ation, and the use of knowledge bases separate from our vector index. To take advantage of these capabilties, a RAG system can be designed with multiple options to choose between depending on the content of the query. To handle this decision-making, we can introduce a *routing* system that intakes a query, decides which actions are best suited to seed a quality response, and activates the correct modules. Typically, this decision making is done by a sophisticated autogenerative model with a carefully designed prompt template that instructs the model to consider the query and choose between enumerated options. Conceptual questions to address include whether a query is sufficiently confusing and should be rewritten, whether multiple subprompts are required to retrieval all of the necessary information, or whether the query is about information in associated databases.

### 7.4.3 Retrieval and Generation

The primary goal of retrieval is to provide the generator with the context necessary to answer the query. However, this goal includes a significant assumption: the text chunk most semantically similar to the query (according to our embedding model) contains the needed information. If this assumption is not met, the RAG call will fail. Retrieval augmentations are concerned with improving the odds that the chosen documents are properly responsive to the user query and rank among the top few most effective additions to a RAG system.

#### 7.4.3.1 Reranking

A common issue with basic RAG is that the text chunks most responsive to a given query often do not appear at the top of the semantic similarity ranking. This retrieval imprecision is partly a result of the relatively small size of typical RAG embedding models. Performance could be improved by using larger and richer embedding models to embed the corpus, but this would be very costly due to the large size of many RAG corpora. A related issue, sometimes called the *lost in the middle* problem (Liu et al., 2023), is that LLMs are more likely to accurately digest in-context information located at the beginning or ends of prompts while being more likely to “lose” information in the middle of the prompt. Without this complication, you could improve performance simply by increasing the quantity of returned documents and hoping to capture the relevant information somewhere in your ranking – *lost in the middle* suggests that this approach will suffer from performance loss.

*Reranking* was developed as a compromise between these considerations. In reranking, a smaller embedding model is used for initial retrieval, and a large number of the top documents are returned – perhaps 20-30 documents – instead of just a few for basic RAG. These returned documents and the original query are then embedded again with a much larger and more semantically rich model, and the top-k chunks

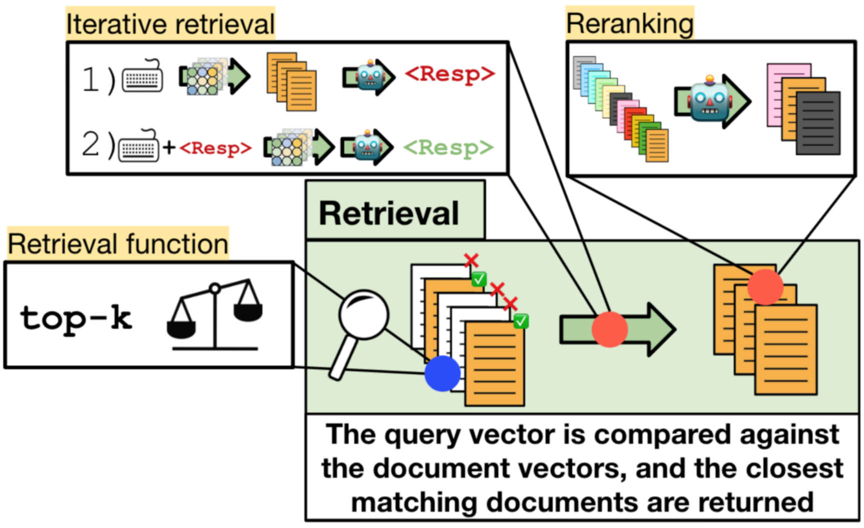
Iterative retrieval
1)
<Resp>
2)
+<Resp>
<Resp>
Reranking
Retrieval
Retrieval function
top-k
The query vector is compared against the document vectors, and the closest matching documents are returned

Fig. 7.5: An illustration of the different enhancements discussed for RAG retrieval. The retrieval function (Sect. 7.3) determines how many top documents to collect. Reranking (Sect. 7.4.3.1) uses a second, larger embedding model to rerank the retrieved documents in order to surface the most pertinent information. Iterative retrieval (Sect. 7.4.3.2) uses successive queries and the documents returned from each to answer multi-hop questions.

are reranked according to the new vectors. This allows you to cast a wide net with an inexpensive model and then perform a fine-grained ranking of the results with a superior model, resulting in a far more accurate choice of the top few documents. This will both ensure the generator uses the most relevant documents and pushes the most relevant to the very front of the list to avoid the *lost in the middle* problem. By only using the more expensive model on the returned documents, the higher cost is significantly mitigated while ensuring that relevant documents appear at the top of the ranking. Although the use of embeddings to rerank results is not new, this specific approach in the context of dense retrieval has been advocated by authors such as Ma et al. (2023b) and widely implemented in RAG development software, including LlamaIndex and LangChain (see the tutorial in Sect. 7.6).

