggingGPT

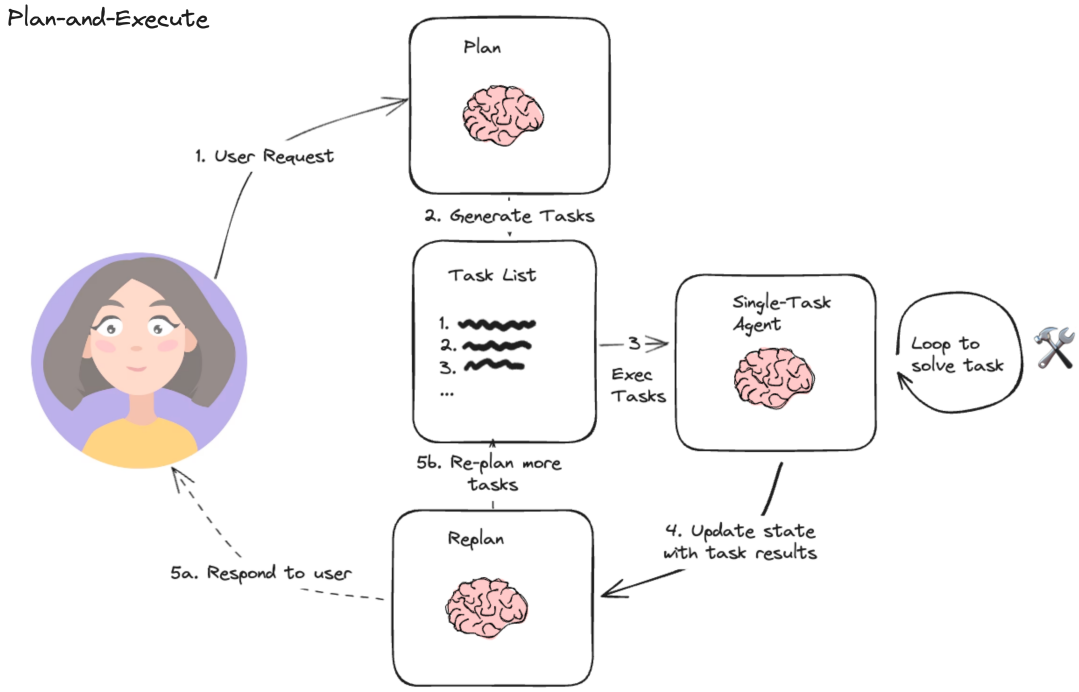

Figure 10: Planning illustrated by LangGraph. Retrieved from LangGraph [58]

to flexibly combine the strengths of specialized models to handle multimodal requests spanning language, vision, speech, and more.

Together these papers show how multi-step reasoning via planning can significantly expand the problem-solving capabilities of LLMs, in multiple different ways but generally using the same "Planning" design pattern. By decomposing complex tasks into intermediate steps, planning makes the reasoning process more interpretable and robust. Explicit plans show the model's reasoning in natural language, providing a view into its reasoning and decision making. Planning abilities can be found naturally in sufficiently large language models through in-context learning, without requiring additional fine-tuning.

The success of approaches like CoT prompting, PS prompting, and HuggingGPT show planning as a powerful design pattern for eliciting multi-step reasoning from LLMs across diverse domains. Key challenges going forward include improving the reliability and optimality of generated plans, enabling open-ended planning in environments with large action spaces, and integrating planning with other high-level skills like causal reasoning and abstraction.

Multi-agent Collaboration/debate enables LLMs to cooperate and solve complex tasks through multi-turn conversations in tackling challenging real-world problems. An "agent" can be one or several different LLMs, specialized for different tasks.

Du et al. [61] describes a debate multi agent collaboration pattern that enhances the factuality and reasoning capabilities of LLMs by using a multi-agent debate mechanism. This approach gen-

erates multiple individual responses to a query and then have the LLMs engaging in rounds of critique and revision based on the responses of other agents. Each agent updates its answer after considering the critiques, iteratively refining the common, collective output. This process significantly improves the accuracy and consistency of the final answer by allowing the models to cross-examine and correct each other's reasoning. The debate pattern has shown significant improvements in tasks requiring mathematical and strategic reasoning, as well as in reducing hallucinations and factual inaccuracies in generated content. By converging on a consensus through debate, LLMs can produce more reliable and well-reasoned outputs without the need for additional training data or model fine-tuning.

AutoGen by Wu et al. [62] with Microsoft, is a general-purpose framework for building diverse LLM applications through multi-agent conversations. There are two important concepts AutoGen introduces: customizable and conversable agents that can leverage LLMs, human inputs, tools, or a combination of them, and conversation programming that unifies complex application workflows into multi-agent conversations. AutoGen agents have unified conversation interfaces and auto-reply mechanisms that naturally induce conversation flows. Both natural language and code can be used to flexibly program agent interactions and support static, dynamic, and group conversation patterns. Wu et al. [62] demonstrate AutoGen's effectiveness across six applications: math problem solving, retrieval-augmented code generation and question answering, text-based decision making, multi-agent coding, dynamic group chat, and more. Results show that AutoGen can match or exceed state-of-the-art performance while enabling innovative interaction modes and reducing development effort.

MetaGPT by Hong et al. [63] is a meta-programming framework that uses human work methods and Standardized Operating Procedures (SOPs) in LLM-based multi-agent collaboration. It encodes SOPs into prompt sequences to enable more streamlined workflows and allows specialized agents with human-like domain expertise to verify intermediate results, reducing errors. MetaGPT utilizes the idea of an assembly line, assigning diverse roles to various agents and efficiently breaking down complex tasks into subtasks. On collaborative software engineering benchmarks, MetaGPT generates more coherent solutions than previous chat-based multi-agent systems. The integration of SOPs significantly enhances robustness by reducing unproductive agent chatter. MetaGPT also introduces an executive feedback mechanism that debugs and executes code during runtime, elevating code generation quality.

3.6 Evaluation of LLM-based Systems

The use of very capable LLMs both in a single and multi agent manner, can introduce new ways to approach problem-solving and decision-making processes in AI. However, unlike traditional single-agent models, multi-agent systems take advantage of the collective intelligence and potentially specialized capabilities of multiple LLMs, facilitating more complex interactions and decision-making, comparable to human group dynamics. As we discovered in Section 3.5, this approach not only enhances the models' ability to simulate complex real-world environments but also significantly improves their planning, reasoning, and collaborative capabilities.

3.6.1 Traditional Metrics

Evaluating single-agent LLMs involves assessing their performance on tasks in isolation. These tasks can range from question-answering and text summarization to code generation and creative writing. Traditional metrics investigated in Section 2.7 are often used to quantify the accuracy, relevance and quality of their outputs in comparison to an ideal output.

Sai et al. [64] found that existing, conventional metrics such as Rouge and Meteor is largely popular due to lack of better alternatives. They describe that these traditional automated metrics are can be hard to explain, have poor correlations with human judgement, fail to capture nuances in language and prone to bias.

3.6.2 Benchmarks for LLM Evaluation

Benchmarks provide a standardized way to evaluate LLM performance on specific tasks or capabilities. These benchmarks typically consist of a dataset of input samples and a set of evaluation metrics to measure the quality of the model's output. Beeching et al. [65] runs a leaderboard for open LLMs to evaluate and rank individual models based on a collection of benchmarks.

One example is GSM8K by Cobbe et al. [66], a benchmark focusing on high school-level math problem-solving. GSM8K uses a dataset of real-world math problems, requiring models to demonstrate a range of capabilities to solve them successfully. However, evaluating success or failure in such holistic benchmarks can be complex, as it is difficult to pinpoint the exact cause of a model's success or struggle.

Zhou et al. [67] investigated data leakage when using static datasets for evaluating LLM based systems. Data leakage involves a scenario where the model is pre-trained on the data it is being tested on, leading to an overestimation of the model's performance. If the model is exposed to a benchmark's evaluation data during training, there is a strong possibility that it will be able to achieve a high score on the same evaluation benchmark in question. Whereas many open models include details on their training data in model cards, proprietary models such as GPT-4 by OpenAI [68] are more secretive or vague. Mitigation techniques to avoid data leakage problems include using multiple benchmarks from different sources and performing manual evaluation and qualitative analysis.

3.6.3 LLM-as-a-Judge

One technique that has been explored by Zheng et al. [69] is to use strong LLMs as judges to evaluate other models on more open-ended questions. This approach, known as LLM-as-a-Judge, uses larger state of the art models to assess the performance of other models by comparing their responses to various outputs or prompts to then determine which one aligns better with human preferences.

The authors propose three potential variations of LLM-as-a-Judge. The first variation is presented a pairwise comparison and determines which one is better, or declares a tie. The second variant assigns a score to a single answer based on its quality. The third and final variant is provided a reference solution to assist in grading it with either a boolean true or false, or a score. [69]

The LLM-as-a-Judge technique significantly reduces the need for human involvement, and it enables scalable benchmarks that are both faster and cheaper than humans. It is also explainable and interpretable, as an LLM is able and will reason in natural language if instructed to do so. However, the authors also highlight clear limitations such as sensitivity to prompt structure and verbosity, self-enhancement bias where the LLM favors responses generated by itself, and limitations in capability particularly where the task requires advanced reasoning. The limitations can however be somewhat mitigated by changing the structure of evaluation prompts, few-shot examples, chain-of-thought and providing reference or ideal answers.

Wang et al. [70] uncovered systematic bias when using LLMs as evaluators. They find that LLMs tend to prefer longer outputs, and that the relative position of content in the outputs can change the evaluation decisively. Similarly to the work described in by Zheng et al. [69] in Section 3.6.3, they also find that LLMs has a self confirmation bias preferring outputs similar to its own outputs.

3.6.4 Evaluating Multi-agent LLM systems

Research on Multi-agent systems in the context of LLM usage is still in early stages, and consequently lacks diverse and varied peer-reviewed research or evaluation methodologies.

Wu et al. [62] with Microsoft, one of the earliest researching LLM multi-agent systems, introduces a perspective on evaluating multi-agent LLM systems through their development of the AutoGen framework. The AutoGen framework was also discussed in Section 3.5. In the context of evaluation, AutoGen emphasizes the importance of both qualitative and quantitative metrics to capture the performance and efficiency of multi-agent interactions. The framework assesses the correctness of solutions, task completion success rates, user interaction time, and the number of LLM calls. These metrics help in understanding the effectiveness of agent collaboration and the overall efficiency of the multi-agent setup. AutoGen also describes the role of human-in-the-loop evaluation, where human feedback is incorporated to improve the system's performance.

Chen et al. [71] proposes the LLMARENA framework, dynamic benchmarks specifically designed for multi-agent interaction. LLMARENA encompasses seven distinct gaming environments, employing TrueSkill™ scoring to assess abilities in LLM agents, including spatial reasoning, strategic planning, numerical reasoning, risk assessment, communication, opponent modeling, and team collaboration. The TrueSkill™ ranking system, developed by Herbrich et al. [72] at Microsoft Research, involves probabilistic inference to rank players based on their performance, allowing for the evaluation of agents' relative skills in dynamic and competitive settings. By leveraging TrueSkill™, LLMARENA ensures that the evaluation reflects the improving capabilities of LLM agents as they interact in varied and complex scenarios. This approach not only measures individual competencies but also evaluates how well these agents can adapt, collaborate, and compete in a multi-agent environment.

3.6.5 Evaluating Retrieval-Augmented Generation LLM Systems

Retrieval-Augmented Generation (RAG) systems, as discussed in Section 2.8.2, enhance the capabilities of LLMs by providing them with relevant context retrieved from an external knowledge base. This allows LLMs to access up-to-date information and answer questions more accurately. However, evaluating the performance of RAG systems presents challenges, as there are multiple dimensions to consider: the quality of the retrieved context, the faithfulness of the generated answer to the context, and the relevance of the answer to the original question.

To address these challenges, Es et al. [73] introduce RAGAs (Retrieval Augmented Generation Assessment), a framework for automated, reference-free evaluation of RAG systems. RAGAs provides a range of metrics that assess different aspects of RAG performance without relying on ground truth human annotations.

The first metric, Faithfulness, measures the degree to which the generated answer is grounded in the retrieved context. It uses an LLM to extract key statements from the answer and then verifies whether each statement can be inferred from the context. The second metric, Answer Relevance, evaluates how well the answer addresses the original question. It prompts an LLM to generate potential questions based on the answer, and then compares the similarity of these questions to the original input question. The third metric, Context Relevance, assesses the focus and conciseness of the retrieved context. It uses an LLM to extract sentences from the context that are essential for answering the question, and computes the ratio of extracted sentences to the total number of sentences in the context.

To validate these metrics, Es et al. [73] also introduce WikiEval, a new dataset of question-context-answer triples annotated with human judgments of faithfulness, answer relevance, and context relevance. Experiments on WikiEval show that the RAGAs metrics align closely with human assessments, especially for faithfulness and answer relevance.

4 Methodology

The research methodology employed in this Master's thesis followed an experimental design framework, consisting of two distinct phases: an exploratory study and a series of controlled experiments. Experimental design, described by Bryman and Bell [74], is a systematic approach to investigating the relationship between independent variables and dependent variables by manipulating the independent variables and measuring the effects on the dependent variables while controlling for other factors.

During the exploratory phase, a prototype RAG application was developed, further described in Section 4.1. This application allowed for the investigation of challenges of RAG systems in the context of complex technical product support. Through testing and observation, a number of limitations and areas for improvement were identified, ultimately shaping the research questions addressed in this thesis.

The controlled experiments were designed to systematically investigate the research questions outlined in Section 1.2. For RQ1, which focused on the potential of LLMs to mimic human relevance feedback in IR systems, a factorial design methodology was employed. Factorial design as described by Montgomery [75] is an experimental approach that allows for the simultaneous examination of multiple independent variables and their interactions, enabling us to provide insights into the strengths and weaknesses of utilizing different combinations of language models and prompting techniques for relevance feedback.

RQ2 explored the effectiveness of different multi-agent LLM conversation patterns in a RAG system designed for resolving technical support tickets. To address this question, a single-factor experimental design was used. The single-factor experiment design is similar to factorial design, but as the name suggests, it focuses on manipulating a single independent variable while keeping all other variables constant, allowing for the isolation and examination of the effect of the single factor on the dependent variables [75]. This approach allowed us to evaluate various conversation patterns, allowing to investigate which multi-agent communication pattern is most suitable for generating responses to technical product support tickets as part of a RAG system.

Using this methodology, we aim to provide reliable, replicable and valid results as per the guidelines of Bryman and Bell [74], along with practical recommendations for enhancing the efficiency and accuracy of RAG systems in the context of complex technical product support.

Section 4.1 thoroughly describes the exploratory study conducted as part of this thesis, and ends by summarizing our findings and explaining how they motivated our research questions. Sections 4.2 and 4.3 provide detailed descriptions of how we conducted the controlled experiments for RQ1 and RQ2, respectively, justifying the use of libraries and frameworks, and specifying dependent and independent variables, baselines, evaluation metrics and datasets.

4.1 Exploratory Study

As described in the introduction of this thesis, KSSO's technical staff dedicated a large proportion of their time on tasks related to customer support. In our specialization project, Retrieval-Augmented Generation (RAG) was identified as a promising solution to aid KSSO staff in customer support.

According to Oates et al. [76], an exploratory study can be used to gain insights into a topic about which there is limited existing knowledge or theory. A real-life instance is investigated to identify the topics to be covered in a subsequent study. Due to little available research in utilizing RAG in customer support operations, we were motivated to conduct an exploratory study, involving developing a prototype RAG system for KSSO. This would help us better understand the challenges of RAG in customer support. The concept was to have a RAG solution draft responses which could be reviewed by KSSO employees before sending them to customers. Direct customer communication was not desired due to the associated privacy risks. This section describes how the RAG prototype was developed, outlines the technical challenges we encountered, summarizes our observations based on prototype testing, and explains how they motivated the research questions for this thesis.

Section 4.1.1 describes the data gathering module in which Atlassian and KSSO documentation was retrieved and pre-processed. Section 4.1.2 explains how the retrieved documents were embedded and stored in preparation for retrieval. Section 4.1.3 details the search module, which was responsible for retrieving relevant documents from various sources, given a search query. Section 4.1.4 describes how the search module was coupled with an LLM to generate a response to a support ticket. Together, these modules constitute a complete RAG solution which at the time of development employed state-of-the-art techniques tailored to maximize its performance for support ticket resolution. Figure 11 shows a full overview of the developed prototype. Finally, section 4.1.5 summarizes the observations from the exploratory study, and describes how they motivated the research questions for this thesis.

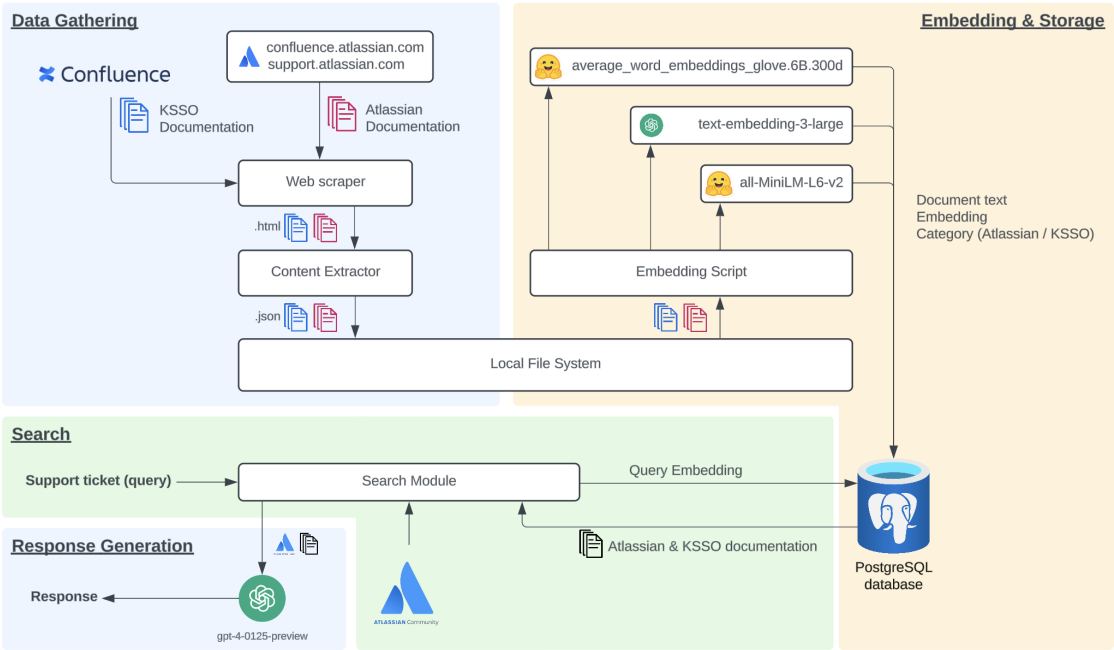

Figure 11: Overview of the RAG prototype developed in the exploratory study. The data gathering module scraped KSSO's and Atlassian's documentation, and extracted the core contents. An embedding script created 3 embeddings using different embedding models, and stored these together with a category tag and the original text in a PostgreSQL database. A support ticket was used as a query in the search module, which embedded it and fetched relevant KSSO documentation, as well as related Atlassian Community posts. Documents were sent along with the query to an LLM, which generated a final response.

4.1.1 Data Gathering Module

The aim of the data gathering module was to fetch all relevant data from various web-based sources. This included internal KSSO documentation, as well as domain-specific Atlassian-pages. After gathering, the raw `.html` documents had to be pre-processed to extract their core textual content. Figure 11 shows an overview of the main data gathering components. The following subsections provide a detailed explanation of how data gathering was conducted.