#### 7.4.3.2 Iterative Retrieval

One stumbling block that can create failures in RAG querying is questions that require the synthesis of multiple pieces of information. This frequently occurs when a query asks for factual information related to a subject that is not explicitly mentioned but is only implied through a second relationship. An example, given in Shao et al. (2023):

```
Can you get Raclette in YMCA headquarters city?
```

A relevant information database, such as Wikipedia, could tell you that the YMCA is headquartered in Paris and that Raclette is a Swiss dish that is also popular in France. Still, you would have to get lucky to find a single chunk of text explicitly linking the YMCA headquarters to Raclette's availability. A primary RAG generator could answer this question if it was handed a document relating the YMCA to France and another relating France to Raclette. Nevertheless, since the prompt doesn't explicitly mention France, the second piece of information will be missed during retrieval. This style of question is referred to as a "multi-hop question", alluding to the fact that multiple reasoning steps are required for a correct response.

*Iterative Retrieval* is a process developed to answer multi-hop reasoning queries using dense retrieval. This process follows these steps:

1. A retrieval action is run for a query, and the top-scoring document is returned.
2. The query and document are passed to the generator for a response to the question
3. This response is combined with the original query, embedded with the indexing LLM, and used for another retrieval call against the vector index (generation-augmented retrieval). The top-scoring document is returned.
4. The documents returned in both steps 1) and 3) are combined with the original query and passed to the generator.
5. The generator LLM is asked if the documents and response satisfy the original query. If not, additional alternations of retrieval and generation are done.

### ! Practical Tips

In the second call to the vector index, the context likely contains hallucinated inaccuracies, but because it is a statement directly responsive to the query, the hope is that it will be semantically similar to the relevant documents in the RAG corpus, which contain the correct information. By allowing multiple calls to the vector index, *iterative retrieval* can answer the different portions of the question step by step. Methodologies along these lines were proposed by both Shao et al. (2023) and Feng et al. (2023).

Using the above example, Shao et al. (2023) shows the following results from their *iterative retrieval* system:

```
Question: Can you get Raclette in YMCA headquarters city?
```

```
*** Iteration 1 ***
-- Retrieved Knowledge --
(1) Title: YMCA Building (Racine, Wisconsin) Context: ... S-
ince the '50s, headquarters of the publishing house and the
bookstore are located in 11 rue de la Montagne Sainte-Genev-
iève 75005 Paris ...

-- Generation --
The YMCA headquarters is located in Paris, France. Raclette
is a traditional Swiss dish, made with melted cheese served
with potatoes, pickles, and other accompaniments. It is not
a trad- itional French dish, so it is unlikely to be found
in Paris. So the answer is No
```

Note that the generator attempted to answer whether Raclette is available in Paris but hallucinated the wrong conclusion. This generation and the original query are then embedded and used for another round of retrieval:

```
*** Iteration 2 ***
-- Retrieved Knowledge --
(1) Title: Raclette Context: Raclette is a Swiss dish, also
popular in France, based on heating cheese and scraping off
the melted part ...
(2) Title: YMCA Building (Racine, Wisconsin) Context: ... S-
ince the '50s, headquarters of the publishing house and the
bookstore are located in 11 rue de la Montagne Sainte-Genev-
iève 75005 Paris ...

-- Generation --
YMCA headquarters is located in Paris, France. Raclette is
a dish native to parts of Switzerland, but it is also popu-
lar in France. So it is likely that Raclette can be found
in Paris. So the answer is Yes
```

In the first iteration, the RAG call returns information only about the location of the YMCA, and then to answer the query hallucinates an answer about whether Raclette is popular in France. In the second iteration, the generation from iteration 1 is embedded and passed with the query, and because it contains a discussion of whether the dish is available in France, the returned top document also relates to the popularity of Raclette in the region. The final generation uses the retrieved information from both steps, and the correct answer is gleaned from the context.

#### 7.4.3.3 Context Consolidation

Once the documents have been selected, they must be added to a template to pass to the generator. The simplest approach is to concatenate each text chunk together along with the prompt and let the LLM sort out the details. However, this approach has downsides: it will fail if the combined text chunks are longer than the LLM context window size; it may miss crucial information if it is not optimally located (i.e., the *lost in the middle problem* discussed above); and a list of disparate and disconnected text chunks might be missing the connective tissue that relates their information to one another.

A number of approaches have been suggested for how to better synthesize the information contained in the top returned documents – this process is called *context consolidation*. A common technique is to use LLM calls to summarize the key facts in each text chunk, leading to a shorter context length for the generator (e.g. Chen et al., 2023b). LLMs can also be prompted to build a global summary of the whole corpus of returned documents by looking one-by-one at each chunk and iteratively updating a single summary (e.g. Xu et al., 2023), or by using a tree summarization approach such as the one implemented in LlamaIndex[7] (e.g. Liu, 2022). Processing the retrieved context from a disconnected series of text snippets into a more coherent and self-consistent document can improve outcomes: across a range of NLP tasks, Xu et al. (2023) showed that prompt compression via summarization both reduced average perplexity (i.e. improved response accuracy) and greatly reduced the length of the input context (reducing the length to as low as 6% in some cases) compared to simply concatenating returned documents in the prompt context.