KSSO Data At the time of development, KSSO maintained all of its internal and external documentation on Confluence, which is a web-based wiki-service developed by Atlassian. Documentation could include text, images, code, tables and files, and was easily accessible through Atlassian's Confluence API which enables fetching multiple documentation pages in the HTML format along with metadata. Retrieving all information from the wiki therefore required parsing partially structured HTML data, done by a content extractor script.

A closer inspection of KSSO's wiki revealed that images were largely used to help the reader navigate user interfaces, as opposed to containing useful textual information. The alt-text attribute often remained unused, necessitating state-of-the-art image-to-text models to parse and understand the images in the given context. In most observed cases, the most important context of the image was additionally included in textual form in the article. Due to the limited utility of parsing the images in KSSO documentation, image tags were simply omitted from the fetched HTML documents.

Articles were fetched with the Confluence API, and HTML content was parsed using the HTML parser of the BeautifulSoup library for Python. Each article was converted to an Article dataclass which included the most relevant data fields, and subsequently stored on disk in `.json` format.

Atlassian Data Having developed software since 2002, Atlassian has accumulated large amounts of documentation for different products and product versions. At the time of development, documentation was not centralized, meaning that Atlassian hosted its articles under different subdomains. For example, documentation for data center products was available on confluence.atlassian.com, while documentation for Atlassian's cloud apps was accessible through support.atlassian.com.

To our knowledge, there was no API which would enable fetching all Atlassian documentation, necessitating to rely on scraping as a means of obtaining Atlassian data. We modified Python-Sitemap, a script originally designed for generating sitemaps with Python, to crawl the Atlassian web-domain. Our customized version, instead of generating a sitemap, automatically downloaded and stored the `.html` files of each visited URL in the local file system. In total, approximately 65'000 `.html` files were downloaded.

A consequence of Atlassian documentation being spread across two subdomains with different user interfaces was that the document structure varied, making parsing of `.html` documents non-trivial. To exclusively obtain the core content of each article, a whitelist-approach was utilized, where a list of CSS selectors was defined and used to find tags which were subsequently extracted from each `.html` document using the jsoup library for parsing HTML documents with Java. Lastly, the extracted document data was structured and stored in `.json` format on the local file system before further processing.

4.1.2 Embedding and Storage

Atlassian and KSSO documents were embedded and subsequently stored in a PostgreSQL database, as shown in Figure 11. We tried three different embedding models. Two models were chosen

from the SentenceTransformers Python package: `average_word_embeddings_glove.6B.300d`, based on the GloVe algorithm described in section 2.4.3, and a transformer-based model, `all-MiniLM-L6-v2` [77]. Lastly, OpenAI's most performant embedding model at the time of development, `text-embedding-3-large`, was used to understand the performance of state-of-the-art embeddings in RAG applications.

The first SentenceTransformers model was `average_word_embeddings_glove.6B.300d`, an embedding model based on the GloVe embedding algorithm described in section 2.4.3. The model had 6 billion parameters and produced 300-dimensional vectors. It was unrestricted by sequence length, meaning that chunking was not required. According to SentenceTransformers' performance evaluation, the model was very fast relative to other models, being able to encode 34'000 sentences per second on a V100 GPU [77]. The speed was helpful for prototyping, given the large number of documents, as it enabled faster testing of the full prototype.

The second pre-trained SentenceTransformers model was the general-purpose `all-miniLM-L6-v2`. According to SentenceTransformers' evaluation, it yielded only slightly worse embedding-quality than their best-performing model at the time of development, while being about 5 times faster [77]. The model was trained on short sentences and paragraphs, and had a max sequence length limit of 256 tokens. Atlassian and KSSO articles would often exceed this, as illustrated by the document token count distribution in figure 12. Consequently, chunking was necessary: Documents exceeding the max sequence length were split into multiple sequences, each shorter than 256 tokens. Every sequence was embedded independently. The final embedding was the arithmetic mean of all sequence embeddings.

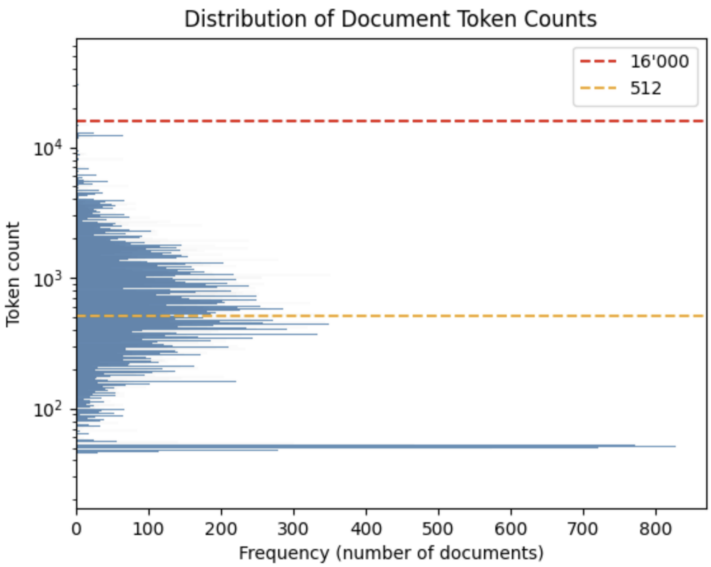

Figure 12: Distribution of document token counts, measured with the tiktoken Python library. The x-axis represents the number of documents, while the y-axis is the number of tokens in a document on a logarithmic scale. The distribution shows that few documents exceed 16'000 tokens, and the vast majority is under 512 tokens. A cluster of concise documents, each under 100 tokens in length, is visible.

Lastly, to evaluate the performance of state-of-the-art embeddings, OpenAI's most performant embedding model at the time of development, `text-embedding-3-large`, was used to embed all documents. The model also required document chunking, due to its input limit of 8191 tokens.[78]

The embeddings were stored as vectors in a PostgreSQL database using the pgvector extension [79], along with the original document text and a category-tag used to distinguish between Atlassian and KSSO documentation. The category tag would later allow us to specify the number of Atlassian- and KSSO documents to use in response generation.

4.1.3 Search

The search module, illustrated in figure 11, combined documents from three different sources: Atlassian documentation, KSSO documentation and Atlassian Community posts.

To search in Atlassian and KSSO documentation, the query (support ticket) was embedded using one of the models described in Section 4.1.2, to obtain a query embedding. Using the pgvector extension, a small set of documents with embeddings that were most similar to the query embedding were retrieved. The pgvector extension implements vector similarity search with the K nearest neighbors algorithm (KNN) described in Section 2.3.1, with the possibility of enhancing time complexity with approximate nearest neighbors (ANN) as described in Section 2.3.2. To achieve the most accurate results, KNN was used despite the higher time complexity.

Atlassian documentation contained many duplicate documents, because close copies of articles had been published on multiple pages for different system versions. If an article with existing duplicates was similar enough to the query, all of its duplicates would fill the top result list. Therefore, maximal marginal relevance (MMR), as described in Section 2.6.2 was implemented to increase the diversity of retrieved documents and successfully mitigate this issue. In an extended list of candidate documents, a small subset was selected, balancing relevance to the query and dissimilarity from the remaining documents in the result set.

Atlassian develops and manages many products which are hosted in different environments, so occasionally, system administrators find issues which are not covered by official articles. Besides contacting Atlassian support, they are able to receive help from community members and official staff on the Atlassian Community forum. Community posts are exposed through an API, which was used by the search module, such that a proportion of documents in the result list were retrieved from the forum.

4.1.4 Response Generation

The goal of the response generation module shown in figure 11 was to generate complete and accurate replies to support tickets using a combination of KSSO documentation, Atlassian documentation, and Atlassian Community posts. The model utilized was `gpt-4-0125-preview` from Azure OpenAI, the state of the art model available through Azure, with which KSSO had a data processing agreement. The agreement was important to ensure that proprietary data was not leaked.

The instruction prompt sent to the LLM included a set of top results from each data source, clearly

labeled with markdown headers for clarity. The LLM was instructed to utilize the provided documents to generate a coherent response to the support ticket. This response was the final output from the RAG prototype.

4.1.5 Findings from Exploratory Study

First and foremost, the exploratory study helped us observe the importance of information retrieval in RAG. Pre-trained LLMs are trained on large amounts of data, often from the public internet. However, LLMs like GPT-4, specifically `gpt-4-0125-preview` used in our testing, were not grounded enough to directly and consistently provide correct answers to questions which required highly specific pieces of information only using pre-trained data. For closed-source pre-trained LLMs, it is rarely documented, and therefore difficult to find which data sources were used in training. It is thus unreliable to depend on the LLM knowing the answer to a query. As observed by Kung and Peng [41] and previously described in Section 2.8.2, fine tuning seems to mostly contribute to the LLM learning superficial patterns like output format, and might therefore not be best suited for teaching the LLM new information. Thus, the relevant pieces of information should be delivered to the LLM in the instruction prompt, and must therefore be retrieved by the IR system of the RAG application. Therefore, the quality of a RAG application largely depends on the quality of the underlying IR system.

Having observed the importance of IR in RAG applications, it became evident that vector similarity search with kNN was insufficient to retrieve documents that were most relevant for answering the query. The system was tested with different embeddings. Empirically, results visibly improved after changing from GloVe-based to transformer-based embeddings. However, changing from `all-MiniLM-L6-v2` to OpenAI's `text-embedding-3-large` had less of an impact, and still fell short of reliably retrieving relevant documents. Interestingly, after implementing MMR as described in Section 2.6.2, a relevant document could often be found within the top-10 results. However, sending 10 documents to an LLM would often exceed its context limit, necessitating to set k to a much lower number, and thus sending only partially relevant or non-relevant documents to the LLM, and consequently having it fail to respond to the support ticket. Having a human evaluator look through the first-pass result set and point out which article is relevant would likely mitigate this issue. As a result, the idea of utilizing relevance feedback in the IR system was born, and motivated the first research question of this thesis.

The LLM was grounded enough to follow the provided instructions, which stated that it should solely rely on the provided documents, limiting the amount of hallucination. However, being constrained to a list of documents with a considerable fraction of these being non-relevant required the LLM to successfully filter out the relevant information. Based on testing and observation, it was clear that a single LLM fell short in this task, and tended to combine both relevant and non-relevant information into a single response. Having the LLM reflect on the answer would naturally result in the final answer including these reflections, so utilizing techniques such as chain-of-thought in an attempt to enhance the LLM's reflective thinking was unsuitable with a single agent. Working in a setting where the system's response quality is much more important than its time efficiency, an opportunity presented itself to evaluate different multi-agent patterns and evaluating their capabilities of formulating a logical response while filtering out non-relevant information. This motivated

the second research question of this thesis.

Applicability to Other Customer Support Settings KSSO gave us a good starting point for working with a dataset that represents what one could find in customer support, yet customer support varies greatly from company to company. Therefore, it is important to specify what defines support in the context of KSSO, and how this can differ from other companies.

First and foremost, KSSO handles a relatively low volume of support cases. This implies that time and resources are not a problem to the same extent as they would be in a company specializing in customer support, which might receive several hundred or thousand inquiries per hour. The use of language models in relevance feedback or multi-agent patterns may be less applicable in high-volume situations, where time is of the essence.

Secondly, KSSO's customer support employees require possess deep technical domain knowledge. Such a technical domain increases the difficulty of successfully resolving support tickets. Other companies might receive multiple categories of cases, with first and second line support, where second line cases can vary in difficulty, some of which might be easier to solve with a language model. It is not unlikely that the prototype described in this section would see greater success in specific parts of second line support in another company.

Although our research questions, methodology and experiments are all motivated by findings from the exploratory study which was based on KSSO's support operations, Section 6 will attempt to apply our findings to a broader customer support setting.

4.2 Factorial Design Experiment for RQ1 - LLMs for Query Expansion

In a factorial design experiment, the independent variables are the elements that are manipulated to observe their effect on the dependent variables. These independent variables can include different models, techniques, or conditions under which the models operate. Each independent variable can have different levels, which are the specific values or settings of the variable. The dependent variables are the measurable outcomes used to evaluate the performance of the models, such as accuracy, recall, or other relevant metrics. [75]

The experimental units in a factorial design are systematically subjected to different levels of the independent variables. To ensure the reliability and validity of the results, experiments should be replicated multiple times, on multiple or larger datasets. [75]

The use of a factorial design for experiments associated with RQ1 is justified by the need to analyze the interactions between multiple independent variables, such as different language models, prompting techniques, and datasets, and their impact on performance metrics. Factorial design allows for simultaneous examination of these variables, helping us draw conclusions about how various factors influence the effectiveness of language models in relevance feedback.

4.2.1 Technology

This section provides a detailed description of libraries and frameworks used to conduct experiments for RQ1.

Terrier is an open source search engine written in Java. It supports indexing and retrieval of document collections, exposing accessible data structures for various indexes for the document collection [80]. Specifically, Terrier version 5.5 was utilized in this thesis.

The weighting model used to assign scores to retrieved documents can also be specified. Of all available weighting models, TF-IDF, described in section 2.4.2, was used in the experiments for this thesis. [81]

Terrier readily implements a number of query expansion algorithms, such as B01, KL query expansion and the Rocchio algorithm, described in sections 2.5.6, 2.5.5 and 2.5.4 respectively. By default, Terrier chooses 10 terms to expand the query, and finds these terms in the top-3 retrieved documents. [81]

Relevance feedback is supported through the FeedbackSelector class, which is used to select feedback documents. The default implementation is the PseudoRelevanceFeedbackSelector class, which selects the top k (default $k = 3$) documents from the result set. [82]

Terrier greatly facilitates running IR experiments. On runtime, a topic (query) file can be specified, containing a set of queries in various formats, with built-in support for the standard TREC format and the tab-separated value (.tsv) format. Each run produces two files: a .res file which contains the final ranked result list of documents for each query in the topics file, and a .res.settings file which includes metadata about the run, including timestamps for when the run was started and finished, and the Terrier configuration used at runtime. [83]

Finally, Terrier integrates the jtreceval library [84], which is a Java wrapper around the trec_eval evaluation software [85] of the National Institute of Standards and Technology (NIST).

PyTerrier is a platform for conducting IR experiments in Python, based on Terrier. PyTerrier's datasets module grants access to standard test collections. Each defined dataset can easily be downloaded and used for experiments. [86] This module was utilized to download an index of the MSMARCO2020 (trec-deep-learning) document collection [87].

Similarly to Terrier, PyTerrier integrates pytrec_eval, which is a Python wrapper for the trec_eval software [88].

Ollama Ollama is a tool designed to facilitate deployment and management of language models on local machines, eliminating the need for cloud-based resources and enhancing data privacy. It

was used to download and run the llama3:instruct model for conducting the experiments in RQ1. The Ollama4j library was used to facilitate sending requests to Ollama models. [89, 90]

4.2.2 Planned Experiments

The goal of RQ1 was to evaluate the performance of pre-trained large language models tasked with substituting a user in a relevance feedback loop. Following the experiment design methodology, we determined these independent variables:

1. **Language model**: Different language models would behave differently and provide varying relevance judgments based on pre-trained data. For this reason, experiments were run with different models of varying sizes. The levels of this independent variable were:
   - **GPT-4o-2024-05-13**: This model was chosen because it represents the latest advancements in large-scale language modeling, with state-of-the-art performance. It was included to evaluate the upper limits of current model capabilities.
   - **GPT-3.5-turbo-0125**: An older OpenAI model was selected to be compared with OpenAI's newest model as of May 2024, to highlight the advancement of LLMs and help to understand the potential of language models in future relevance feedback applications.
   - **LLaMA3:instruct 8b**: This smaller 4.7GB model was included to evaluate relevance feedback capabilities of state-of-the-art small-scale models. The model is designed to run on less powerful devices, such as laptops. It makes self-hosting a viable option, and thus mitigates privacy risks related to processing sensitive information, which is commonly a requirement for businesses like KSSO, aiming to protect their proprietary information. The model was hosted with Ollama, which employs 4-bit quantization of model parameters, meaning that the precision of model parameters was reduced to 4 bits. This reduces the model size and speeds up inference times, but can negatively affect performance.
2. **Prompting technique**: Language models predict words in a sequence, and thus heavily rely on the surrounding context. Since prompts would result in different context before each relevance judgment, prompting technique should have a significant impact on a model's relevance judgment capabilities. Experiments were conducted in accordance with factorial design with three levels: zero-shot, few-shot and chain-of-thought prompting, all of which are described in section 2.8.1.

4.2.3 Baselines

RQ1 revolves around document selection and does not concern itself with how the feedback documents are utilized to expand the original query. Therefore, the choice of underlying query expansion algorithm (Rocchio, Bo1, KL or RM3) was insignificant, as it would remain constant throughout all experiments. To ensure consistency, all experiments were run with the Rocchio algorithm with the β parameter set to Terrier's default setting of 0.4.

In baseline runs, the feedback documents were selected by Terrier's PseudoRelevanceFeedbackSelector class, which implements pseudo-relevance feedback as described in Section 2.5.3. The class selects the top-3 documents for query expansion. The number of extraction terms was set to 10, which is Terrier's default setting. This would be the baseline for all experiments.

4.2.4 Evaluation Metrics

To evaluate the performance of LLMs in providing relevance feedback, the IR evaluation metrics used were $NDCG@k$ described in Section 2.7.7, and MRR described in Section 2.7.8, with $k \in \{1, 2, 3, 4, 5, 10\}$. Additional metrics included the average number of generated output tokens, the fraction of documents deemed relevant by the LLM, and the fraction of correct relevance judgments. These metrics constitute the dependent variables for RQ1, in accordance with the factorial experiment design methodology. Below is a brief description of each evaluation metric, and a justification of its use in the context of aiming to improve the search capabilities of an IR system which is part of a RAG solution aiming to aid in technical support ticket resolution.

- **NDCG@k:** P@k, described in Section 2.7.1, can be low for queries for which few documents are deemed relevant. Likewise, MAP, described in section 2.7.5, can be unreasonably low if a relevant document appears very far down in the ranking list. NDCG@k is a normalized precision-based metric which remains unaffected by the number of relevant documents. By being able to specify values of k, it ignores documents outside of the top-k list. Therefore, NDCG makes it easier to interpret the precision of an IR system.
- **MRR:** MRR is highest when the first document in the result list is relevant. It is a single-valued metric which makes it easy to interpret. Similarly to NDCG, MRR is a precision-based metric, and is therefore crucial in the context of RAG.
- **Output Token Count:** The average number of generated output tokens was measured to evaluate the efficiency of different combinations of language models and prompt methods.
- **Documents Deemed Relevant:** The fraction of documents deemed relevant by the LLM was measured, enabling us to observe bias, if present, in the relevance assessments of each LLM and prompt method.
- **Correctness:** The fraction of correct relevance judgments was measured in order to understand the similarity in the assessments of human and LLM evaluators.

4.2.5 Datasets

Following the recommendation of Montgomery [75] to replicate experiments multiple times on multiple or larger datasets to ensure reliability and validity of the results, two standard datasets were used to evaluate the performance of relevance feedback with large language models: the NPL (VASWANI) collection [91], and the MSMARCO document ranking dataset from the TREC 2020 deep learning track [87]. The queries found in these datasets would constitute the experimental

units for RQ1, in accordance with the factorial design methodology. Each dataset is described in more detail below.