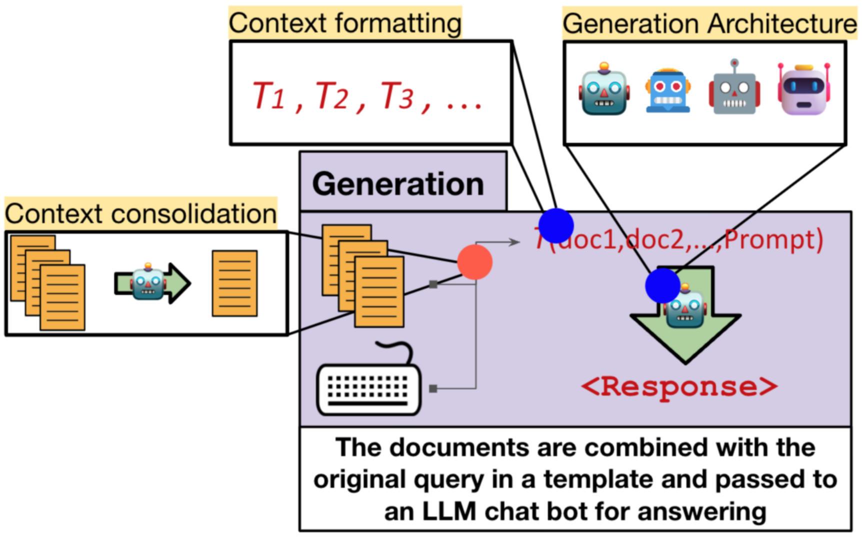

Fig. 7.6: An illustration of the different enhancements discussed for RAG generation. Context consolidation (Sect. 7.4.3.3) comprises methods for distilling the information from multiple retrieved documents into a single document before the generation call. Context formatting (Sect. 7.3) involves choosing an appropriate generation prompt template to suit the needs of the RAG system. Finally, the architecture of the LLM chatbot chosen for generation can be optimized through model selection or even fine-tuning (Sect. 7.3).

[7] https://docs.llamaindex.ai/en/latest/examples/response_synthesizers/tree_summarize.html

Table 7.1: Summary of retrieval-augmented generation features

| Stage | Description | Optimizations | Confounders | Enhancements |
|---|---|---|---|---|
| Indexing | Converting documents into segments of text, embedding the segments, and storing them in a vector-index. | • Text normalization<br>• Table parsing<br>• Chunking strategy<br>• Metadata labels<br>• Choice of embedding LLM<br>• Choice of vector index | • Data contained in PDF tables<br>• Data contained in structured databases<br>• Documents contain technical language unfamiliar to the embedding model | ⇒ Pre-process with LlamaParse<br>⇒ Automated generation of database search queries<br>⇒ Fine-tune the embedding model |
| Querying | User inputs a question that can be answered with information in the documents, and the question in transformed by the embedding model. | • Enhanced querying strategies | • Small variations in query wording can lead to diverse outcomes<br>• Low relevance of returned documents<br>• Complex queries require multiple pieces of information to answer<br>• Different queries require different querying strategies | ⇒ Query rewriting<br>⇒ Query-to-document<br>⇒ Subquery generation<br>⇒ Routing |
| Retrieval | The vector-index is searched with the embedded query, and the most semantically-similar documents (as determined by cosine-similarity) are returned. | • Quantity of top documents to return<br>• Re-ranking top documents | • Most relevant documents not ranking near top<br>• Multi-hop questions cannot be answered with one document | ⇒ Reranking<br>⇒ Iterative retrieval |
| Generation | The query and the retrieved documents are passed to a chat bat which produces an answer based on the information returned by the search. | • Choice of generator LLM<br>• Generation template and system prompts<br>• How to consolidate documents | • Generator output not faithful to returned documents<br>• Information in certain documents getting ”lost in the middle” | ⇒ Change prompting template<br>⇒ Context consolidation |

### 7.4.4 Summary

We have described many necessary considerations when building a RAG system (7.3), and enumrated useful enhancements (or modules) that can boost RAG performance in key areas. Table 7.1 summarizes these optimizations and enhancements. Considering the enormous variety of RAG components, there is no one-size-fits-all approach to constructing a RAG system. Each enhancement targets a specific RAG weakness for improvement, but not every RAG application will be well suited for each possible enhancement.

The utility of individual RAG modules needs to be considered in the context of a given RAG system's use case and balanced against the downsides of adding more complexity to your model. These downsides may include greater implementation and testing effort, the addition of more potential points of failure, decreased application latency, and increased computational and financial cost of more calls to LLM agents.

The surest approach is a prudent selection of potential candidate enhancements and a careful trial-and-error process that tests the impact of individual modules on RAG performance. In the next section, we will discuss approaches to performance testing and detail a number of practical considerations required for ensuring your RAG system is performant.

## 7.5 Evaluating RAG Applications

A key aspect of building a successful RAG solution is an effective evaluation strategy. With the potential for so many modules in more complex RAG applications, there are often multiple evaluation targets to focus on. Similarly, evaluation methods are typically selected according to the application-specific requirements.

In general, the evaluation of RAG leverages preexisting quality and capability measurement metrics applied to either the retriever, generator, or end-to-end output. With multiple evaluation targets, evaluating and leveraging suitable metrics and criteria according to the evaluation target(s) and when evaluation and optimization occur in the application development life-cycle is important. For instance, evaluating the generator component on its own early in the development life-cycle may be a case of premature optimization since generator performance is heavily influenced by the quality of the context provided by the retriever, which may not have been optimized yet. Therefore, sequencing evaluation steps are as critical as selecting suitable metrics and capabilities for measurement (Hoshi et al., 2023).

Indeed, practical guidance provided within the LlamaIndex documentation suggests that an appropriate way to handle this complexity is to begin with an end-to-end evaluation workflow, and as insights into limitations, edge cases, and fail-states are identified, pivoting to the evaluation and optimization of causal components systematically is a good strategy (Liu, 2022). Assuming an effective evaluation workflow, we will explore the most common evaluation aspects that have emerged for RAG applications.

There are, in essence, seven key aspects commonly leveraged for evaluating RAG applications (Gao et al., 2024). Three can be considered *quality metrics*, and four *system capabilities*:

- Quality metrics
  1. Context relevance
  2. Answer faithfulness
  3. Answer relevance
- System capabilities
  1. Noise robustness
  2. Negative rejection
  3. Information integration
  4. Counterfactual robustness

In the next two sections, we define these aspects, with insights into how and where they are evaluated within a typical RAG framework. Available software tooling and frameworks that enable specific evaluations will also be highlighted where possible.

## 7.5.1 RAG Quality Metrics

This section describes the context relevance, answer faithfulness, and answer relevance RAG metrics, with a summary illustration shown in Fig. 7.7.

### 7.5.1.1 Context Relevance

*Context relevance* measures the effectiveness of the RAG retriever in returning relevant context while passing over irrelevant context. This is typically measured based on a number of preexisting metrics. Some metrics simply look at all retrieved contexts independent of their relevance ranking and are referred to as *rank-agnostic metrics*, while others take context relevance ranking into account and are referred to as *rank-aware metrics*.

**Common metrics for measuring context relevance**

*Recall* is a rank-agnostic metric that focuses simply on the number of relevant contexts retrieved and the total number of relevant contexts present within the retrieval corpus. The recall value is calculated as the proportion or percentage of relevant contexts retrieved relative to the total number of relevant contexts within the retrieval corpus. Since the maximum number of relevant contexts returned is often set within a retrieval setting, a common modification of the recall calculation is $recall@K$, where $K$ is the fixed number of contexts retrieved.

**! Practical Tips**

Recall is a good context relevance metric when the rank of returned context is of little impact, such as when short contexts are being used in the generation step or a reranker is being employed downstream (Sect. 7.4.3.1). However, retrieved-context recall may be misleading in this setting when the length of the generator prompt context is susceptible to the *lost in the middle* problem (Liu et al., 2023). Measuring recall in context relevance requires labeled data, typically in the form of query -> relevant document(s) pairs. However, innovations in using highly-capable LLMs to sem-automate recall calculations have been proposed in practical settings. For example, a prompt in the form of ``is all of the required information to answer {query} available in {retrieved_context}" will allow the LLM to reason over the context conditioned on the query itself.

*Precision* is another rank-agnostic metric, that measures the proportion or percentage of retrieved documents that are relevant to the query. For example, if the retriever returns 100 contexts, but only 60 of these are relevant to the query, then the precision

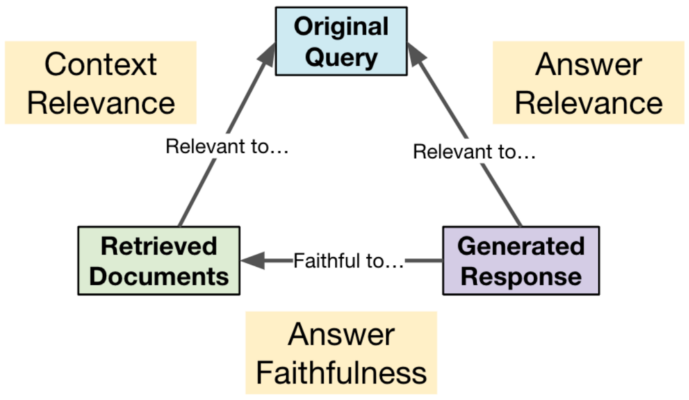

Fig. 7.7: Graphical illustration of the relationships between the three highlighted RAG quality metrics. Context relevance measures how relevant retrieved documents are to the original query, answer relevance measures how relevant the generated response is to the original query, and answer faithfulness measures how faithful the generated response is to the retrieved documents.

will be 0.6 (or 60%). This metric provides insight into how noisy the retriever is, allowing developers to focus on increasing retrieval precision or implementing controls downstream to handle the extra noise in contexts. Such downstream controls include postretrieval reranking (Sect. 7.4.3.1), context consolidation (Fig. 7.6), and simple filter rules. Similar to recall, precision can also be measured as a function of a fixed value for $K$ to give $precision@K$.

**! Practical Tips**

Traditionally, labeled query/relevant document pairs are used to evaluate retrieved-context precision. However, as in the case of recall, highly capable LLMs are increasingly used for this task, with a prompt of the form:

```
``how many of the search results below are relevant to the
{query}? {retrieved_context}".
```

As above, the LLM is expected to reason around specific contexts and their relevance to the query, so the extent to which the *lost in the middle* problem impacts this metric calculation should also be carefully evaluated.