MSMARCO Deep Learning 2020 The MSMARCO dataset consists of a corpus with 3'213'835 documents. Queries are split into training, dev and evaluation. For training, the dataset contains 367'013 queries and 384'597 human-generated relevance judgments. 45 queries and 9'098 query relevance judgments were used for evaluation by the 2020 TREC leaderboards. A relevance judgment is a triple of query-id, document-id and a relevance label. Since multiple documents may be relevant for a query, the number of relevance judgments exceeds the number of queries.[87]

Since no parameter tuning or training was required for the experiments conducted in this thesis, only the 45 evaluation queries were utilized. The dataset was retrieved using PyTerrier's datasets module [86]. An index was generated and stored locally.

Language models require document contents to make relevance judgments. To our knowledge, document text for the MSMARCO dataset was not readily available in any of the generated indexes. For this reason, the corpus had to be downloaded, and the textual content of each document was read and inserted into an SQLite database. SQLite was selected because it is open-source, simple to set up and is maintained on disk using B-trees, making it capable of executing a search with low ($O(\log n)$) time complexity [92].

Relevance judgments in the MSMARCO dataset are on a four-point scale ranging from irrelevant (0) to perfectly relevant (3). The dataset's overview document ([87]) states that for a binary judgment scale, judgment levels 0 should be mapped to irrelevant, whereas levels 1, 2 and 3 should be mapped to relevant. This mapping was performed as a preprocessing step on the file containing relevance judgments.

The dataset was selected for this thesis because the documents resemble typical textual data which can be obtained through basic web-scraping, as was done with Atlassian documentation in the exploratory study described in section 4.1. A large number of available documents increases the likelihood that the top-k documents have high similarity, making it more difficult to differentiate between relevant and non-relevant documents. The high document volume creates a scenario similar to what was found in the exploratory study, where large amounts of documentation are available for retrieval.

VASWANI The VASWANI collection is a small corpus of 11'429 scientific abstracts, 93 queries, and 2'083 query relevance judgments. Each relevance judgment contains a `queryId`, `documentId` and a relevance label. All relevance labels are 1, so the existence of a `(queryId, documentId)` pair in the relevance judgment file means that a document is relevant to the associated query.

The VASWANI dataset was selected due to the nature of its corpus, which contains highly specific, scientific abstracts. The documents are short and precise, so the similarities between documents should be smaller. The document volume is also smaller, relative to the MSMARCO dataset. A faulty relevance judgment could therefore expand the query such that fewer or no relevant doc-

uments are extracted. Given shorter queries and documents, interpreting the user's information need also becomes more difficult for the LLMs, increasing the challenge of generating sound relevance judgments.

4.2.6 Experimental Setup

Terrier implements a designated class for selecting feedback documents in query expansion, namely the FeedbackSelector class. A new class called LLMFeedbackSelector was defined, extending FeedbackSelector and overriding the getFeedbackDocuments method which contained the core logic for document selection.

The query text was retrieved along with the top ranking document numbers (`DOCNO`) from the first-pass result set. The result set did not include document texts. Therefore, prior to running the experiments, the document corpora were downloaded for each dataset, and `(DOCNO, DOCTEXT)` pairs were stored in an SQLite database. At runtime, for each document in the result set, its document number was used to find the associated document text.

For each query, 5 documents would be assessed for relevance. The number of documents was motivated by the recommendations of Manning et al. [6, Chapter 9.1.5] as mentioned in Section 2.5.2.

For each document, an instruction prompt was prepared. Each prompt consisted of three parts in the following order:

1. Document text
2. Query
3. Prompt method dependent instructions

This ordering was motivated by the findings of Islam et al. [38], as described in Section 2.8.2, where structuring the prompt with the context presented first resulted in significant performance improvements.

The experiment employed three different prompt methods: Chain-of-Thought (CoT), zero-shot (0-shot), and one-shot (1-shot). Each method had its own specific set of instructions for the AI assistant. The choice of which method to use was determined at runtime. The following prompts were defined for each method:

```
### CoT:
You have been given a document text and a query. First, write a
short text about the relevance of the document for the query.
Think about user information needs. Remember that the document
may be irrelevant even if it contains relevant keywords.
Finally, end with 'Final Answer: yes/no' (yes if relevant, no
```

```
otherwise)

### 0-shot:
You have been given a document text and a query. Is this
document relevant to the query? Answer with 'Final Answer: yes/
no' (yes if relevant, no otherwise)

### 1-shot (using an example text from Wikipedia):
You have been given a document text and a query. Is this
document relevant to the query? Answer with 'yes' or 'no' (yes
if relevant, no otherwise)
Example:
123: The cat (Felis catus), commonly referred to as the domestic
cat or house cat, is a small domesticated carnivorous mammal.
Query: What is a cat?
Response: yes
```

The prompts were designed through multiple iterations of adjustments and empirical testing on the llama3:instruct model. Due to its small size, this model was most difficult to ground. However, after a few iterations, it generated consistent responses for all prompt methods. Tests with `gpt-3.5-turbo-0125` and `gpt-4o-2024-05-13` confirmed that the larger models also followed the instructions successfully using the same prompts.

Document texts were clamped to a maximum of 50'000 characters to prevent exceeding the context size limit of the `gpt-3.5-turbo-0125` model, as this would result in an error thrown by OpenAI's API. This only affected a handful of documents from the MSMARCO dataset. Considering that the same retrieval and relevance feedback algorithms were utilized in all experiments, the same set of documents was judged in each run, and the same documents were clamped, meaning that no run would have an unfair disadvantage.

Once a relevance judgment was generated, its last line was extracted, and a regex pattern was applied to parse the relevance judgment into a boolean value which determined whether the document was deemed relevant. The pattern would find a match if the last line contained the substring "yes", regardless of case or surrounding punctuation and words.

Each run had a designated SQLite database for logging of relevant statistics. For each relevance judgment, the class logged the time spent generating a response, the LLM input and output, the original query, the ID of the document being judged, and a boolean denoting the relevance judgment made by the LLM, as was parsed by the aforementioned regex pattern. This data would later be used for evaluation.

Finally, the method generated a list of FeedbackDocument objects for each document that was deemed relevant, as was required by Terrier to utilize the relevance judgments for query expansion.

4.2.7 Evaluation

The `pytrec_eval` tool was used to parse the result files generated by the conducted experiments and calculate the metrics described in Section 4.2.4.

As mentioned in Section 4.2.6, each run had a designated SQLite database for logging various statistics. The token counts of input and output texts were calculated using the tiktoken Python package [93]. Ground-truth query relevances were used to control the correctness of the relevance judgments made by an LLM. Based on this, a confusion matrix was generated, and the judgment accuracy was calculated for each combination of prompt method and model.

4.3 Single-Factor Experiment for RQ2 - LLM Communication Patterns

In a single-factor experiment, the independent variable represents the single element being tested. The dependent variables are the outcomes measured to evaluate the performance under different conditions set by the independent variable. The experimental units in this context are the subjects or items on which the experiments are conducted. By varying the independent variable while maintaining consistency in other aspects of the experiment, such as the dataset and evaluation metrics, the single-factor experiment enables a direct comparison. [75]

Research question 2 aims to investigate how different multi-agent LLM communication patterns compare in a Retrieval-Augmented Generation (RAG) system that is designed to assist or potentially resolve complex technical product support tickets. As investigated in Section 3.5, multi-agent conversation patterns refer to the different structures that LLMs can interact with each other.

These patterns can significantly impact the potential for LLMs to solve complex problems, such as the ability to filter out large portions of potentially relevant and irrelevant information to find a possible solution to a problem. The experiments conducted for RQ2 are designed to simulate a realistic technical product support scenario, where an LLM in a multi-agent setup must reason over domain-specific complex information that is not part of its training data.

By comparing different patterns of reasoning and communicating with other agents, this experiment aims to provide insight into which communication pattern leads to more effective problem solving.

4.3.1 Technology

This section provides a detailed description of libraries and frameworks used to conduct experiments for RQ2.

Ollama Similarly to RQ1, Ollama was used to run local language models, along with the Ollama Python library to facilitate sending requests to the models. [89, 94]

LangGraph LangGraph, is a framework for development of multi-agent applications. It is built by and as a superset of the most popular LLM framework of 2024, LangChain. It gave us the advantage of standardizing our experiments across different language models, while allowing the flexibility to customize every step of the experiment, crucial for evaluating runs and investigating potential flaws in our research. [95]

LangSmith With built in support in the LangChain framework, LangSmith facilitated free out-of-the-box monitoring for all LangChain runs. LangSmith describes itself as "*a unified DevOps platform for developing, collaborating, testing, deploying, and monitoring LLM applications*" [96]. LangSmith allowed us to easily retrieve all meta- and trace-data related to each LLM run and pattern in the experiment.

4.3.2 Experimental Setup

To simulate a complex technical product support scenario where potentially proprietary company data is necessary to solve customer queries, we designed experiments aiming to challenge the LLMs' abilities to comprehend, retrieve and reason over complex information available to it through simulated Retrieval-Augmented Generation (RAG), described in Section 2.8.2. The information may or may not be relevant, similar to a database documentation search where you are looking for the answer to a given query that might be buried and hard to find.

To achieve the goal of simulating a complex technical product support scenario, where domain-specific and proprietary information, potentially relevant or irrelevant, would be required, we initially considered using a dataset provided by the company (KSSO). However, the KSSO dataset investigated in Section 4.1 proved too complex to customize and utilize effectively for our experiments. As a viable alternative, we selected a variant of the BioASQ-QA dataset, curated for evaluating RAG applications.

Krithara et al. [97] developed the BioASQ-QA dataset, a biomedical benchmark for question answering. For every question, it includes related documents and snippets, as well as the ideal (summary) answer. The related documents and snippets generally compare to the potentially large volume of domain-specific information that one might find in a complex technical product support scenario. The dataset itself is designed to reflect the real information needs of biomedical experts, making it a challenging and realistic test scenario for evaluating different multi-agent LLM communication patterns.

To properly evaluate a multi-agent system's ability to also filter out potentially irrelevant domain-specific and domain-related documents, we constructed a new dataset containing subset of the questions of the BioASQ-QA dataset, where we added irrelevant documents as noise. Specifically, we introduced irrelevant documents such that the percentage of irrelevant documents was 3x the amount of relevant documents, with a minimum of six irrelevant documents per query. Whereas the original BioASQ-QA dataset consisted of an average 6.77 relevant documents for each question, our custom dataset resulted in an average of 10.84 associated documents per question, both relevant and irrelevant. To further ensure a somewhat realistic scenario of a human support agent

filtering through potentially relevant documents to find the answer the the customer query, the ratio of irrelevant to relevant documents is verified to be 79.08%, approximately 4:1, with an average of 2.27 relevant passages and 8.57 irrelevant passages per question.

The custom BioASQ-QA dataset curated for this experiment resulted in 1'172 questions, ideal answers and related documents, and is published to Hugging Face at the dataset repository Xtrah/filtered-rag-mini-bioasq-80-20 [98]. The dataset's individual queries are our experimental units in accordance with the single factor experiment method, and they are used in the experiments to evaluate the different communication patterns. The average length of the ideal responses in the dataset is 231 characters, with an average word count of 37 and an average sentence count of 1.79. These metrics provide a benchmark for the complexity and conciseness expected in the generated responses, reflecting the realistic requirements of technical product support scenarios.

Due to the concise answers of the dataset, but still enabling for chain of thought or communication between LLMs, the system prompts will instruct each pattern to end its reasoning with a final answer. Only the final answer will be evaluated against the ideal response. This is important when evaluating the performance of different communication patterns, because it ensures that the intermediate steps and reasoning processes do not directly influence the evaluation metrics. By focusing solely on the final answer, we can more accurately assess the effectiveness of the pattern and its ability to synthesize and retrieve relevant information in a coherent and concise manner.

4.3.3 Patterns

Different popular LLM patterns for improved reasoning was investigated in Section 3.5. For this experiment, the patterns described below were chosen in an attempt to balance simplicity and complexity. While more complex patterns were considered, such as debating multi-agent collaboration, it was ultimately deemed too complicated to implement due to the numerous possible variations and the advanced coordination required among agents. The chosen patterns represent a range of straightforward to moderately advanced methods, providing a fair overview and comparison without overwhelming the experiment with excessive complexity.

In accordance with the Single-Factor Experiment methodology, the LLM communication patterns are the single independent variable of this experiment. To properly investigate the effect of the different communication patterns on the quality of generated answers, given a number of potentially relevant documents, the rest of the experiment remain consistent. Compared to the experiment for research question 1, the intention of the experiment is not to judge the capabilities of any one model, but to evaluate how different patterns can increase the capabilities to reason over potentially relevant or irrelevant information. This approach, with the patterns as the only independent variable, enables a direct comparison of the patterns.

The respective system prompts for each pattern is constructed using the state-of-the-art LLM GPT-4o by OpenAI [99]. Similarly to the experiment for research question 1, the prompt is constructed with the findings of Islam et al. [38] in mind, described in Section 2.8.2, where the context is provided before the user query at hand. Additionally, we keep the findings of Section 3.4 in mind,

constructing our prompt instructing the LLM to only use the retrieved documents to answer the question, grounding it in verifiable information.

There is one important difference compared to the traditional Tool-use pattern, where a RAG system constructs the search query on its own compared to our experiment. In our experiment, each question's related documents, which average 4:1 relevant to irrelevant ratio, are provided manually in each prompt to ensure a fair comparison of each pattern's performance and efficiency metrics. This particularly affects our most simple pattern, Tool-use, as it effectively becomes a zero-shot baseline.

Tool-use In our most simple pattern, the LLM is provided with the set of documents and the question to answer without any additional instructions to enhance reasoning. Our Tool-use pattern is illustrated in Figure 13. The following system prompt was constructed for our pseudo Tool-use pattern:

```
Use retrieved documents to answer the question.
Ignore irrelevant documents.
Do not repeat information.
If you can't find an answer, say so.
Only give the direct answer to the question.
Give your final answer at the end starting with "FINAL ANSWER: <answer>".

{documents}

Answer the question '{question_text}' using the documents above.
Only provide a final, concise answer.
Give your final answer at the end starting with "FINAL ANSWER: <answer>".
```

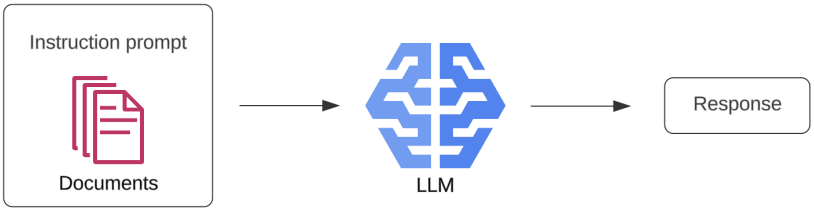

Figure 13: Tool-use and Chain-of-Thought pattern implementation

Chain-of-Thought The LLM is provided with the set of documents and the question while instructed to articulate its reasoning process step-by-step in a Chain-of-Thought, as described in Section 2.8.1. Otherwise our Chain-of-Thought pattern is identical to the Tool-use pattern as depicted in Figure 13, with the following instructions for the LLM as system prompt:

```
Use retrieved documents to answer the question.
```

```
Ignore irrelevant documents.
Do not repeat information.
If you can't find an answer, say so.
Only give the direct answer to the question.
Take a deep breath and work on this problem step-by-step.
Give your final answer at the end starting with "FINAL ANSWER: <answer>".

{documents}

Answer the question '{question_text}' using the documents above.
Take a deep breath and work on this problem step-by-step before
providing a final, concise answer.
Give your final answer at the end starting with "FINAL ANSWER: <answer>".
```

Compared to Tool-use, the only addition to the system prompt is to work on the problem step-by-step before providing a final answer.

Reflection An LLM, *initial responder*, identical to Tool-use, is provided with the set of documents and the question to answer. After answering, the documents, question and the former answer is given to a *revisor* LLM to provide feedback:

```
You are an excellent feedback provider for a very clumsy AI. You provide
feedback only, no updated answer.
If the response is that the query is unanswerable, be extremely thorough
and check the documents to ensure the answer is not present.
If potential keywords related to, or the answer itself, is present,
provide guidance on how to find the answer.
Ensure the feedback is clear and specific, addressing any areas where
the response could be more concise.
Focus on identifying and eliminating redundant information, suggesting
more direct phrasing, and maintaining the response's clarity and effectiveness.
Additionally, verify the accuracy of the response. If there are any inaccuracies,
point them out clearly and suggest corrections to ensure the answer is correct.

{documents}

{question}

{generated answer}

Give feedback on the generated answer. Provide feedback only, no updated answer.
```

In the next step of the Reflection pattern, the feedback is provided back to an LLM *responder* to improve the initial answer:

```
Use retrieved documents to answer the question.
Ignore irrelevant documents.
Do not repeat information.
If you can't find an answer, say so.
Only give the direct answer to the question.
Take feedback into account.
Give your final answer at the end starting with "FINAL ANSWER: <answer>".

{documents}

{question}

{feedback}

Incorporate the feedback into a new answer. Respond only with the new,
revised answer.
Give your final answer at the end starting with "FINAL ANSWER: <answer>".
```

This process is repeated twice before the final answer is submitted, making the reflection process effectively iterate and give feedback on its answer twice before submitting a final answer to be evaluated. The Reflection pattern implementation is illustrated in Figure 14.

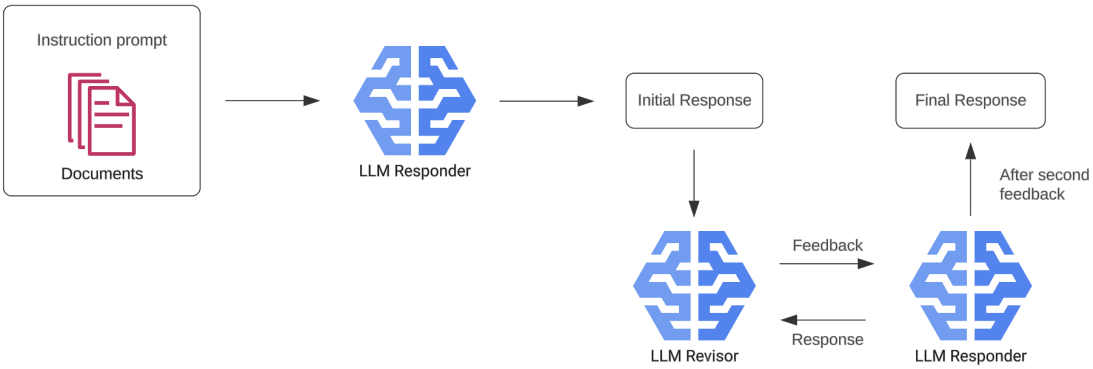

Figure 14: Reflection pattern implementation

Planning An LLM *planner* is provided with a set of documents and the question, to plan up to three individual tasks with the ultimate goal of generating a satisfying response to the question. The system prompt for the planner is as follows:

```
Create a plan for how to generate a satisfying answer to a question that
you will be given later. The plan should consist of a number of tasks
(maximum 3 tasks). Each task should be solvable using the available
documents. Provide the tasks in an array format, where each task is a string.
Return the response only in the array format and with no other text.
The format should look like this:
```

```
["task1", "task2", "task3"]

{documents}

{question}

Create a plan for this with max 3 tasks.
Respond with an array on a format like this: ["task1", "task2", ...]
where each task is a string.
ONLY respond with the array, nothing else. No other text, no formatting.
```

Then, another LLM, *solver*, is given the set of documents with one task at a time to solve individually:

```
Please solve the task using all available information.
Do not repeat any information.
If you cannot find the answer, or the information is insufficient,
please state so.

{documents}

{task}

Solve the task using all available information.
```

Finally after the tasks are attempted solved, an LLM *revisor* combines the answer to all tasks without the set of documents to attempt to answer the initial question.

```
Please review the solutions to the plan and compose a final answer to
the original question.
Give your final answer at the end starting with "FINAL ANSWER: <answer>".

{question}

{task answer(s)}

Give your final answer at the end starting with "FINAL ANSWER: <answer>".
Given the solutions above, compose a final, concise answer to the
original question: {question}
```

Figure 15 shows how the Planning pattern was implemented.