---

*Mean Reciprocal Rank* (MRR) is a measure of where the most relevant context is within a rank-ordered set of retrieved contexts, on average, across a set of queries. Interpretation of this metric follows the logic that if $MRR = 1$, then for the set of queries evaluated, the most relevant context is always returned by the retriever in the first position for relevance, while $MRR = 0$ indicates that either the retriever is returning no relevant context, or that the most relevant context for each query is always returned in the last position in relevance rank.

**! Practical Tips**

In practice, MRR typically falls somewhere between these extremes. MRR is particularly useful in measuring retrieval effectiveness in RAG applications where $K = 1$ with respect to the number of retrieved contexts passed to the generator since it is a direct measure of how effectively the retriever is at retrieving the most relevant context in the first position. When used in conjunction with *hit rate* (see below), some of the ambiguity around whether a low MRR is because limited relevant context is being retrieved vs relevant context being retrieved, but with low relevance rank, can be resolved. As an example, for an evaluation with a higher *hit rate* value than *MRR* value is indicative of poor relevance ranking in the retriever, allowing for practical remediation, such as the introduction of a reranker prior to generation.

---

*Hit Rate* is a metric that measures the proportion of queries for which the most relevant contexts are retrieved. Practically, this metric is usually limited to measurement on an application-appropriate value for $K$ retrieved contexts assessed. The simplicity of this metric allows for straightforward interpretation and can enable powerful insights into application inefficiencies when combined with other metrics such as MRR.

*Normalized Discounted Cumulative Gain* (nDCG) is a rank-aware metric that measures the relevance of retrieved contexts normalized by their retrieval order. Conceptually, nDCG measures a retrieved-context's usefulness (gain) based on its retrieved order. Practically, nDCG is calculated by accumulating gain top to bottom within the retrieved contexts, importantly applying a discount (normalized) to the cumulative gain for highly relevant contexts with low rank order in the list of retrieved contexts. nDCG makes several assumptions, which should be assessed when using the metric and interpreting its meaning with respect to RAG outcomes. In particular, the "discount" portion of the metric calculation assumes that more relevant documents should occur higher in the retrieved context order, thus highly relevant contexts occurring lower in the retrieved context order are penalized through the normalization step. This may not always be an appropriate measure if, for example, all retrieved contexts are used in the generation stage since those high-relevance, low-order contexts will still be provided to the generator.

**! Practical Tips**

nDCG can be interpreted as a score of how closely the retrieved contexts align to a perfectly ordered list of relevant contexts, where the most pertinent contexts are at the top of the ranked list and relevance declines top to bottom. Thus, nDCG provides clear insights into how well-ranked retrieved contexts are. A low cumulative gain score can indicate the need for better ranking of contexts or the need for better recall in the retriever if few relevant contexts and many irrelevant contexts cause the low score. nDCG is also a helpful metric when evaluating the generator in RAG, where the relevance ranking is simply evaluated on a set of possible responses to a given query rather than retrieved contexts.

#### 7.5.1.2 Answer Faithfulness

As discussed in Sect. 6.1, hallucination is a common weakness of LLMs. This property of even the most advanced LLMs significantly increases the complexity of leveraging them in production settings where honesty is critically important (Sect. 5.1.2). Indeed, RAG is largely an innovation on top of LLMs that attempts to ameliorate this challenge. However, as with all systems leveraging probabilistic mechanisms for generating outputs, how well such innovations ameliorate the hallucination issue needs to be evaluated during development and on an ongoing basis as the application evolves through its development life cycle.

*Answer faithfulness*, also sometimes called *groundedness*, is a quality measure used to ensure that the responses from the generator are as factually accurate as possible, given the retrieved context. Essentially, this measures how well the generator can ground its responses in the knowledge the retriever provides. Answer faithfulness can be evaluated using several metrics that fall into two categories, *lexical-based* and *model-based*.

**Lexical-based metrics**
Lexical-based metrics for answer faithfulness aim to measure the extent to which the tokens in the response overlap with or are grounded in the retrieved context or knowledge provided to the generator. In practice, these approaches have been superseded by the model-based metrics discussed later, but we include them here for completeness. Some of the most commonly used lexical-based metrics are as follows:

- **Knowledge-F1** or K-F1 measures the F1 overlap in tokens within the generated answer and the retrieved context provided to the generator. This metric has limitations where the span length from the generator differs significantly from the retrieved context. For example, consider a generator that returns short, concise answers to user queries, while the retrieved context is typically longer than the answer span. In such instances, the response may be penalized on recall, even though the precision of the response tokens is high and the response is otherwise correct with respect to the user's needs (Adlakha et al., 2023).
- **Knowledge-precision** or K-precision, measures the proportion of tokens found in the generator's response that are also present in the retrieved context. This metric provides a valuable measure of answer faithfulness without the potential bias that the asymmetry of the measurement introduces in K-F1. This asymmetry arises because the factually relevant tokens within a generator response can only be evaluated as equal to or a subset of the facts present within the retrieved context.