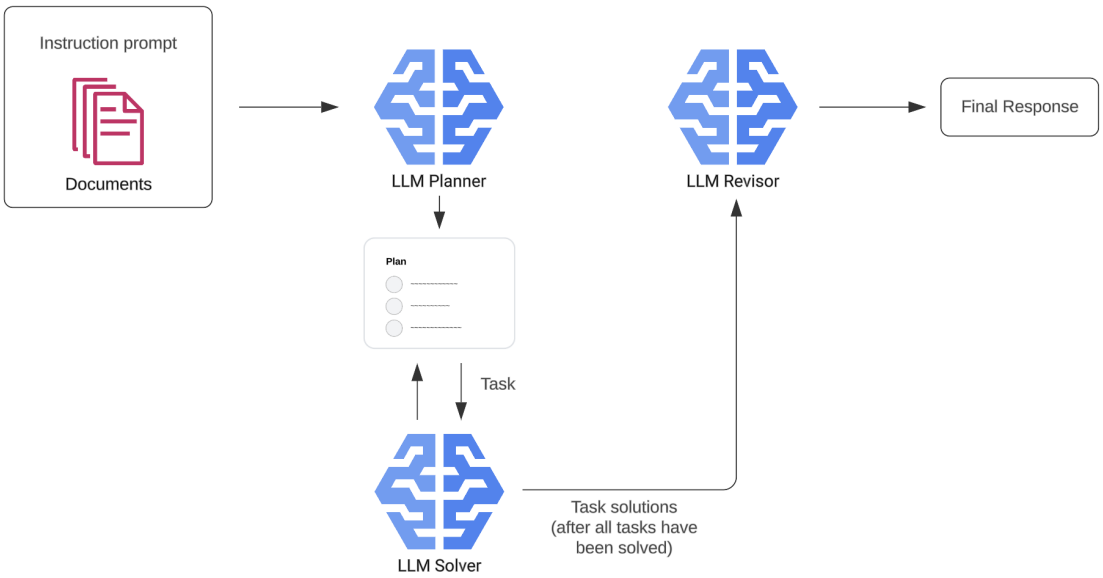

Figure 15: Planning pattern implementation

4.3.4 Baselines

The baseline communication pattern for our LLM experiment is the simplest communication pattern "Tool-use". In our experiment, all patterns were given the same set of documents to reason upon, with the goal to simulate a normal support agent having access to a search tool to find potentially relevant data to the customer query.

4.3.5 Evaluation Metrics

We investigated potential metrics to evaluate LLM output in Section 3.6. To evaluate the performance of LLMs in providing correct and relevant answers in our experiment, the following metrics were chosen as our dependent variables in accordance with single factor experimental design. In each paragraph below, their individual relevance and potential insight is described in the context of our experiment.

ROUGE-1, ROUGE-L Recall-based metrics that measure the overlap of unigrams, bigrams, and longest common subsequences between the generated response and reference responses. In our context, ROUGE metrics are used to evaluate the coverage of key terms and phrases present in the ideal responses. ROUGE-1 focuses on single word matches, and ROUGE-L on the longest matching sequence of words. These metrics provide insight into how well the LLM is capturing the essential content and keywords from the provided documents.

METEOR Combines precision, recall, and a measure of word alignment between the generated response and reference (ideal) response. METEOR is useful in our experiment for assessing not

only the correctness of the responses but also their fluency and alignment with the ideal responses. It takes into account synonymy, stemming, and paraphrasing, providing a more nuanced view of the LLM communication pattern's performance in generating human-like and contextually appropriate answers.

LLM-judged correctness Based on what we learned in Section 3.6.3, we use a human-like evaluator where another independent LLM judges whether the ideal answer from the dataset and the generated answer from the LLM include the same key elements, indicating a boolean TRUE or FALSE. For our experiment, this metric simulates a human-like evaluator's perspective, providing a binary judgment on the correctness of the response. This metric helps determine if the LLMs not only retrieve information accurately but also understand it in the context given and apply it correctly to answer complex queries. This metric is especially significant in scenarios where understanding and analysis of information are required, as it reflects the practical usability of the LLM-generated responses in real-world complex technical support tasks. By using this approach, we can gain insights into how well the communication patterns perform in terms of both accuracy and contextual relevance in the final response.

5 Results

This chapter presents the results obtained from the experiments conducted for both research questions, as outlined in the methodology section. The results are organized into sections that align with the research questions and sub-questions. For each experiment, relevant metrics are reported, while observations based on the results will be discussed in Section 6.

5.1 RQ1: Evaluating LLMs for Query Expansion

This section presents the results of the experiments conducted to evaluate the performance of large language models (LLMs) in providing relevance feedback for query expansion.

Table 3 shows the mean reciprocal rank (MRR) and percentwise difference in MRR from the baseline pseudo-relevance feedback algorithm. The table shows results for all combinations of dataset, prompt method and language model. Positive values in the percentwise difference indicate that the combination of independent variables has provided more accurate search results than the naive PRF algorithm, with the first relevant document tending to appear higher in the result list.

Figure 16 consists of three plots, each showing the percentwise difference in NDCG@k values for different values of k for different language models on the MSMARCO dataset. Each plot is for a different prompt method. From left to right, the plots show results for 0-shot, 1-shot and Chain-of-Thought (CoT) prompting. The baseline PRF algorithm can be imagined as a line along the x-axis for $y = 0$. Positive values indicate that the relevance feedback with LLMs gives better search results than PRF, with a higher fraction of relevant documents in the top-k result list. The results

Table 3: Mean reciprocal rank (MRR) and percentwise difference in MRR from baseline (% Difference) for llama3-instruct (llama3) with 4-bit quantization, gpt-3.5-turbo-0125 (gpt-3.5-turbo) and gpt-4o-2024-05-13 (gpt-4o) on the VASWANI and MSMARCO datasets. Best relative result for each prompt method is highlighted with bold text for both datasets. Best result across all prompts is highlighted with an underline for both datasets.

		VASWANI		MSMARCO	
Prompt	**Model**	**MRR**	**% Difference**	**MRR**	**% Difference**
Baseline	-	0.675670	0	0.799841	0
CoT	llama3	**<u>0.732717</u>**	**<u>7.887</u>**	0.833686	4.231
	gpt-3.5-turbo	0.722029	6.313	0.848934	6.138
	gpt-4o	0.730530	7.565	**<u>0.859630</u>**	**<u>7.475</u>**
0-shot	llama3	0.686415	1.069	0.842761	5.366
	gpt-3.5-turbo	0.726970	7.041	0.827946	3.514
	gpt-4o	**0.728790**	**7.309**	**0.859428**	**7.450**
1-shot	llama3	0.675670	-0.513	0.842963	5.391
	gpt-3.5-turbo	0.701242	3.252	0.834921	4.386
	gpt-4o	**0.719624**	**5.959**	**0.850168**	**6.292**

shows that all combinations of models and prompt methods tend to perform better than the PRF algorithm.

Figure 17 shows the same category of results as Figure 16, but for the VASWANI dataset. Here, it is evident that the Llama3 model does poorly, relative to the other models, with 0-shot and 1-shot prompt methods, even dropping below the baseline for NDCG@1 with 1-shot prompting.

Figure 18 shows the average output lengths with standard deviation of different combinations of language models and prompt methods when making relevance judgments. From the results, it is clear that the CoT prompt method generated the highest number of tokens. Llama3 seems to have generated longer responses than other models with the 1-shot technique on the MSMARCO dataset.

Figure 19 shows the fraction of documents deemed relevant by each model with each prompt method. It seems like fewer documents were deemed as relevant using the CoT technique. Generally, models seem to evaluate more documents as non-relevant from the VASWANI dataset, compared to the MSMARCO dataset.

Finally, Figure 20 shows the fraction of correct relevance judgments made by each model and prompt method. Models tend to be quite consistent, with slight variations in correctness.

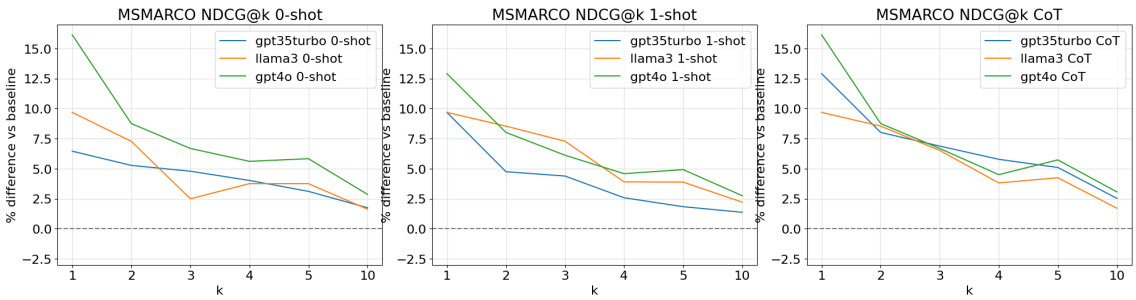

Figure 16: Percentwise difference (from baseline PRF algorithm) in NDCG@k for different values of k with different language models and prompting methods on the MSMARCO dataset.

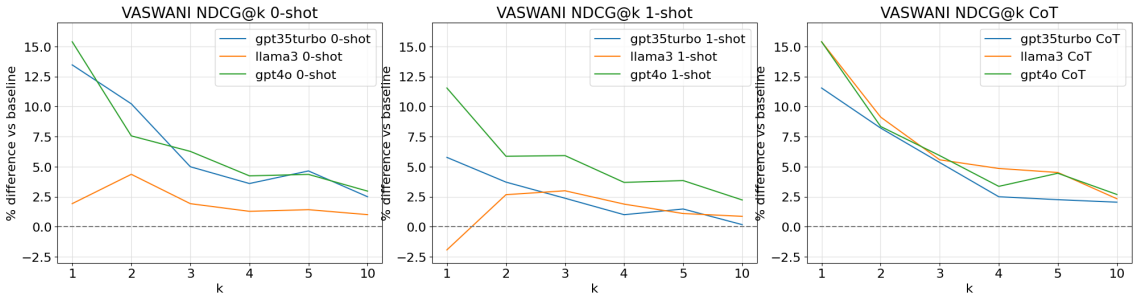

Figure 17: Percentwise difference (from baseline PRF algorithm) in NDCG@k for different values of k with different language models and prompting methods on the VASWANI dataset.

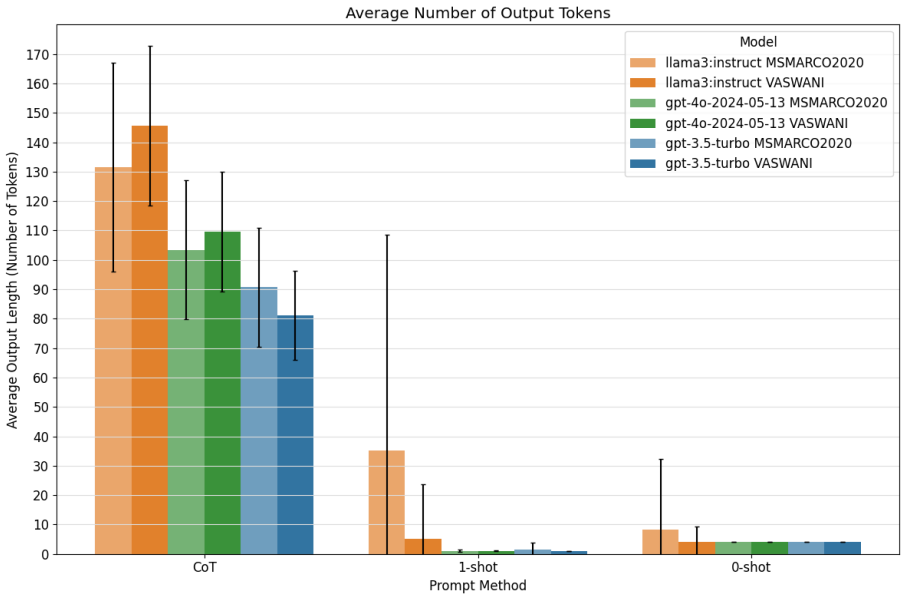

Figure 18: Average number of output tokens generated by language models on different datasets, categorized by prompt method. Standard deviation is shown with black error bars.

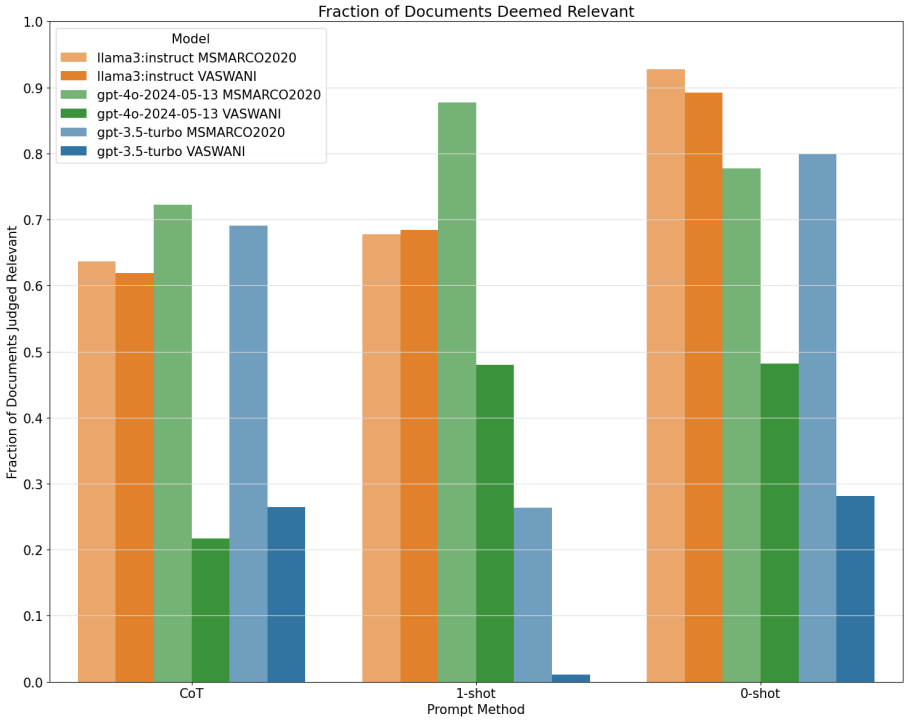

Figure 19: Fraction of documents deemed 'relevant' by each model for the MSMARCO and VASWANI datasets, categorized by prompting method.

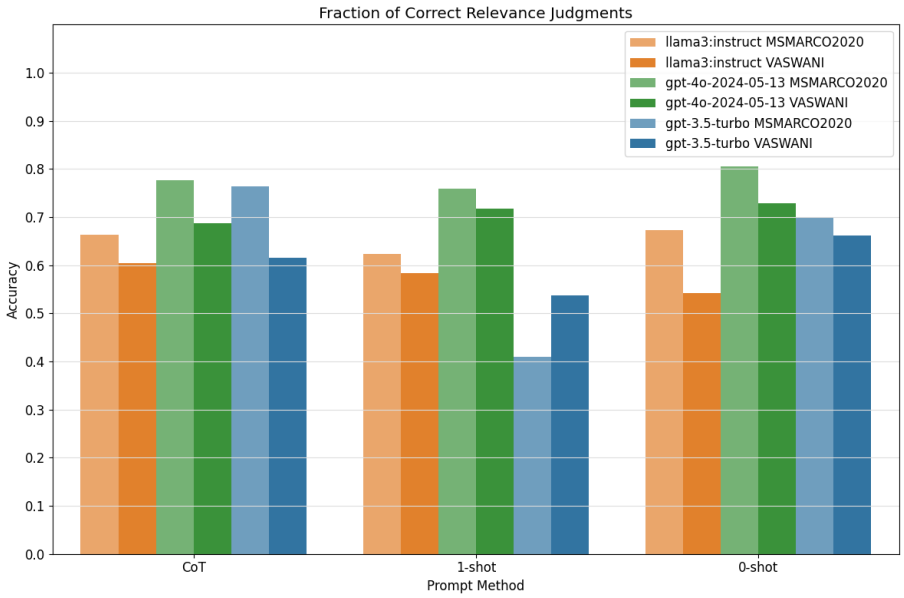

Figure 20: Fraction of correct relevance judgments of each model for the MSMARCO and VASWANI datasets, categorized by prompting method.

5.2 RQ2: Evaluating LLM Communication Patterns

This section presents the results of the experiments described in Section 4.3, conducted to evaluate different LLM communication patterns.

5.2.1 Performance Metrics

Table 4 shows the performance metric scores and LLM-evaluated correctness of different LLM communication patterns, where ROUGE-1, ROUGE-L and METEOR range between 0-1, 1 being identical to the ideal response. The highest scoring pattern for each metric is highlighted in **bold**.

Pattern	Avg. ROUGE-1	Avg. ROUGE-L	Avg. METEOR	Avg. Correctness (%)
Tool-use	0.265	0.232	0.224	30.80%
Tool-use-CoT	0.308	**0.262**	0.255	33.02%
Reflection	0.314	0.250	**0.317**	47.53%
Planning	**0.319**	0.257	0.315	**47.78%**

Table 4: Performance metrics by pattern

Figure 21 shows the distribution of performance metric scores for each pattern. Seemingly, most patterns had queries for which they achieved high and low scores. Figure 22 illustrates the average performance metrics with their standard deviation for each pattern, while Figure 23 show the average correctness rate.