**Model-based metrics**
As mentioned, lexical-based evaluation metrics for answer faithfulness have been largely superseded by model-based approaches in practice. This is due to the difficulty in generating labeled contexts through annotation and the low correlation that some of these metrics have with human-level judgment (Adlakha et al., 2023), but perhaps more significantly, the ever-improving competency of LLMs for such tasks. While not yet a panacea (e.g., Wang et al. (2023c)), the most capable LLMs have been shown to provide excellent correlation in the evaluation of answer faithfulness with human-based judgment approaches to the same evaluation task (Adlakha et al., 2023). This correlation lends promise to using highly capable LLMs to improve the efficiency of evaluating answer faithfulness in RAG.

One of the earliest model-based approaches for evaluating answer faithfulness was $Q^2$ (Honovich et al., 2021).

Calculation of this metric begins first with extracting informative spans in the answer. This is typically done using some form of Named Entity Recognition

(NER). As an example, consider the following hypothetical answer response from a RAG system:

```
"Red wine is acidic"
```

Using an NER system to extract informative spans mapping to named entities and noun phrases, the informative span `Red wine` is derived. These informative spans and the responses from which they were identified are then passed to a large generative model to generate relevant question(s) conditioned on the informative span and the original response. In the example above, given `Red wine` as the informative span and the response above, a generated question could be:

```
"What is acidic?"
```

These generated questions are then answered using a fine-tuned QA model.

The answer faithfulness of the original generative system is similar to the informative spans extracted from the original response and the answers to the generated questions. If informative spans are extracted from the original response and the answers to the generated questions are a perfect match, a $Q^2$ score of 1 is given. If there is no perfect match, then similarity in the informative span from the response and the answer to the generated question is determined using natural language inference (NLI). In this NLI step, entailment receives a score of 1, while contradictions receive score of 0. QA responses with no answer take on a token-level F1 score. The overall system-level $Q^2$ score is then the average across all answer pairs (Honovich et al., 2021).

More recently, however, model-based approaches have changed to capitalize on the evermore sophisticated generative LLMs available to provide more consolidated measures of answer faithfulness (i.e., the need to have distinct models for question generation, NER, and question answering as in Honovich et al. (2021) is significantly decreased when using only GPT-4, for example). The general approach is very similar to that described for $Q^2$, if much less modular since GPT-4 is more capable of leveraging its natural language understanding to complete the task more comprehensively.

Introduced in Adlakha et al. (2023), `LLMCritic` leverages a simple prompting approach to enable GPT-4 to evaluate whether the response answer from a RAG system contains only information/fact-claims that are either present within or can be inferred from the retrieved context. An example prompt template for this task given by these authors is shown below:

```
System prompt: You are CompareGPT, a machine to verify the
    groundedness of predictions. Answer with only yes/no.

You are given a question, the corresponding evidence and a
    prediction from a model. Compare the "Prediction" and the "
    Evidence" to determine whether all the information of the
    prediction in present in the evidence or can be inferred from
     the evidence. You must answer "no" if there are any specific
     details in the prediction that are not mentioned in the
    evidence or cannot be inferred from the evidence.

Question: {Question}
Prediction: {Model response}
Evidence: {Reference passage}

CompareGPT response:
```

Listing 7.1: ”Example prompt for assessing Answer Faithfulness”

Here, GPT-4 is prompted to verify whether all of the information within the RAG answer is present or can be inferred from the evidence (retrieved context). In general, when using LLMs to evaluate answer faithfulness, the formula for calculating the metric is:

$$\text{Faithfulness} = \frac{|\text{\# of facts in answer that can be inferred from retrieved context}|}{|\text{Total \# of facts in answer}|} \tag{7.2}$$

As the capabilities of LLMs have increased, their use for the calculation of these kinds of RAG evaluation metrics has increased. Indeed, RAG evaluation frameworks/tools such as `LlamaIndex` (Liu, 2022), `TruLens` (Reini et al., 2024) and `Ragas` (Es et al., 2023) have various methods and approaches like this available for use, some of which we will see in action in the tutorial section.

#### 7.5.1.3 Answer Relevance

*Answer relevance* is another important metric for evaluating the quality of a RAG system. It answers the question “how relevant is the answer generated to the user query and the retrieved context?”. The most common approaches to calculating this metric also leverage highly capable LLMs. In this instance, the LLM is prompted with the generator’s answer, the context used in generating that answer and instructions to generate $N$ synthetic questions based on this information. These questions are then semantically compared to the original user-query, which results in the reference answer used to generate the synthetic questions. Answer relevancy is measured as the mean “semantic similarity” between the original user query and $N$ synthetic questions. As such, RAG answers that prompt the generation of questions that are most semantically aligned to the original user query will result in higher answer relevancy scores, and *vice versa*. More specifically, the `Ragas` framework calculates this metric as:

$$\text{Answer Relevance} = \frac{1}{N}\sum_{i=1}^{N} sim(E_{g_i}, E_o) \tag{7.3}$$

where $N$ is the number of synthetic questions generated by the evaluation LLM, $E_{g_i}$ is the embedding of the $i^{th}$ synthetic/generated question, $E_o$ is the embedding of the original query, and $sim$ is an appropriate measure of similarity between the two (e.g., cosine similarity).