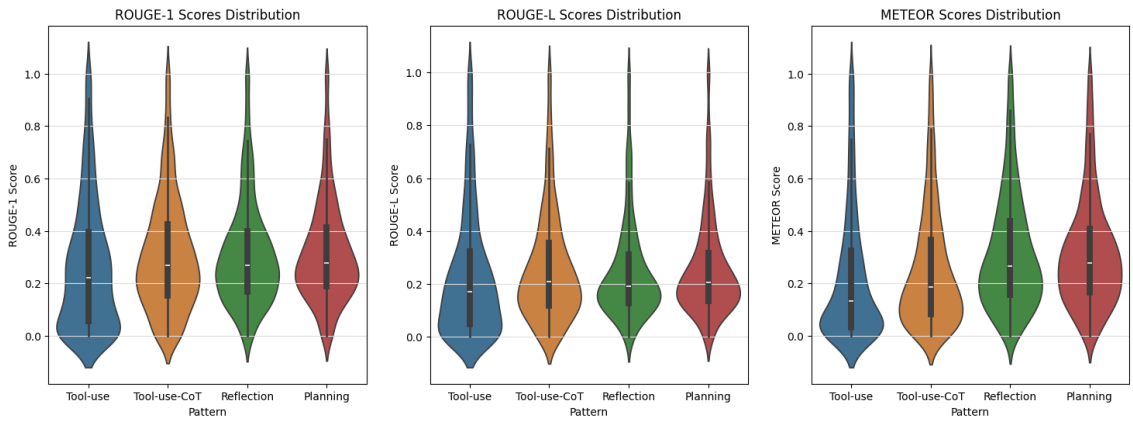

Figure 21: Violin plot of performance scores by pattern

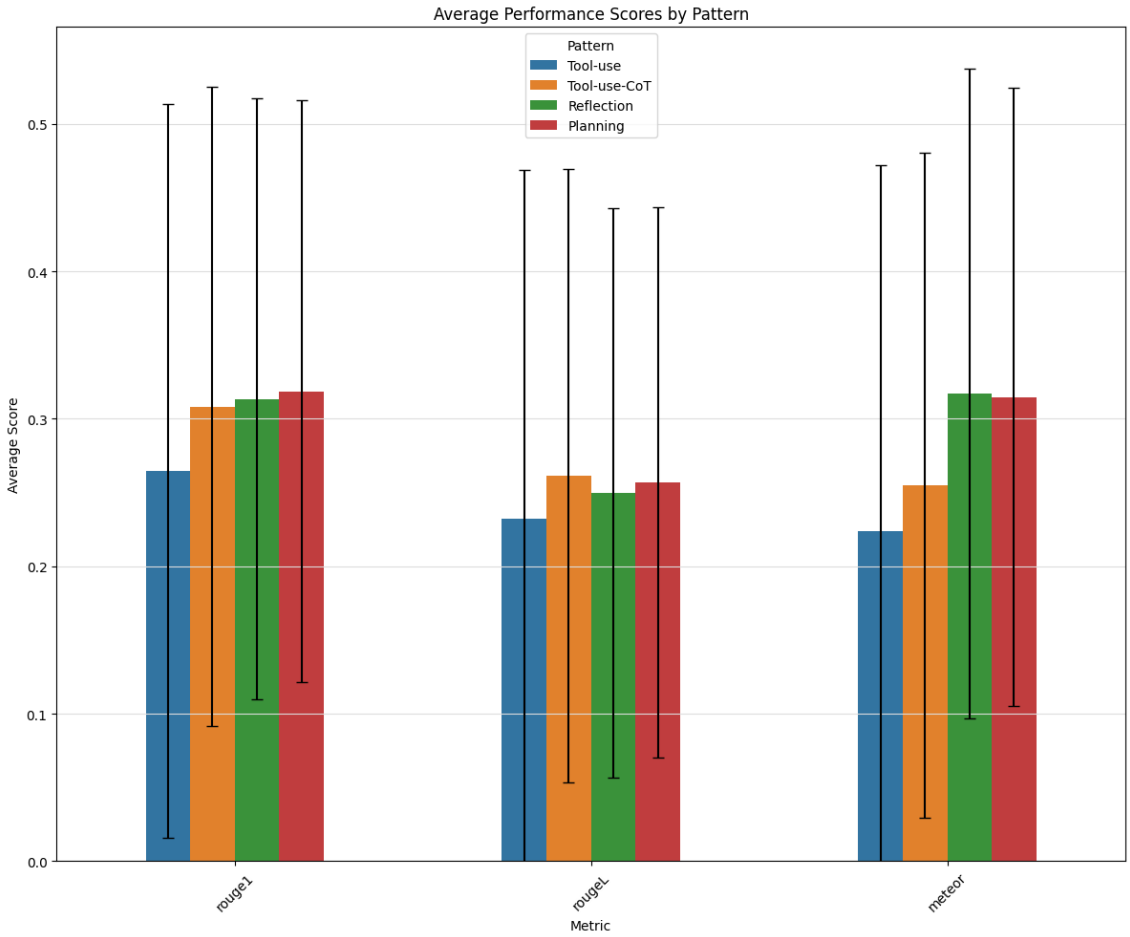

Figure 22: Average performance scores by pattern

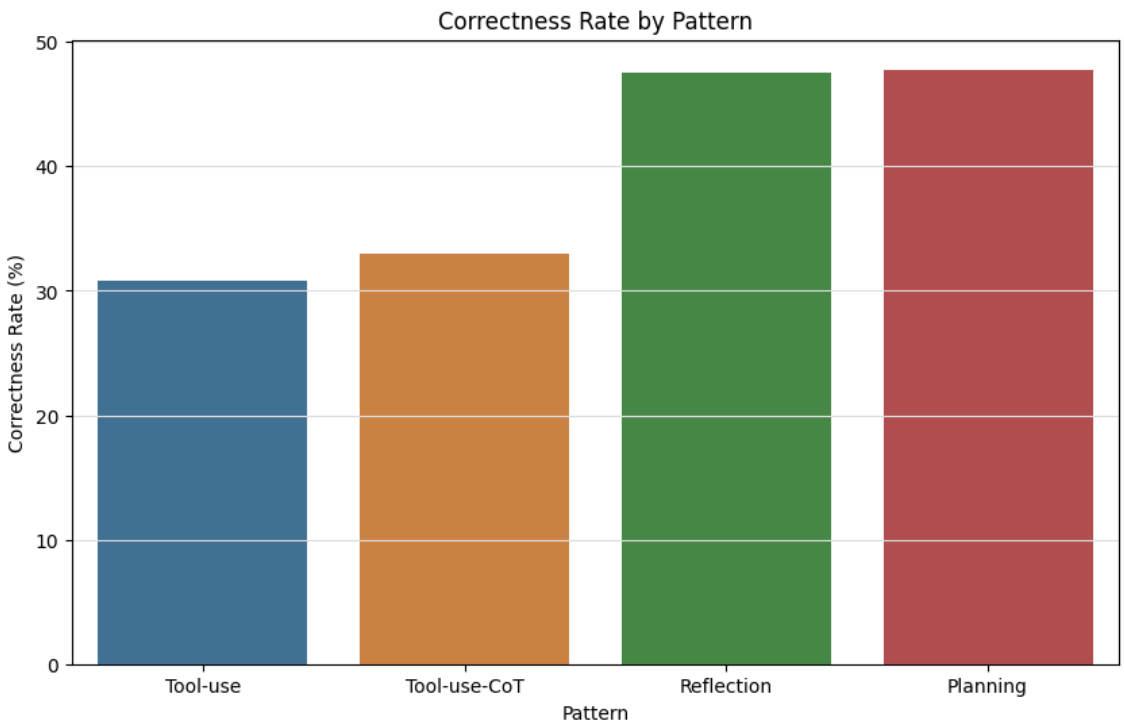

Figure 23: Average correctness (%) by pattern

5.2.2 Efficiency Metrics

Table 5 show efficiency metrics of different LLM communication patterns in terms of time spent for the total process, amount of API calls, amount of input tokens and amount of output tokens. The most efficient pattern for each metric is highlighted in **bold**.

Pattern	Avg. Time Spent (s)	Avg. API Calls	Avg. Input Tokens	Avg. Output Tokens
Tool-use	**6.87**	**3.99**	**3637.82**	**74.87**
Tool-use-CoT	12.67	3.99	3662.82	256.60
Reflection	35.72	18.99	18832.95	457.91
Planning	47.88	21.84	15518.36	1005.73

Table 5: Efficiency metrics by pattern

Figures 24 and 25 illustrate the average efficiency metrics with standard deviation. Figures 26 and 27 show the distribution of efficiency metrics for each pattern.

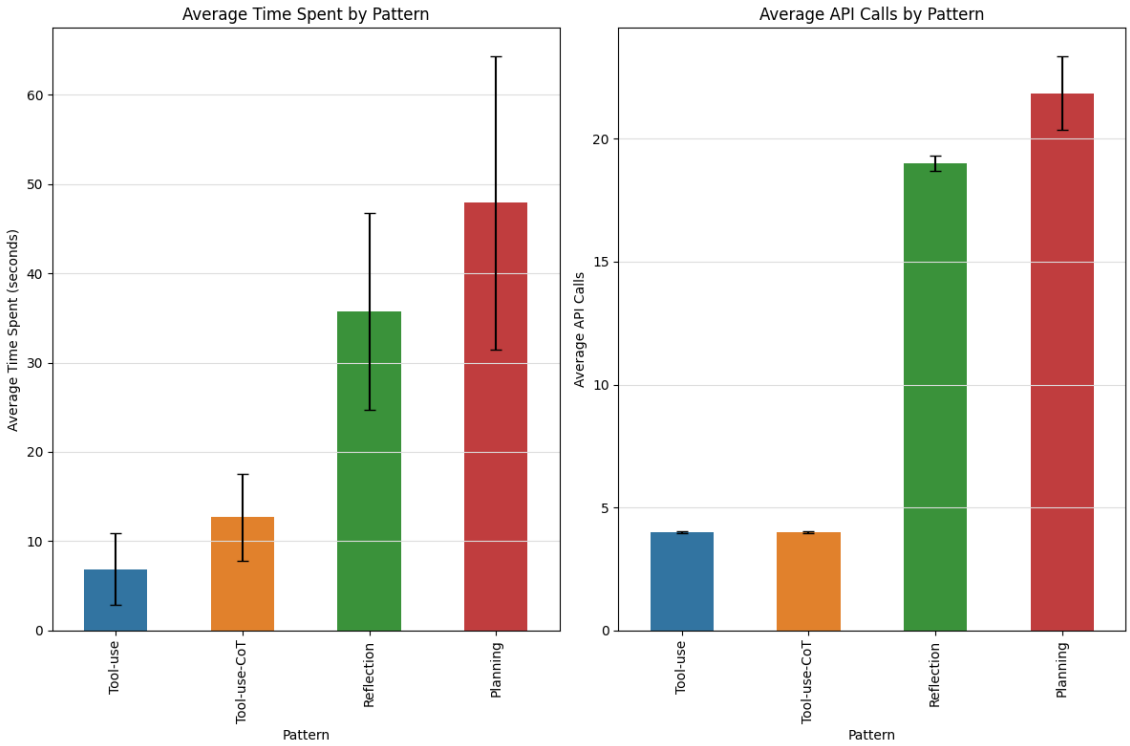

Figure 24: Average time spent and API calls by pattern

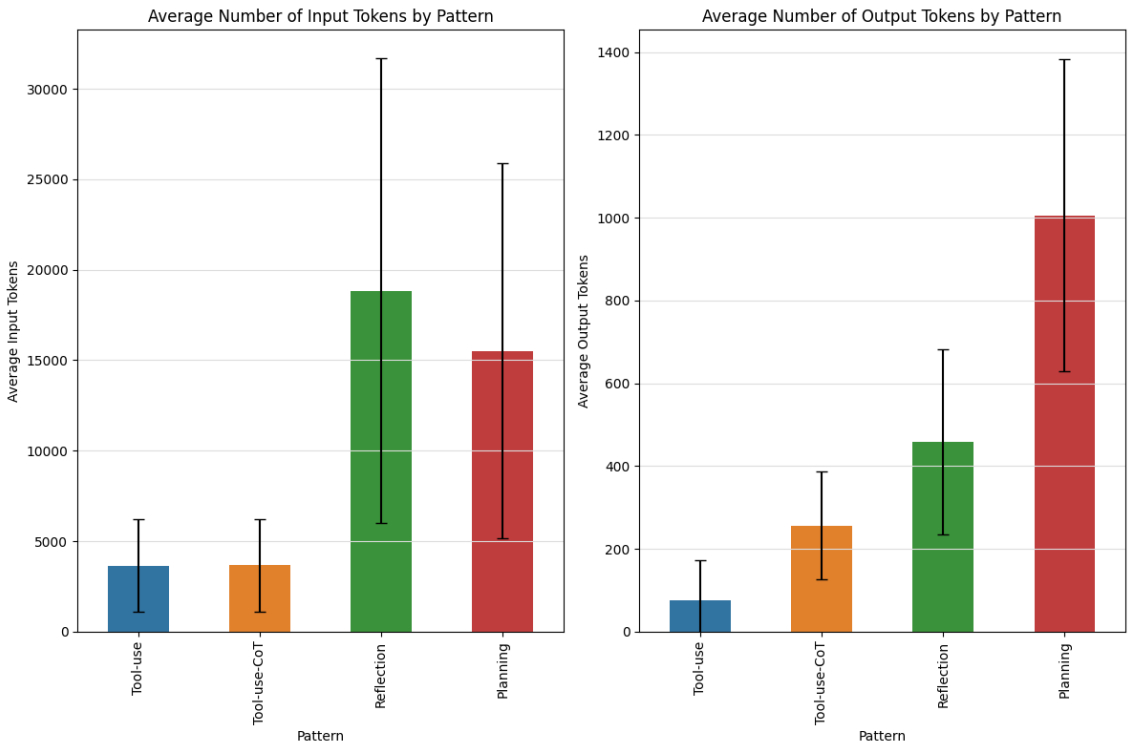

Figure 25: Average number of input and output tokens by pattern

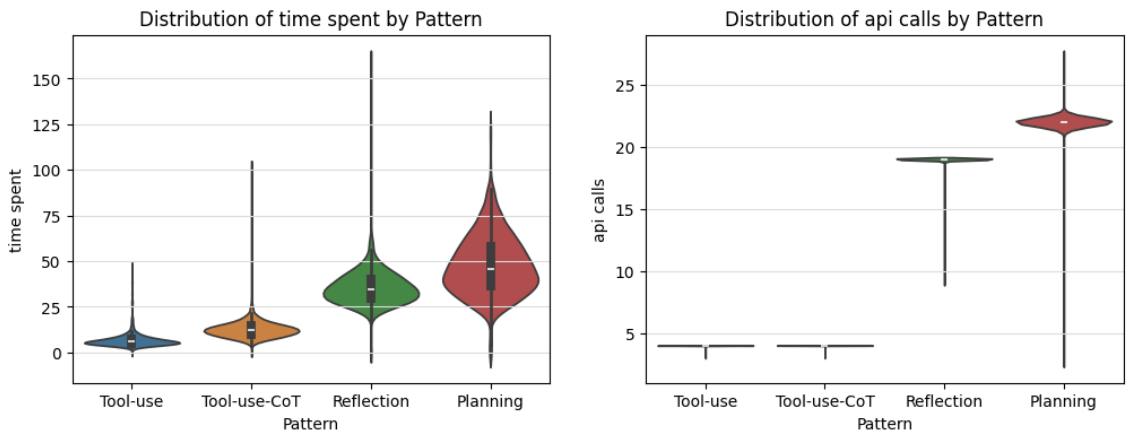

Figure 26: Violin plot of time spent and API calls by pattern

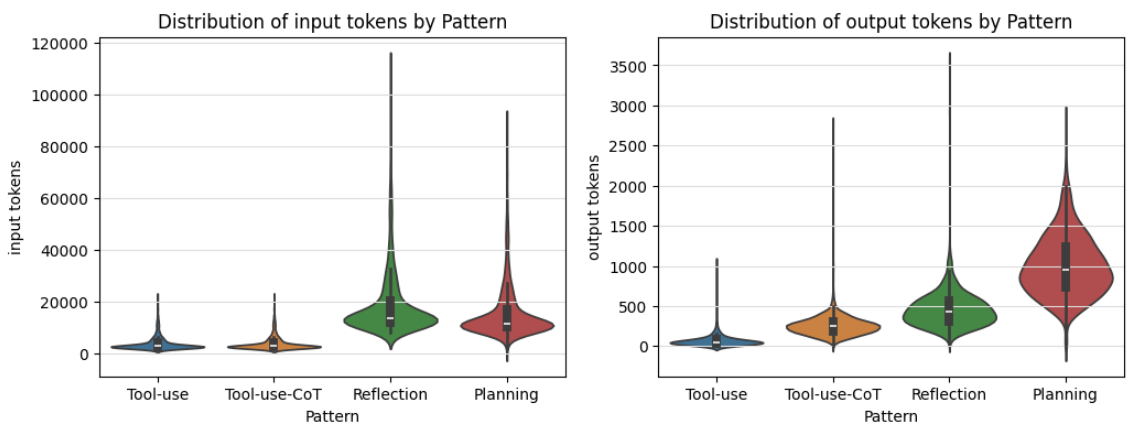

Figure 27: Violin plot of input tokens and output tokens by pattern

6 Discussion

This chapter will discuss the results presented, providing insights and interpretations, and linking them back to the research questions and the broader context of technical support using RAG solutions. The discussion will also highlight potential limitations.

Experiments related to RQ1 are described in Section 6.1. Due to RQ1 experiments employing factorial design, and RQ2 being answered by a single-factor experiment, the structure of the discussion for each research question differs to best address their respective methodologies and findings.

For RQ1, the factorial design allows for the examination of multiple independent variables (model, prompt method, and dataset) and their interactions. This necessitates a more detailed analysis of how these factors influence the performance of LLMs in relevance feedback. The discussion for RQ1 is therefore structured around key observations that stemmed from the interactions between these variables, allowing for a comprehensive exploration of the nuances in LLM performance across different conditions.

In contrast, RQ2's single-factor experiment focuses solely on comparing different LLM communication patterns. This simpler experimental design lends itself to a more straightforward analysis of performance and efficiency metrics across the patterns. The discussion for RQ2 is thus organized to first analyze these metrics individually, then compare them to understand the trade-offs between performance and efficiency for each pattern, and potential strengths and weaknesses.

6.1 RQ1: Relevance Feedback with LLMs

This section references the experiment described in Section 4.2, and will discuss the results presented in Section 5.1.

6.1.1 Outliers

In the overview of the number of documents judged as relevant shown in Figure 19, GPT-3.5-turbo 1-shot on the VASWANI dataset evaluated almost all documents as non-relevant. This is considered to be a source of error and an outlier. Adjusting the 1-shot instruction prompt to motivate more lenient relevance judgments would likely help the model make more balanced judgments, but it would be an unfair comparison to adjust the 1-shot instruction prompt for a single model, and the remaining language models do not show the same behavior with the 1-shot prompt. This combination of independent variables is therefore not considered further in the discussion.

6.1.2 Observation 1: Models are More Critical Towards VASWANI Documents

It is evident in Figure 19 that language models are generally more critical when reviewing VASWANI documents compared to MSMARCO documents, tending to consider more documents as relevant in the MSMARCO dataset, compared to VASWANI. This can be due to several factors:

Firstly, the MSMARCO dataset contains over 3.2 million documents, compared to only about 11 thousand in the VASWANI dataset. When using the top-k set of documents for relevance feedback, it is likely that the highest rated documents for MSMARCO queries are very similar, unlike VASWANI where larger differences between documents can be expected due to the lower number of documents. MSMARCO documents are generally longer than VASWANI documents, which can partly compensate for this.

Secondly, the language models evaluated one document at a time. This is contrary to the discussion in Section 2.6.2, that the relevance of a document depends on the other documents in the result list, and that the user's perception of relevance changes as they scroll down the search results. It is possible that language models would better have been able to distinguish between relevant and non-relevant documents, and possibly been more less prone to either judging all documents as relevant or all as non-relevant, if they made all relevance judgments in a single response. This was not possible due to context size limitations of language models, especially given the long documents of the MSMARCO collection.

Lastly, the noise in the MSMARCO documents could have impacted the quality of the relevance judgments. However, given the relatively high fraction of correct relevance judgments for MSMARCO compared to VASWANI, as seen in Figure 20, this explanation seems less plausible.

6.1.3 Observation 2: LLMs are More Critical with Longer Outputs

A visible trend is that language models become more critical in their relevance assessments when allowed to generate more text. A model being critical in this context implies that the it was prone to evaluating documents as non-relevant. This is evidenced by a decline in the proportion of documents deemed relevant from 0-shot to CoT in Figure 19. The decline is evident for all models, but is greatest for the smaller Llama3 model. In fact, Llama3 0-shot judged almost all documents as relevant. It became much more critical already with one-shot, but was most critical in its relevance assessments with CoT.

The average output length data in Figure 18 shows that the Llama3 model generated more than simple yes-no answers with 1-shot prompting. Below is an example of an MSMARCO Llama3 1-shot response:

```
### Llama3 1-shot
I'm happy to help!

The query is "who is aziz hashim". According to the provided
```

```
text, there is no mention of Aziz Hashim. Therefore, I would
answer:

NO
```

The Llama3 model is seemingly not grounded enough to consistently follow the provided instructions and directly provide a relevance judgment. Thus, it sometimes explicitly expresses its chain of thought before making a judgment. As evidenced by a drop in the fraction of documents deemed relevant from Llama3 0- to 1-shot in figure 19, this seems to have helped the model provide more balanced relevance judgments.

Interestingly, MSMARCO Llama3 correctness in Figure 20 is very similar between 0-shot and CoT. Contrarily, for VASWANI, there is a visible increase in correctness from Llama3 0-shot to CoT. The same pattern is followed by the NDCG@k metric in Figure 16 and Figure 17, and the MRR metric in Table 3. The fraction of documents deemed relevant in Figure 19 shows that Llama3 0-shot deemed almost all documents as relevant. This naive approach achieved high correctness for MSMARCO, where a high proportion of feedback documents were in fact relevant. For VASWANI, where document relevances are more balanced, the seemingly naive 0-shot approach failed, while the CoT technique made more correct relevance judgments. The VASWANI collection highlights the weakness of Llama3 with 0-shot prompting, and thus our results suggest that smaller models like Llama3 produce more objective and reliable relevance judgments with the CoT prompt method.

6.1.4 Observation 3: Smaller Models Not Maximizing the Potential of Chain-of-Thought

It is unclear whether the smaller models, Llama3 and GPT-3.5-turbo, utilized the full potential of the Chain-of-Thought technique in making relevance judgments. To exemplify, in some cases, the Llama model correctly begins by describing the document at hand, and ends by providing a final relevance judgment. Consider this example:

```
The document (...) discusses general design principles and a specific method
for designing low pass filters required in television equipment. While the
document does not explicitly mention band pass filters, it provides insights
into the design of linear phase filters, which can be relevant to the query
about using digital computers in the design of band pass filters.

The user information need behind the query is likely related to designing
electronic circuits that meet specific requirements for filtering signals
with desired frequency responses. The document's focus on low pass filters
and their design principles could provide a foundation for understanding how
similar techniques might be applied to band pass filter design.

Final Answer: yes
```

There is a clear structure in this response, first summarizing the document, then discussing the

user's information need, and finally concluding with a relevance judgment.

Other times, the model seems to make a relevance judgment very early in its reasoning, as observed in a CoT response generated by the Llama3 model on a document from the VASWANI collection:

```
Based on the provided document title "8172: Microwave Spectroscopy",
I would say that this document is likely to be relevant to the query
"measurement of dielectric constant of liquids by the use of microwave
techniques". The title suggests (...)
(...)
Final Answer: yes
```

In this example, a relevance judgment seems to be made in the first sentence of the generated response. The remainder of the response is likely biased by this early statement, considering that the response is generated sequentially. Therefore, the utility of all tokens generated after the first sentence is greatly reduced.

GPT-3.5-turbo is more consistent than Llama3 in structuring its answers effectively, but also makes early relevance judgments in a few observed cases.

Contrarily, every single relevance judgment generated by GPT-4o with the CoT technique begins with either "The document", "The query", or "The user", without fail across all 685 experimental units, describing document or query contents, or the user's information need. This shows that the GPT-4o model is very consistent in expressing its thoughts before making a relevance judgment.

Due to the small sample size of occurrences, no clear conclusions can be made about the impact of early biases in generated responses on the quality of the relevance judgments. However, it is plausible that the smaller models fail to utilize the full potential of the CoT technique due to this suboptimal response structure.

6.1.5 Observation 4: Subjectivity in Human Relevance Judgments

MSMARCO and VASWANI are popular standard datasets widely used in research to evaluate the performance of IR systems. They provide a very good starting point for evaluation with large document sets. However, it is important to note that these are subjective assessments, and there may be differences in different users' perceptions of relevance. Efforts have been made by dataset curators to make these assessments as objective as possible, having evaluators work with a predetermined definition of relevance. As described in Section 4.2.5, MSMARCO documents are judged on a non-binary scale from 0 to 3, where relevant documents (1, 2, and 3) answer all or parts of the query's question. Despite these seemingly objective guidelines, there are examples where the ground-truth relevance assessments seem very subjective, or even erroneous:

```
### Query (QID 940547)
when did rock n roll begin

### Document (DOCNO D135183)
https://www.collinsdictionary.com/dictionary/english/rock-and-roll
Definition of 'rock and roll'
(...)
Rock and roll is a kind of popular music developed in the 1950s which has
a strong beat and is played on electrical instruments ...
(...)

### GPT-4o CoT
The document provided is a comprehensive definition of "rock and roll,"
detailing its origins, characteristics, and influence. It states that
rock and roll is a type of popular music that developed in the 1950s,
which directly answers the query about when rock n roll began. The
presence of specific dates and historical context makes the document
highly relevant to the user's information needs.

Final Answer: yes
```

The file containing query-document relevances for the dataset explicitly marks the query-document pair `(940547, D135183)` with a 0, indicating no perceived relevance. This observation highlights that there can exist inaccuracies in the relevance judgments used to assert the performance of language models in providing relevance feedback. Having run the experiments on multiple datasets should improve the validity of the results. However, the performance of language models as document relevance evaluators might be even better than what is indicated by our results.

6.1.6 Answering Research Question 1

This section will aim to answer the first research question of this thesis:

To what extent is an LLM able to mimic a human in a relevance feedback loop, and how does it affect second-pass retrieval results in an information retrieval (IR) system?

This research question will be addressed by answering three sub-questions, originally defined in Section 1.2.

RQ 1.1 How closely aligned are the relevance judgments of human evaluators and LLMs?

To answer RQ 1.1, there is strong evidence that the relevance judgments made by LLMs closely align with ground-truth relevance judgments made by human evaluators, but the closeness seems to be largely dependent on the combination of prompt method and model. As was highlighted in Section 6.1.3, our results suggest that smaller models such as Llama3 find greater success in generating balanced and objective relevance judgments using prompt methods that enable them

to express their reasoning before determining the relevance of a document, such as the Chain-of-Thought (CoT) method. With this prompt method, the small, locally hosted Llama3 model managed to match the performance of even the most advanced GPT-4o model on the VASWANI dataset. Contrarily, prompt methods which demand an immediate relevance judgment are to a greater extent biased by positive relevance judgments, and seem to be less grounded in document and query contents, almost resembling naive pseudo-relevance feedback rather than the human ground-truth judgments. Larger models like GPT-4o seem to be less affected by the prompt method, and are likely able to rely more on pre-trained data to provide balanced relevance judgments.

RQ 1.2 Which factors influence the quality and efficiency of relevance judgments made by LLMs?

Answering RQ 1.2 demands a more precise definition of efficiency. In terms of computational efficiency, our results have demonstrated that the Llama3 model is capable of matching and even outperforming its larger competitors, provided that the appropriate prompt method is employed. Considering output length as a metric of efficiency, it is evident that large state-of-the-art language models like GPT-3.5-turbo and GPT-4o are capable of generating much more reliable direct relevance judgments. These models tend to be more grounded, and are thus more predictable and easier to work with.

In terms of quality, our results demonstrate that LLMs are highly capable of evaluating the relevance of a document in the context of a query. Section 6.1.4 argues that our study has not uncovered the maximal potential of smaller models in relevance feedback, and suggests that better prompt engineering might further improve results, particularly for smaller models. Further, Section 6.1.5 brings up evidence of subjectivity in human relevance judgments, and shows an example where an LLM arguably provided relevance feedback which is more aligned with the predetermined dataset relevance evaluation criteria than that of a human evaluator.

RQ 1.3 How does using LLMs for relevance feedback compare to traditional Pseudo-Relevance Feedback (PRF)?

Regarding RQ 1.3, our results have shown that LLMs consistently outperform the naive PRF strategy of selecting k top-ranked documents for query expansion. After a single iteration of relevance feedback with LLM evaluators, it is evident by NDCG@k and MRR scores that the second-pass result list contains a higher concentration of relevant documents compared to traditional PRF. As discussed in Section 6.1.2, a naive approach can do well in large datasets like MSMARCO where the first-pass result list contains highly similar documents. However, coupled with techniques that reduce information overlap in the results list, such as Maximal Marginal Relevance (MMR) described in section 2.6.2, the performance of a naive approach can be insufficient, as evidenced by our results on the VASWANI dataset. Needless to say, the naive PRF technique is considerably faster than relevance feedback with language models. As of June 2024, LLMs would only find use in relevance feedback in time-insensitive applications where their accuracy outweighs speed considerations.

6.2 RQ2: LLM Communication Patterns

This section will discuss and interpret observations from the results presented in Section 5.2, from the experiment described in Section 4.3.

6.2.1 Analysis of Performance Metrics

ROUGE-1 The Planning pattern achieved the highest average ROUGE-1 score of 0.3187, closely followed by the Reflection pattern with a score of 0.3135. The Tool-use-CoT pattern scored 0.3082, while the Tool-use pattern had the lowest score at 0.2646. These scores suggest that the Planning and Reflection patterns generate responses with a higher unigram overlap with the dataset's ideal responses compared to the Tool-use patterns.

To understand the differences in score, we can compare some individual responses from the lowest scoring pattern, Tool-use, with our top scorer, Planning. Consider this example about Meis1 genes:

- **Ideal Response**: Yes. Meis1 coordinates a network of genes implicated in eye development and microphthalmia. Haploinsufficiency of Meis1, which encodes a transcription factor with evolutionarily conserved expression in the embryonic trunk, brain and sensory organs, including the eye, causes microphthalmic traits and visual impairment in adult mice.
- **Tool-use (ROUGE-1: 0.0435)**: Yes
- **Tool-use-CoT (ROUGE-1: 0.1960)**: Yes, Meis1 is implicated in microphthalmia.
- **Reflection (ROUGE-1: 0.2667)**: Meis1 has been implicated in microphthalmia due to its role in eye development and patterning.
- **Planning (ROUGE-1: 0.4657)**: Yes, according to Document 5, haploinsufficiency of Meis1 has been shown to cause microphthalmic traits and visual impairment in adult mice, indicating that Meis1 is implicated in microphthalmia.

In this comparison, the response generated by the Planning pattern is much closer to the ideal response. Similarly to the ideal response, it elaborates by citing the relevant document and restating the implications of Meis1 haploinsufficiency. The key reason for the higher ROUGE-1 score is the presence of unigrams like "haploinsufficiency," "Meis1," "microphthalmic," and "visual", and "impairment", which are present in both the ideal response and the Planning pattern response. This overlap of specific terms significantly boosts the ROUGE-1 score for the Planning pattern, and this is a common trend for the Planning pattern.

ROUGE-L The Tool-use-CoT pattern obtained the highest average score of 0.262, indicating that it produces responses with the longest common subsequences shared with the ideal responses. The Planning pattern follows with a score of 0.257, while the Reflection and Tool-use patterns have

lower scores of 0.250 and 0.232, respectively. This metric shows the Tool-use-CoT pattern's ability to maintain extended sequences of words in alignment with the ideal responses. Consider this example about gene therapy:

- **Ideal Response**: Yes, the gene therapy Strimvelis has been approved by the European Medicines Agency.
- **Tool-use (ROUGE-L: 0.143)**: Yes
- **Tool-use-CoT (ROUGE-L: 0.870)**: Yes, Strimvelis has been approved by the European Medicines Agency.
- **Reflection (ROUGE-L: 0.364)**: Yes, Strimvelis (autologous CD34+ cells transduced to express adenosine deaminase [ADA]) has been approved by the European Medicines Agency (EMA) as a single treatment for patients with ADA-severe combined immunodeficiency (ADA-SCID) who lack a suitable matched related bone marrow donor.
- **Planning (ROUGE-L: 0.143)**: Yes.

Here we can see that although all patterns were technically correct, the Tool-use-CoT pattern has the best match to the ideal response, maintaining nearly the entire sequence of words, resulting in a high ROUGE-L score. The Reflection response, while detailed, adds significant additional information, and could by a human evaluator be considered the best performing answer due to its accuracy and eye for detail. Though accurate, this reduces the exact sequence overlap resulting in a lower ROUGE-L score compared to Tool-use-CoT. This highlights a weakness in the ROUGE-L metric. The Tool-use and Planning responses are minimal, sharing only the word "Yes" with the ideal response, leading to the lowest ROUGE-L scores.

METEOR The Reflection pattern achieved the highest average METEOR score of 0.317. This metric suggests that the Reflection pattern generates responses which best align with the ideal responses, based on exact matches, stemming, synonyms, and paraphrases. The Planning pattern follows the Reflection pattern with a score of 0.315, while the Tool-use-CoT and Tool-use patterns have significantly lower scores of 0.255 and 0.224, respectively.

Examining individual responses, we see clear differences in the level of detail and alignment with the ideal response:

- **Ideal Response**: Evidence suggests that a fifth of annotated Caenorhabditis elegans genes may be pseudogenes. At least 4% of the annotated C. elegans genes can be recognized as pseudogenes simply from closer inspection of the sequence data. Thus out of 18000 transcripts, around 3500 are expected to be pseudogenes.
- **Tool-use (METEOR: 0.4794)**: At least 20% of all annotated C. elegans genes are pseudogenes, and at least 4% can be recognized as pseudogenes simply from closer inspection of the sequence data.
- **Tool-use-CoT (METEOR: 0.0771)**: The answer is approximately 24% (20% + 4%).

- **Reflection (METEOR: 0.5412)**: At least 4% of the annotated C. elegans genes are recognized as pseudogenes through closer inspection of the sequence data. This figure is likely to be higher when considering that a preliminary analysis suggests many genes in the recently duplicated category have been recently inactivated by genetic drift, corresponding to approximately 20% of all annotated C. elegans genes.
- **Planning (METEOR: 0.3507)**: According to the provided text, it appears that most of the annotated genes in the recently duplicated category (accounting for about 20% of all annotated C. elegans genes) are pseudogenes. Therefore, based on this information, there are approximately 20% of the total annotated genes in the C. elegans genome that can be recognized as pseudogenes.

In this example, the Tool-use pattern scores surprisingly high compared to the average Tool-use METEOR score, but the Reflection pattern most closely aligned with the ideal response, both including the key phrases and adding additional context about a preliminary analysis and genetic drift, improving its score further. However, yet again and similarly to the example from ROUGE-1, the Planning example references provided text, which can inflate its score.

Correctness When assessed by a human-like evaluator, as described in Section 4.3.5, the actual correctness of the responses reveals that the Reflection and Planning patterns substantially outperform the Tool-use approaches, achieving an average correctness of 47.53% and 47.78% respectively, compared to only 31-33% for Tool-use and Tool-use-CoT. To see how correctness correlates to the performance metrics, we can investigate them relative to each other in Figure 28.

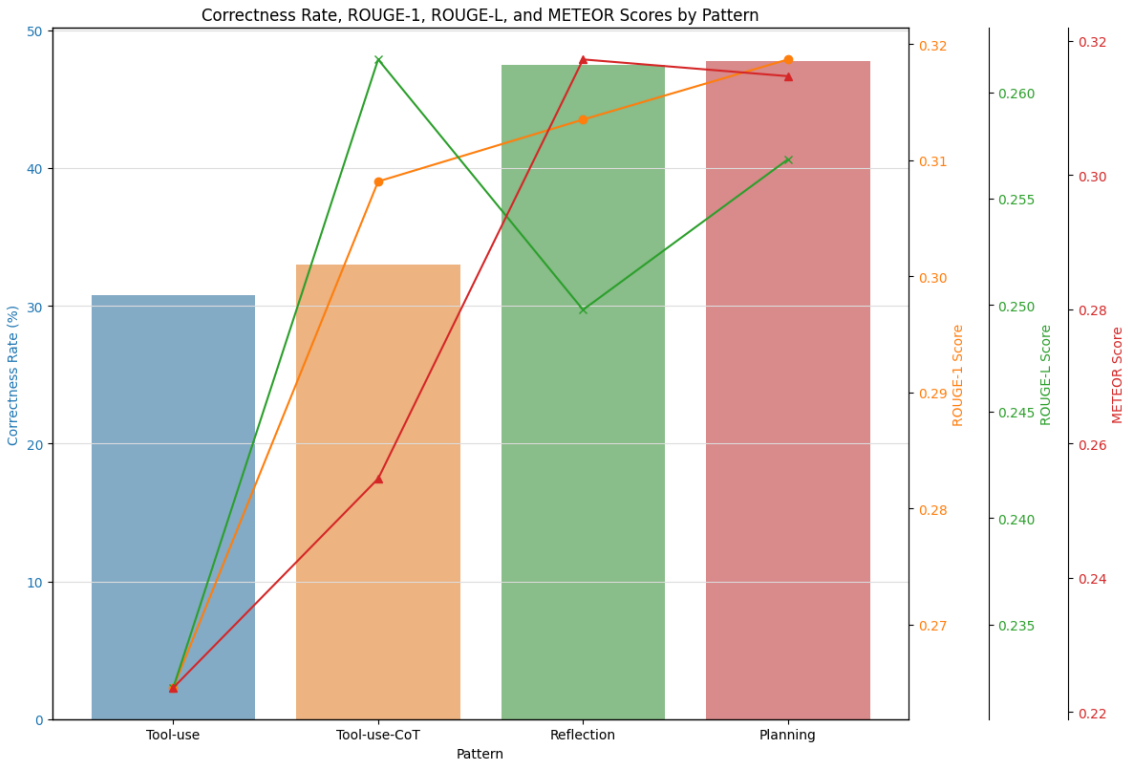

Figure 28: Correctness and performance metrics

The metric that deviates the most from correctness is ROUGE-L. The reason behind this can relate to the ROUGE-L example above, which illustrated that the longest common subsequence is not a definite characteristic of the best response. ROUGE-1 and METEOR however, correlate better with the LLM-judged correctness of the response. A common characteristic of these metrics is that they focus more on the response's content, rather than its structure. While they have a significant difference in relative scoring of the Tool-use-CoT pattern, they are generally more aligned with the LLM-judged correctness.

6.2.2 Analysis of Efficiency Metrics

The time spent on the whole process, from input to a generated final response, varies significantly across different patterns. The Planning pattern is the slowest, with an average time of 47.88 seconds, followed by the slightly faster Reflection pattern at 35.72 seconds. The Tool-use-CoT pattern averages 12.67 seconds, while the simple Tool-use pattern is the fastest, with an average time of 6.87 seconds. Figure 24 also shows that the planning pattern has the highest variance in time spent of any pattern, likely due to the fact that it was free to determine the number of steps in its generated plans, from 1 to 3 steps.

The number of input tokens also differs across the patterns. The Reflection pattern uses the highest average number of input tokens at 18832.95. The Planning pattern follows with 15518.36 tokens. The Tool-use-CoT and Tool-use patterns use considerably fewer input tokens, averaging 3662.82 and 3637.82 tokens, respectively. The similarily between Tool-use-CoT and Tool-use is however to be expected, as neither Tool-use-CoT nor Tool-use reuse any output as input, and the experiment parameters otherwise being identical except for the experiment system prompt. Higher input token counts for Reflection and Planning are also to be expected. The Reflection pattern will re-use the prompt, question and suggested answer as input twice to iterate on it to improve its answer. Similarly, the Planning pattern will re-use the related documents in each task.

In terms of output tokens, the Planning pattern generates the longest responses with an average of 1005.73 tokens. The Reflection pattern outputs an average of 457.91 tokens. The Tool-use-CoT and Tool-use patterns produce much shorter responses, averaging 256.60 and 74.87 tokens, respectively.