While there are various approaches for evaluating answer relevance, including comparisons to ground-truth answers, etc., the LLM evaluator approaches are becoming dominant because of their ability to overcome the often costly and complex task of defining ground truth for such expressive applications.

### 7.5.2 Evaluation of RAG System Capabilities

In addition to deriving direct quantitative metrics to evaluate the performance of RAG systems in terms of retrieval quality, answer faithfulness, and relevance, several additional capabilities must be evaluated. The capabilities are evaluated to assess the general performance, versatility, and reliability of the LLM generator used within the RAG system (Chen et al., 2023a). This evaluation approach is necessary to understand how the retrieved context and the prompt augmentation impact the generation process within the LLM. As an example, it is essential to understand the extent to which the LLM generator integrates nonparametric knowledge – the contextual information in the prompt – over its parametric knowledge – the information embedded in the generator LLM's parameters during pre-training. If the generator overrides its parametric knowledge with the nonparametric retrieved context and said context is erroneous, then as a developer, we understand that we may need to modify our prompt or improve our retrieval precision to ensure that the generated responses are as faithful to ground truth as possible.

In the following sections, we will detail the four most popular and commonly used RAG capability evaluations, with illustated examples taken from Chen et al. (2023a). A graphical summary of the metrics is shown in Fig. 7.12.

#### 7.5.2.1 Noise Robustness

In simple terms, *noise robustness* measures the LLM generator's ability to leverage only the useful information within noisy retrieved-context documents. Fig. 7.8 illustrates this property of a RAG system. Effectively, the aim is to understand how well the LLM generator can navigate irrelevant context and still respond with the correct answer to the user's query.

**! Practical Tips**

Assessing the RAG LLM's ability to handle noisy contexts relies on ground-truth knowledge of positive and negative contexts relative to a set of generated question-answer pairs. The typical approach is to pair a relevant document with a random

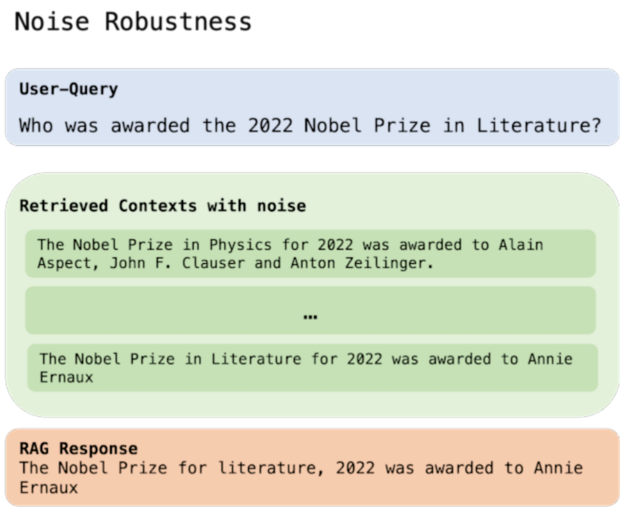
Noise Robustness
User-Query
Who was awarded the 2022 Nobel Prize in Literature?
Retrieved Contexts with noise
The Nobel Prize in Physics for 2022 was awarded to Alain Aspect, John F. Clauser and Anton Zeilinger.
...
The Nobel Prize in Literature for 2022 was awarded to Annie Ernaux
RAG Response
The Nobel Prize for literature, 2022 was awarded to Annie Ernaux

Fig. 7.8: An example of *noise robustness* in a RAG response. Here we can see that even though the retrieved contexts contain noise (e.g., information about the Nobel Prize in Physics rather than literature), the generator can still respond with the correct answer to the user query.

sampling of negative contexts at an appropriate ratio. These "retrieved" contexts can then be passed to the LLM generator, and its answer can be compared to the original answer generated as part of a question-supporting information-answer triplet dataset. An accuracy-based metric such as exact match (EM) calculates the LLM's ability in this aspect (Chen et al., 2023a).

---

### 7.5.2.2 Negative Rejection

In *negative rejection*, the RAG application refuses to answer a given user query in the instance where none of the retrieved contexts contain the relevant information necessary to do so. In Fig. 7.9,we can see that none of the contexts shown contain the relevant facts to answer the question `Who was awarded the 2022 Nobel Prize in Literature?`. Only contexts relevant to the Nobel Prize in Physics were retrieved. Evaluation of this capability in RAG enables developers to optimize application behavior in the event that the available knowledge sources do not allow faithful or factual responses, such as implementing more stringent system instructions for such settings.