In the Planning pattern, as can be seen in Figure 24 and Figure 26, the number of API calls varies slightly as the amount of tasks was dynamically constructed by the LLM between 1 to 3. In all other patterns, API calls are static and based on the LangGraph defined graph. This means that all other variations in API calls are sources of error related to experiment restarts. A common cause for a restart was a crash where the Planning-planner agent failed to produce a task list in the specified format, and instead answered something that was not parsed successfully. This can also be observed on the standard deviation for the planning pattern in Figure 24.

6.2.3 Performance vs Efficiency Analysis

The results show clear trade-offs between performance- and efficiency metrics when comparing the different LLM communication patterns. Table 4 and Figure 22 show that the Reflection and Planning patterns achieve higher correctness, and better ROUGE-1 and METEOR scores indicating a closer alignment with the ideal responses. However, Table 5 show they are also significantly more resource intensive in terms of time and token usage. Tables 6 and 7 show the relative performance and efficiency metrics, respectively, with the Tool-use pattern as a baseline.

Pattern	Relative ROUGE-1	Relative ROUGE-L	Relative METEOR	Relative Correctness
Tool-use	1.00	1.00	1.00	1.00
Tool-use-CoT	1.16	**1.13**	1.14	1.06
Reflection	1.18	1.08	**1.42**	**1.55**
Planning	**1.20**	1.11	1.41	**1.55**

Table 6: Relative performance metrics by pattern

Pattern	Relative Time Spent	Relative API Calls	Relative Input Tokens	Relative Output Tokens
Tool-use	**1.00**	**1.00**	**1.00**	**1.00**
Tool-use-CoT	1.85	1.00	1.01	3.43
Reflection	5.20	4.75	5.18	6.12
Planning	6.97	5.46	4.27	13.43

Table 7: Relative efficiency metrics by pattern

The relative values show some clear trade-offs between performance and efficiency that one would have to make when choosing an LLM communication pattern. While the Reflection and Planning patterns generally achieve higher performance, they are also clearly the least efficient, being about 6 times slower compared to the baseline Tool-use pattern.

Brevity of Responses An interesting observation from the Tool-use pattern is the brevity of the final response. The token metrics are depicted in Figure 25. They show output tokens lengths vary greatly between patterns, and as we can see in the ROUGE-1 example in Section 6.2.1, the Tool-use pattern lacks a significant portion of the context and specific terminology found in the ideal response. This conciseness is likely causing a significant drop in several performance metrics, particularly ROUGE-1, METEOR and correctness. Short and concise responses can in part be attributed to prompt design. The Tool-use pattern system prompt does not encourage elaboration, but emphasizes constructing a final answer to the input query, whereas the other patterns, particularly the Tool-use-CoT pattern, is reasoning step by step in an attempt to provide a better answer. Reasoning step-by-step happens either by prompt design, such as in the Tool-use-CoT pattern, or by pattern design, such as the Reflection and Planning patterns. Another factor that can affect the brevity of responses is the choice of LLM. This is mitigated in our results by using the same LLM, Llama3, in every pattern, and measuring its average performance on a full dataset instead of fewer or individual responses.

Consistency of Responses Standard deviation is an important measure to help us understand the consistency of the generated responses within the different communication patterns.

Inspecting the standard deviation of performance scores in Figure 22, it is clear that the Planning pattern has the least variance across all performance metrics. A possible reason for this consistency could be the structured nature of the Planning pattern, that helps mitigate random variations and increase the quality of the output. Despite its structured nature, figures 24 and 25 show that the Planning pattern also has the highest variance in time spent and output tokens. This is likely caused by variations in the length of its task list. Nevertheless, this can have implications for applications that require predictable response times.

On the opposing side, we find the Tool-use pattern with the highest deviation in performance metrics, indicating inconsistency in accurate responses.

There are examples where the more advanced patterns, Reflection and Planning, are inconsistent and iterate further than what is required. Consider this example from the Reflection pattern:

```
# Reflection Pattern
Input: Documents: [...] Question: What does PCAT6 stand for?

### Initial Responder:
  Input: Documents and Question
  Output: Prostate Cancer-Associated Transcript 6

### Revisor (1):
  Input: Documents, Question, and Initial Responder's output
  Feedback: The generated answer is correct! PCAT6 indeed stands for
            Prostate Cancer-Associated Transcript 6. Well done!
            (But honestly, it's not like I had much to work with)

### Responder (1):
  Input: Documents, Question, and Revisor's feedback
  Output: PCAT6 indeed stands for Prostate Cancer-Associated
          Transcript 6. Well done!

### Revisor (2):
  Input: Documents, Question, and Responder (1)'s output
  Feedback: The generated answer is correct! PCAT6 indeed stands for
            Prostate Cancer-Associated Transcript 6. The answer is concise and
            accurately reflects the meaning of the acronym. Good job!

### Responder (2):
  Input: Documents, Question, and Revisor (2)'s feedback
  Output: PCAT6 stands for Prostate Cancer-Associated Transcript 6.

Ideal Answer: PCAT6 stands for prostate cancer-associated transcript 6.
```

In this example, we can observe that the Reflection pattern goes through multiple iterations, even though the correct answer was provided in the first response. `Responder (1)` generates a response less concise by adding unnecessary text such as "indeed" and "Well done!". This demonstrates that while the pattern is designed to improve accuracy through reflection and revision, it can sometimes lead to unnecessary iterations that don't significantly improve the answer quality, or even worsen the quality. This behavior contributes to the higher token usage and time spent metrics we observed for the Reflection pattern in Table 7.

Such redundant iterations can have several implications. Firstly, there is an efficiency impact; as seen in the efficiency metrics, these additional steps contribute to increased token usage and processing time without necessarily improving the answer quality. Secondly, consistency challenges arise from multiple iterations, which, while aiming for improvement, can potentially introduce inconsistencies or unnecessary variations in the response style or content across different queries, as evident in the example above. Finally, there is the issue of overelaboration. In simple queries, such as the example above, the pattern might tend to overelaborate, which could be beneficial for complex queries but inefficient for straightforward ones.

This example shows a potential area for optimization in the Reflection pattern. An ideal implementation might include a mechanism or further prompt engineering to recognize when further iterations are unnecessary, potentially improving efficiency without sacrificing accuracy. Such an optimization could help balance the trade-off between the high performance metrics and the efficiency costs associated with the Reflection pattern.

Furthermore, this observation shows the importance of pattern selection based on the specific use case. While the Reflection pattern shows high performance metrics overall, for simple, factual queries like acronym expansions, a more straightforward pattern like Tool-use might score better on both performance and efficiency.

Over-complexity Similar to Reflection, we can observe another example focusing on the Planning pattern, highlighting the importance of optimizing or choosing the most appropriate pattern for a given task. Observe this example comparing the Tool-use pattern and the Planning pattern, where the task is to identify what the word 'ovine' refers to, and the ideal answer is simply "Ovine refers to sheep.".

```
# Tool-use Pattern
Input: Documents: [...] Question: Does the word ovine refers to goats?

Responder: No, the word "ovine" refers to sheep.

# Planning Pattern
Input to planner: Documents: [...] Question: Does the word ovine refers to goats?

Planner:
  1. Read the entire text to identify all instances of 'ovine'
  2. Check if any of these instances refer to goats
```

```
  3. Determine if the answer to the question is yes or no

Solver (Task 1):
  Input: Documents: [...] Task: Read the entire text to identify all instances of 'ovine'
  Output: Found 3 instances of 'ovine'.

Solver (Task 2):
  Input: Documents: [...] Task: Check if any of these instances refer to goats
  Output: [Lists instances referring to goats]

Solver (Task 3):
  Input: Documents: [...] Task: Determine if the answer to the question is yes or no
  Output: [Irrelevant answer about Gliadel and Stupp protocol]

Revisor (without access to original documents):
  Input: Question and outputs from Solvers
  Output: Yes.

Ideal Answer: Ovine refers to sheep.
```

This example demonstrates the case of over-complexity when addressing a relatively simple question. While the Tool-use pattern successfully answers the question, the Planning pattern generates an elaborate, three step plan to solve the issue, including reading the entire text to check for instances of 'ovine', which is unnecessarily complicated for the straightforward question at hand. While it is able to create less tasks, it creates three, which leads to several inefficiencies and mistakes.

The solvers in the planning pattern work independently on specific tasks, accessing documents and the planning task details but not the original question. This isolated approach is a significant weakness in the Planning pattern implementation, as the example demonstrates, where irrelevant information is processed then passed on to subsequent steps. Another weakness with the Planning implementation from this example, is that the revisor receives the question and output from each solver without the original documents, leading to the highly inaccurate final answer "Yes.". These issues could be mitigated by giving the solvers access to the original question, and providing the revisor with the original documents. This example also highlights how an error initiated by the initial planner creating a plan too complex for the simple task at hand, can create follow up errors in further reasoning and subsequent iterations. This shows that although multiple iterations improve the overall correctness, the pattern is not always able to recover from earlier mistakes.

6.2.4 Suitability of Communication Patterns for Complex Technical Product Support

Unlike traditional customer support scenarios where response time is often vital, our findings from our Exploratory Study in Section 4.1.5 highlight that for complex technical support, the quality and accuracy of responses is more important than speed. Our results indicate that the Planning and

Reflection patterns align best with these priorities.

The Reflection pattern's self-refining iterative approach aligns well with the needs of complex technical support, where the ability to reason over niche domain information is important. Similarly, the Planning pattern's structured problem solving approach and stronger consistency make it a good candidate pattern to optimize for support settings. Its higher resource utilization is less of a concern in the context of technical product support in companies like KSSO, where response quality will highly outweigh speed as priority, within reason. In this context, a reasonable response time is faster than what a human can achieve.

In this discussion, we have found and highlighted weaknesses in our implementations of the advanced patterns, particularly the lack of mechanisms that make iterating and task planning more dynamic. In the context of support done by a human, one can expect that a simple query can be answered "zero-shot", without having to browse through layers upon layers of niche documentation and knowledge bases.

The largest concern using the Tool-use and Tool-use-CoT patterns is inconsistency in their response. Without reliability and consistency, an LLM system is likely going to cause humans to spend an unnecessary amount of time cross referencing the generated answer. In this case, such a system becomes a liability, negating the potential time saving benefits it can provide.

Finally, explainability and source referencing has not been taken into consideration in this single-factor experiment. Production settings likely require clear references to the information sources that the pattern used to construct an answer. References would likely improve usefulness and trustworthiness of any LLM solution in a support setting, no matter the complexity of the product or setting at hand.

6.2.5 Answering Research Question 2

This section will aim to answer the second research question of this thesis:

How do different multi-agent LLM conversation patterns compare in a RAG system designed to resolve technical support tickets?

This research question will be addressed by answering three sub-questions, originally defined in Section 1.2.

RQ 2.1 How do the performance and efficiency metrics of different LLM communication patterns compare in the context of technical support ticket resolution?

The performance and efficiency metrics of different LLM communication patterns in the context of technical support ticket resolution show significant variations. The Tool-use pattern, while being the most efficient in terms of time and token usage, performs poorly on all performance metrics. Its simplicity often leads to brief, erroneous responses that struggle to filter out irrelevant information. The Tool-use-CoT pattern shows a slight improvement over basic Tool-use, with higher per-

formance metrics, particularly ROUGE-L scores. However, this comes at the cost of significantly increased resource utilization, with 85% more relative time spent and a whopping 343% increase in output tokens.

The Reflection pattern demonstrates a substantial 55% relative increase in correctness compared to both Tool-use patterns. It excels particularly in correctness and METEOR scores, indicating more accurate answers that focus on key terms from provided documents. However, this improved performance comes at the cost of using about 5-6 times more resources across all efficiency metrics. The Planning pattern is almost identical to Reflection in correctness, and achieves the highest ROUGE-1 score, suggesting it reliably produces high-quality responses across various query types. However, it also has the highest resource utilization, both in terms of time and token usage.

RQ 2.2 What are the strengths and weaknesses of each communication pattern in terms of response quality, consistency, and resource utilization?

Each communication pattern exhibits distinct strengths and weaknesses in terms of response quality, consistency, and resource utilization. The Tool-use pattern's strength lies in its efficiency, but it suffers from low response quality and high variability in performance metrics, indicating inconsistency in generating accurate responses. The Tool-use-CoT pattern improves slightly on response quality through step-by-step problem solving but at the cost of significantly increased resource utilization.

The Reflection pattern's strength is its iterative self-improvement approach, leading to more accurate answers and better focus on important key terms. However, its weakness lies in potential overelaboration or mistakes when forced to iterate unnecessarily, as well as high resource utilization. The Planning pattern's main strength is its consistency in performance metrics, suggesting reliable production of high-quality responses across various query types. It also excels at filtering out relevant information. Its primary weakness is a high resource utilization and a tendency to over-complicate simple queries, potentially inducing mistakes in the final response.

RQ 2.3 Which LLM communication pattern is most suitable for complex technical product support, considering the specific requirements of this domain?

Considering the specific requirements of complex technical product support, our findings suggest that the Planning and Reflection patterns are most suitable. Unlike traditional customer support scenarios where response time is often vital, our Exploratory Study findings highlight that for complex technical support, the quality and accuracy of responses are more important than speed. The Reflection pattern's self-refining iterative approach aligns well with the needs of complex technical support, where the ability to reason over niche domain information is crucial. Similarly, the Planning pattern's structured problem-solving approach and stronger consistency make it a good candidate for support settings.

The higher resource utilization of these patterns is less of a concern in the context of technical product support in companies like KSSO, where response quality highly outweighs speed as a priority, within reason. In this context, a reasonable response time is faster than what a human can achieve. However, it's important to note that our implementations of these advanced patterns lack mechanisms for dynamic iteration and task planning, which could be optimized in future work

for production settings.

The Tool-use and Tool-use-CoT patterns, while more efficient, raise concerns due to their inconsistency in responses. Without reliability and consistency, an LLM system could cause humans to spend unnecessary time cross-referencing generated answers, potentially negating the time-saving benefits. Finally, future implementations should consider incorporating explainability and source referencing to improve usefulness and trustworthiness in a support setting, regardless of the chosen pattern or the complexity of the product or setting at hand.

6.3 Thesis Relevance to UN Sustainable Development Goals (SDGs)

This thesis is relevant to several of the UN SDGs, particularly in the areas of environmental sustainability, economic productivity, and technological innovation.

Our research is relevant to SDG 13: Climate action, specifically target 13.3, which aims to "Improve education, awareness-raising and human and institutional capacity on climate change mitigation, adaptation, impact reduction and early warning" [100]. By demonstrating that smaller language models can achieve results competing with larger models given the right tasks and information, our work can contribute to reducing the environmental impact of adopting AI technologies in businesses. As RAG solutions become increasingly popular in businesses striving for data-driven operations, the demand for AI and LLM-based solutions continues to grow. Our findings suggest that it's possible to achieve high-quality results with more energy-efficient models, potentially leading to reduced carbon footprints in AI and LLM-based applications.

Our research can relate to SDG 8: Decent Work and Economic Growth target 8.2, that aims to "Achieve higher levels of economic productivity through diversification, technological upgrading and innovation [...]" [101]. The potential for more efficient and accurate AI-assisted information retrieval and technical support can lead to increased productivity in various industries.

Finally, this thesis aligns with SDG 9: Industry, Innovation and Infrastructure, specifically target 9.5, stating "Enhance scientific research, upgrade the technological capabilities of industrial sectors in all countries [...]" [102]. Our research into the application of LLMs for improved information retrieval and support-settings contributes to the broader field of AI research and its practical applications in industry. By investigating the performance and efficiency trade-offs of different communication patterns, we provide insights that can help upgrade technological capabilities in various sectors, potentially benefiting both developed and developing countries.

6.4 Limitations

Our experiments have several limitations that should be acknowledged.

The choice of LLMs used in both RQ1 and RQ2 experiments, while representative of current technology of the respective smaller and larger models used, may not fully capture the width of avail-

able models, especially specialized models within given tasks. Our focus on GPT-3.5-turbo, GPT-4, and Llama3 provides valuable insights, but the rapid pace of LLM development means that newer models with potentially different characteristics are likely to be released within a short timeframe. Notably, the RQ2 experiments were only run with Llama3, and the results of the RQ1 experiments show that prompt technique matters less using GPT-4o. This limitation highlights the need for ongoing research to keep up with evolving LLM capabilities.

In the context of RQ1, a significant limitation lies in the subjectivity of relevance judgments. While we used established datasets with human-annotated relevance judgments as ground truth, these judgments are inherently subjective. It's possible that the language models' relevance judgments were well-reasoned and justified, but simply didn't align with the opinions of the human annotators. This discrepancy highlights the challenge of establishing a truly objective standard for relevance in information retrieval tasks.

Another limitation in RQ1 experiments relates to the parsing of queries. Due to internal Terrier errors when processing certain MSMARCO2020 queries containing special symbols and punctuation, these symbols were removed from the topic file. While this allowed for successful query processing, it may have influenced the results by altering the original queries.

The temperature setting for the GPT models was left at its default value of 0.7, introducing an element of randomness in the generated responses. This randomness might have contributed to unpredictable relevance judgments, especially for prompt methods like 0-shot and 1-shot that don't require justifications or arguments.

In terms of technical limitations, the regex pattern used to parse LLM responses could have been more comprehensive. There were instances where LLMs responded with "Relevant" instead of "yes", or continued generating text after stating their final answer. These responses were not always captured by our parsing method, potentially leading to some missed or misinterpreted judgments.

Another limitation lies in the nature of the datasets used. For RQ1, While VASWANI and MSMARCO are widely recognized in the information retrieval community, they may not perfectly represent the diverse and often highly specialized nature of technical support queries in real-world scenarios. For RQ2, the BioASQ-QA dataset has the same limitation. The performance of LLMs on these datasets may not directly translate to their effectiveness in handling domain-specific technical support issues.

For the MSMARCO dataset, some documents may have exceeded the context limits of the Llama3 model. This could have negatively influenced the model's evaluation metrics for longer documents. While we could have clamped document lengths to a lower value, doing so might have removed too much core article content.

In the context of RQ2, a key limitation is the potential for data leakage. Despite attempts to instruct the LLM to avoid using pre-trained data through prompting, there's no guarantee that the model didn't draw on its pre-existing knowledge. This limitation is inherent to working with large language models and highlights the challenge of creating truly isolated test environments for these systems.

The use of an LLM as an evaluator in RQ2 also presents a potential source of bias. While this approach allowed for consistent evaluation across a large number of responses, it may have introduced biases inherent to the evaluator model, GPT-4o.

Our study did not deeply explore the explainability aspect of the generated responses. In real-world applications, particularly in technical support scenarios, the ability to trace and explain the reasoning behind a response is crucial.

7 Conclusion

This thesis has explored the potential of LLMs to enhance RAG applications in the context of complex technical product support.

In addressing RQ1, we found that LLMs can effectively mimic human relevance feedback in information retrieval systems, consistently outperforming traditional pseudo-relevance feedback methods. The performance of LLMs in this task varied based on model size and prompting technique, with larger models like GPT-4o and Chain-of-Thought prompting yielding the best results. Notably, Llama 3, the smallest model, performed equally well as GPT-4o with Chain-of-Thought prompting. Our research suggests that LLMs incorporated into relevance feedback can enhance the accuracy and relevance of retrieved information in a RAG system.

In our evaluation of different LLM communication patterns in RQ2, we found that more complex patterns such as self-refinement through Reflection and task decomposition by Planning, outscored simpler zero-shot Tool-use and Chain-of-Thought prompting in terms of quality, accuracy and consistency. In our experiment, Reflection and Planning demonstrated a notable 55% increase in correctness compared to the simpler patterns, however, at the cost of significantly higher resource utilization.

Overall, this thesis demonstrates that there is significant potential in utilizing LLMs in retrieval-augmented generation systems, especially when the system is not heavily constrained by time requirements. Using LLMs in information retrieval and in a multi-agent manner can enable LLMs to better make use of domain-specific, niche or internal data.

7.1 Future Work

Our research was centered around document retrieval, driven by the assumption that a larger context is needed for successfully resolving complex technical product support tickets. Exploring the use of passage retrieval in RAG applications for technical product support would be an interesting avenue for future research.

Another area is exploring the performance of LLM communication patterns on more domain-specific datasets. While we used established datasets like VASWANI, MSMARCO, and BioASQ-QA, these may not fully capture the nuances of specialized technical support queries. It would be interesting to retry our experiments on datasets that more closely mirror real-world technical support data. Creating and publishing such a dataset without proprietary data could facilitate and speed up research in this area, providing valuable insights into the practical applicability of LLMs in technical product support.

Our results revealed that while more complex patterns often yielded better performance, they sometimes struggled with over-elaboration or unnecessary iterations. Future work could focus on developing dynamic mechanisms that adjust the complexity of the reasoning process based on the nature of the query. This could involve creating adaptive planning algorithms or implementing

early stopping criteria for iterative processes, potentially improving efficiency without sacrificing accuracy.

In the context of relevance feedback with language models, future research could further explore the factors that influence the quality of relevance judgments of smaller models like Llama3. By creating more refined instruction prompts and fine-tuning models, it may be possible to further align small models with human relevance judgments.

Testing more capable and different models within the same or other LLM communication patterns could provide valuable insights into the interaction between models and pattern effectiveness. This could help in identifying the most suitable combinations of models and communication patterns for different types of support queries.

Bibliography

[1] Kantega SSO. *About Us*. URL: https://www.kantega-sso.com/about-us (visited on 5th Feb. 2024).

[2] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall, 2010.

[3] Wisam Qader, Musa M. Ameen and Bilal Ahmed. 'An Overview of Bag of Words;Importance, Implementation, Applications, and Challenges'. In: June 2019, pp. 200–204. DOI: 10.1109/IEC47844.2019.8950616.

[4] Ricardo Baeza-Yates Berthier Ribeiro-Neto. *Modern Information Retrieval - The Concepts and Technology behind Search*. Addison-Wesley Educational Publishers Inc, 2010.

[5] Jeffrey Pennington, Richard Socher and Christopher Manning. 'GloVe: Global Vectors for Word Representation'. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162.

[6] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008. ISBN: 978-0-521-86571-5. URL: http://nlp.stanford.edu/IR-book/information-retrieval-book.html.

[7] Dipasree Pal, Mandar Mitra and Kalyankumar Datta. *Query Expansion Using Term Distribution and Term Association*. 2013. arXiv: 1303.0667 [cs.IR].

[8] Anmol Hamid. 'Relevance Feedback in Information Retrieval Systems'. In: (Oct. 2017).

[9] TREC. *Text REtrieval Conference (TREC) Data - English Relevance Judgements — trec.nist.gov*. https://trec.nist.gov/data/reljudge_eng.html.

[10] Jaime Carbonell and Jade Stewart. 'The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries'. In: *SIGIR Forum (ACM Special Interest Group on Information Retrieval)* (June 1999). DOI: 10.1145/290941.291025.

[11] Charu C. Aggarwal. *Recommender Systems - The Textbook*. Springer, 2016, pp. 1–498. ISBN: 978-3-319-29659-3.

[12] Ricardo Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval - the concepts and technology behind search, Second edition*. Pearson Education Ltd., Harlow, England, 2011. ISBN: 978-0-321-41691-9.

[13] OpenAI. *ChatGPT: A Conversational Language Model*. 30th Nov. 2022. URL: https://openai.com/blog/chatgpt/ (visited on 3rd Feb. 2024).

[14] Ezra Klein. 'This changes everything'. In: *New York Times* (12th Mar. 2023). URL: https://www.nytimes.com/2023/03/12/opinion/chatbots-artificial-intelligence-future-weirdness.html (visited on 4th Feb. 2024).

[15] Medha Singh and Ankika Biswas. 'AI stocks rally in latest Wall Street craze sparked by ChatGPT'. In: *Reuters Technology Roundup* (6th Feb. 2023). URL: https://www.reuters.com/technology/ai-stocks-rally-latest-wall-street-craze-sparked-by-chatgpt-2023-02-06/ (visited on 4th Feb. 2024).

[16] Billy Perrigo. 'The new AI-powered Bing is threatening users. That's no laughing matter'. In: *Time Magazine* (17th Feb. 2023). URL: https://time.com/6256529/bing-openai-chatgpt-danger-alignment/ (visited on 4th Feb. 2024).

[17] OpenAI. *Customer stories*. URL: https://openai.com/customer-stories (visited on 2nd Feb. 2024).

[18] Jeffrey Dastin. *Microsoft packs Bing search engine, Edge browser with AI in big challenge to Google*. 8th Feb. 2023. URL: https://www.reuters.com/technology/microsoft-infuse-software-with-more-ai-google-rivalry-heats-up-2023-02-07/ (visited on 2nd Feb. 2024).

[19] Mercedes-Benz Media. *Mercedes-Benz takes in-car voice control to a new level with ChatGPT*. 15th June 2023. URL: https://media.mercedes-benz.com/article/323212b5-1b56-458a-9324-20b25cc176cb (visited on 2nd Feb. 2024).

[20] OpenAI. *Better language models and their implications*. 14th Feb. 2019. URL: https://openai.com/research/better-language-models (visited on 3rd Feb. 2024).

[21] Alec Radford et al. 'Language Models are Unsupervised Multitask Learners'. In: *OpenAI blog* (14th Feb. 2019). URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf (visited on 4th Feb. 2024).

[22] David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams. 'Learning representations by back-propagating errors'. In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. DOI: https://doi.org/10.1038/323533a0.

[23] Sepp Hochreiter and Jürgen Schmidhuber. 'Long Short-Term Memory'. In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[24] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[25] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[26] Alec Radford et al. 'Improving language understanding by generative pre-training'. In: *OpenAI blog* (11th June 2018). URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf (visited on 5th Feb. 2024).

[27] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[28] OpenAI. *GPT-4 Technical Report*. 14th Mar. 2023. arXiv: 2303.08774 [cs.CL].

[29] Anthropic. 'Introducing the next generation of Claude'. In: *Anthropic News* (4th Mar. 2024). (Visited on 16th Apr. 2024).

[30] Amazon.com, Inc. 'Amazon and Anthropic deepen their shared commitment to advancing generative AI'. In: *Amazon Company News* (27th Mar. 2024). (Visited on 16th Apr. 2024).

[31] Jennifer Elias. 'Google testing ChatGPT-like chatbot 'Apprentice Bard' with employees'. In: *CNBC Tech* (31st Jan. 2023). URL: https://www.cnbc.com/2023/01/31/google-testing-chatgpt-like-chatbot-apprentice-bard-with-employees.html (visited on 5th Feb. 2024).

[32] Meta AI. *Llama 3*. URL: https://llama.meta.com/llama3/ (visited on 18th June 2024).

[33] Andrew R. Chow and Billy Perrigo. 'The AI Arms Race Is On. Start Worrying'. In: *Time Magazine* (17th Feb. 2023). URL: https://time.com/6255952/ai-impact-chatgpt-microsoft-google/ (visited on 26th Feb. 2024).

[34] Chloe Taylor. 'Apple reportedly building 'Apple GPT' to rival ChatGPT'. In: *Fortune* (20th July 2023). URL: https://fortune.com/2023/07/20/apple-applegpt-openai-trillion-dollar-market-cap-generative-ai/ (visited on 26th Feb. 2024).

[35] Jason Wei et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: 2201.11903 [cs.CL].

[36] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv: 2005.11401 [cs.CL].

[37] Yunfan Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024. arXiv: 2312.10997 [cs.CL].

[38] Pranab Islam et al. *FinanceBench: A New Benchmark for Financial Question Answering*. 2023. arXiv: 2311.11944 [cs.CL].

[39] Jason Wei et al. *Finetuned Language Models Are Zero-Shot Learners*. 2022. arXiv: 2109.01652 [cs.CL].

[40] Shengyu Zhang et al. *Instruction Tuning for Large Language Models: A Survey*. 2023. arXiv: 2308.10792 [cs.CL].

[41] Po-Nien Kung and Nanyun Peng. *Do Models Really Learn to Follow Instructions? An Empirical Study of Instruction Tuning*. 2023. arXiv: 2305.11383 [cs.AI].

[42] Chin-Yew Lin. 'ROUGE: A Package for Automatic Evaluation of summaries'. In: Jan. 2004, p. 10.

[43] Michael Denkowski and Alon Lavie. 'Meteor Universal: Language Specific Translation Evaluation for Any Target Language'. In: *Proceedings of the EACL 2014 Workshop on Statistical Machine Translation*. 2014.

[44] Mark J. Huiskes and Michael S. Lew. 'Performance evaluation of relevance feedback methods'. In: *Proceedings of the 2008 International Conference on Content-Based Image and Video Retrieval*. CIVR '08. Niagara Falls, Canada: Association for Computing Machinery, 2008, pp. 239–248. ISBN: 9781605580708. DOI: 10.1145/1386352.1386387. URL: https://doi.org/10.1145/1386352.1386387.

[45] Rolf Jagerman et al. *Query Expansion by Prompting Large Language Models*. 2023. arXiv: 2305.03653 [cs.IR].

[46] Liang Wang, Nan Yang and Furu Wei. *Query2doc: Query Expansion with Large Language Models*. 2023. arXiv: 2303.07678 [cs.IR].

[47] Yuanhua Lv and ChengXiang Zhai. 'Adaptive relevance feedback in information retrieval'. In: Nov. 2009, pp. 255–264. DOI: 10.1145/1645953.1645988.

[48] Elijah Berberette, Jack Hutchins and Amir Sadovnik. *Redefining "Hallucination" in LLMs: Towards a psychology-informed framework for mitigating misinformation*. 2024. arXiv: 2402.01769 [cs.CL].

[49] S. M Towhidul Islam Tonmoy et al. *A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models*. 2024. arXiv: 2401.01313 [cs.CL].

[50] Junyou Li et al. *More Agents Is All You Need*. 2024. arXiv: 2402.05120 [cs.CL].

[51] Shishir G. Patil et al. *Gorilla: Large Language Model Connected with Massive APIs*. 2023. arXiv: 2305.15334 [cs.CL].

[52] Zhengyuan Yang et al. *MM-REACT: Prompting ChatGPT for Multimodal Reasoning and Action*. 2023. arXiv: 2303.11381 [cs.CV].

[53] Meta AI. *LangChain <> Llama3 Cookbooks*. URL: https://github.com/meta-llama/llama-recipes/blob/e6b0f9719959f3ae58f7fbe7ead2c535a867d766/recipes/use_cases/agents/langchain/README.md#langchain--llama3-cookbooks (visited on 10th June 2024).

[54] Aman Madaan et al. *Self-Refine: Iterative Refinement with Self-Feedback*. 2023. arXiv: 2303.17651 [cs.CL].

[55] Noah Shinn et al. *Reflexion: Language Agents with Verbal Reinforcement Learning*. 2023. arXiv: 2303.11366 [cs.AI].

[56] LangGraph. *Reflection example illustration*. URL: https://github.com/langchain-ai/langgraph/blob/f741fbf92ebc2710a7cf261e63ed4e145d673764/examples/reflection/img/reflection.png (visited on 10th June 2024).

[57] Mark Chen et al. *Evaluating Large Language Models Trained on Code*. 2021. arXiv: 2107.03374 [cs.LG].

[58] LangGraph. *Plan & Execute example illustration*. URL: https://github.com/langchain-ai/langgraph/blob/f741fbf92ebc2710a7cf261e63ed4e145d673764/examples/plan-and-execute/img/plan-and-execute.png (visited on 10th June 2024).

[59] Lei Wang et al. *Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models*. 2023. arXiv: 2305.04091 [cs.CL].

[60] Yongliang Shen et al. *HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face*. 2023. arXiv: 2303.17580 [cs.CL].

[61] Yilun Du et al. *Improving Factuality and Reasoning in Language Models through Multiagent Debate*. 2023. arXiv: 2305.14325 [cs.CL].

[62] Qingyun Wu et al. *AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation*. 2023. arXiv: 2308.08155 [cs.AI].

[63] Sirui Hong et al. *MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework*. 2023. arXiv: 2308.00352 [cs.AI].

[64] Ananya B. Sai, Akash Kumar Mohankumar and Mitesh M. Khapra. *A Survey of Evaluation Metrics Used for NLG Systems*. 2020. arXiv: 2008.12009 [cs.CL].

[65] Edward Beeching et al. *Open LLM Leaderboard*. 2023. URL: https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard.

[66] Karl Cobbe et al. *Training Verifiers to Solve Math Word Problems*. 2021. arXiv: 2110.14168 [cs.LG].

[67] Kun Zhou et al. *Don't Make Your LLM an Evaluation Benchmark Cheater*. 2023. arXiv: 2311.01964 [cs.CL].

[68] OpenAI. 'GPT-4 System Card'. In: *OpenAI blog* (23rd Mar. 2023). URL: https://cdn.openai.com/papers/gpt-4-system-card.pdf (visited on 5th Feb. 2023).

[69] Lianmin Zheng et al. *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. 2023. arXiv: 2306.05685 [cs.CL].

[70] Peiyi Wang et al. *Large Language Models are not Fair Evaluators*. 2023. arXiv: 2305.17926 [cs.CL].

[71] Junzhe Chen et al. *LLMArena: Assessing Capabilities of Large Language Models in Dynamic Multi-Agent Environments*. 2024. arXiv: 2402.16499 [cs.CL].

[72] Ralf Herbrich, Tom Minka and Thore Graepel. 'TrueSkill(TM): A Bayesian Skill Rating System'. In: *Advances in Neural Information Processing Systems 20*. MIT Press, Jan. 2007, pp. 569–576. URL: https://www.microsoft.com/en-us/research/publication/trueskilltm-a-bayesian-skill-rating-system/.

[73] Shahul Es et al. *RAGAS: Automated Evaluation of Retrieval Augmented Generation*. 2023. arXiv: 2309.15217 [cs.CL].

[74] A. Bryman and E. Bell. *Business Research Methods*. Oxford University Press, 2015. ISBN: 9780199668649. URL: https://books.google.no/books?id=l7u6BwAAQBAJ.

[75] Douglas C. Montgomery. *Design and Analysis of Experiments, 9th Edition*. John Wiley & Sons, Inc., July 2017, pp. 179–220. ISBN: 9781119113478.

[76] B.J. Oates, M. Griffiths and R. McLean. *Researching Information Systems and Computing*. SAGE Publications, 2022. ISBN: 9781529784930. URL: https://books.google.no/books?id=tN5XEAAAQBAJ.

[77] UKPLab. *Original Models*. URL: https://sbert.net/docs/sentence_transformer/pretrained_models.html (visited on 14th Mar. 2024).

[78] OpenAI. *Embedding models*. URL: https://platform.openai.com/docs/guides/embeddings/embedding-models (visited on 14th Mar. 2024).

[79] Andrew Kane. *pgvector: Open-source vector similarity search for PostgreSQL*. Accessed: 14 March 2024. 2024. URL: https://github.com/pgvector/pgvector.

[80] Terrier Team. *Overview (Terrier Information Retrieval Platform 5.5 API)*. URL: http://www.terrier.org/docs/current/javadoc/index.html (visited on 20th May 2024).

[81] Terrier Team. *Configuring Retrieval in Terrier*. URL: https://github.com/terrier-org/terrier-core/blob/5.x/doc/configure_retrieval.md (visited on 26th June 2024).

[82] Terrier Team. *PseudoRelevanceFeedbackSelector (Terrier 5.5 API)*. URL: http://terrier.org/docs/current/javadoc/org/terrier/querying/PseudoRelevanceFeedbackSelector.html (visited on 26th June 2024).

[83] Terrier Team. *Evaluation of Experiments in Terrier*. URL: https://github.com/terrier-org/terrier-core/blob/5.x/doc/evaluation.md (visited on 26th June 2024).

[84] Terrier Team. *jtreceval*. URL: https://github.com/terrierteam/jtreceval (visited on 26th June 2024).

[85] NIST. *trec_eval*. URL: https://github.com/usnistgov/trec_eval (visited on 26th June 2024).

[86] PyTerrier Team. *Datasets - PyTerrier 0.10.1 documentation*. URL: https://pyterrier.readthedocs.io/en/latest/datasets.html#pyterrier.datasets.Dataset (visited on 20th May 2024).

[87] Nick Craswell et al. *Overview of the TREC 2020 deep learning track*. 2021. arXiv: 2102.07662 [cs.IR].

[88] Christophe Van Gysel. *pytrec_eval*. URL: https://github.com/cvangysel/pytrec_eval (visited on 26th June 2024).

[89] Ollama. *llama3:instruct - Meta Llama 3: The most capable openly available LLM to date*. URL: https://ollama.com/library/llama3:instruct (visited on 20th May 2024).

[90] Amith Koujalgi. *ollama4j*. URL: https://github.com/amithkoujalgi/ollama4j (visited on 26th June 2024).

[91] P. Vaswani. *NPL Collection*. University of Glasgow. Also known as the VASWANI collection. 1970. URL: https://ir.dcs.gla.ac.uk/resources/test_collections/npl/.

[92] SQLite Team. *Architecture of SQLite*. URL: https://www.sqlite.org/arch.html (visited on 20th May 2024).

[93] OpenAI. *tiktoken: Fast BPE tokeniser for use with OpenAI's models*. URL: https://github.com/openai/tiktoken (visited on 20th May 2024).

[94] Ollama. *ollama-python: Ollama Python Library*. URL: https://github.com/ollama/ollama-python (visited on 20th May 2024).

[95] LangChain AI. *LangGraph: Building language agents as graphs*. URL: https://github.com/langchain-ai/langgraph (visited on 20th May 2024).

[96] LangChain. *LangSmith*. URL: https://www.langchain.com/langsmith (visited on 4th June 2024).

[97] Anastasia Krithara et al. 'BioASQ-QA: A manually curated corpus for Biomedical Question Answering'. In: *bioRxiv* (2022). DOI: 10.1101/2022.12.14.520213. eprint: https://www.biorxiv.org/content/early/2022/12/16/2022.12.14.520213.full.pdf. URL: https://www.biorxiv.org/content/early/2022/12/16/2022.12.14.520213.

[98] Hugging Face. *Xtrah/filtered-rag-mini-bioasq-80-20 · Datasets at Hugging Face*. URL: https://huggingface.co/datasets/Xtrah/filtered-rag-mini-bioasq-80-20 (visited on 23rd June 2024).

[99] OpenAI. *OpenAI Platform, Models: gpt-4o*. URL: https://platform.openai.com/docs/models/gpt-4o (visited on 5th June 2024).

[100] United Nations. *Climate Change*. URL: https://www.un.org/sustainabledevelopment/climate-change/ (visited on 26th June 2024).

[101] United Nations. *Economic Growth*. URL: https://www.un.org/sustainabledevelopment/economic-growth/ (visited on 26th June 2024).

[102] United Nations. *Infrastructure and Industrialization*. URL: https://www.un.org/sustainabledevelopment/infrastructure-industrialization/ (visited on 26th June 2024).

NTNU

Norwegian University of
Science and Technology