**! Practical Tips**

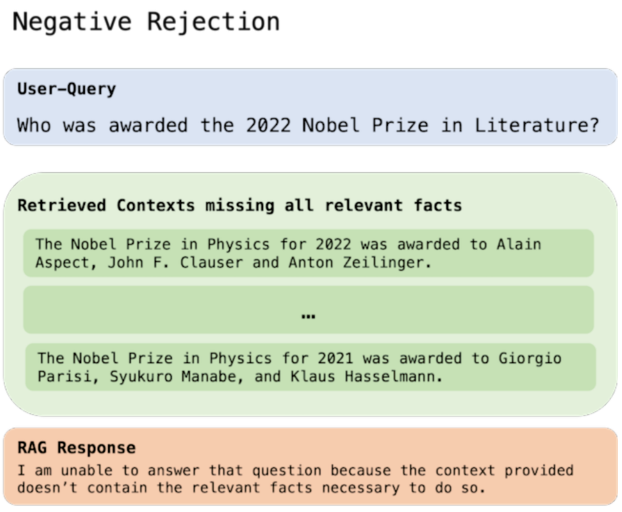
Negative Rejection
User-Query
Who was awarded the 2022 Nobel Prize in Literature?
Retrieved Contexts missing all relevant facts
The Nobel Prize in Physics for 2022 was awarded to Alain Aspect, John F. Clauser and Anton Zeilinger.
...
The Nobel Prize in Physics for 2021 was awarded to Giorgio Parisi, Syukuro Manabe, and Klaus Hasselmann.
RAG Response
I am unable to answer that question because the context provided doesn't contain the relevant facts necessary to do so.

Fig. 7.9: A RAG generator is considered capable of negative rejection in instances where it does not provide an answer to the user query when the necessary information is not provided in any retrieved context. As illustrated, none of the displayed contexts contain information on who the recipient of the 2021 Nobel Prize for Literature was. Therefore, the generator does not answer this question.

To assess negative rejection in practice, again, a set of question-answer pairs is generated, along with supporting information for arriving at the answer. By only sampling contexts from negative documents (i.e., those not generated along with the generated question-answer pairs), the RAG LLM's negative rejection ability can be measured as a function of the number of times the model correctly answers with the rejection-specific content, which is generally specified through instructions to the LLM. The rejection rate metric can be calculated to understand what proportion of correct vs incorrect negative rejections occur when querying the RAG system.

#### 7.5.2.3 Information Integration

*Information integration* in RAG refers to the LLM generator's ability to integrate information across multiple context documents to synthesize a correct answer to a complex question. Suppose the user query contains more than one subquestion or multiple pieces of information distributed across contexts. In that case, it is important that the RAG system systematically integrate the relevant context for each constituent information unit to be provided in the response. As we see in Fig. 7.10, the user query asks for both the 2021 and 2022 winners of the Nobel Prize in Physics. The external information required to answer this question is also distributed across two separate context documents. The RAG response correctly integrates these contexts to provide a correct response.

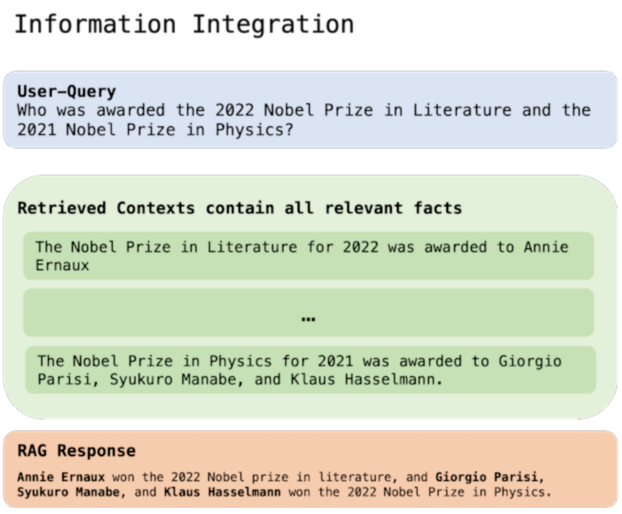
Information Integration
User-Query
Who was awarded the 2022 Nobel Prize in Literature and the 2021 Nobel Prize in Physics?
Retrieved Contexts contain all relevant facts
The Nobel Prize in Literature for 2022 was awarded to Annie Ernaux
...
The Nobel Prize in Physics for 2021 was awarded to Giorgio Parisi, Syukuro Manabe, and Klaus Hasselmann.
RAG Response
Annie Ernaux won the 2022 Nobel prize in literature, and Giorgio Parisi, Syukuro Manabe, and Klaus Hasselmann won the 2022 Nobel Prize in Physics.

Fig. 7.10: A RAG generator is said to successfully integrate information, demonstrating the ability to leverage information from multiple contexts/documents to answer complex questions. In this example, the user query asks for both the 2021 and 2022 winners of the Nobel Prize in Physics and successfully integrates the relevant context to provide the correct answer.

## ! Practical Tips

Again, the evaluation of information integration in practice relies on the generation of question-supporting information-answer triplets. However, an additional step in the test data generation is carried out to create additional aspects to the question's answer, such as combining two questions, their answers, and supporting information, such that the supporting information required to answer the more complex question is distributed across more than one context document. Successful information integration is also determined using an accuracy metric such as EM, where the RAG-generated response is directly compared to the originally generated answer to the question(s) (Chen et al., 2023a).

#### 7.5.2.4 Counterfactual Robustness

Factual errors are common in external knowledge bases commonly used in RAG applications. As such, it is important to evaluate the ability of the RAG generator to identify these falsehoods in retrieved contexts – this is called *counterfactual robustness*. Since identifying errors in the retrieved context relies entirely on the LLM generator's parametric knowledge, this aspect of the RAG application can be challenging to evaluate where knowledge within the application domain is either not represented or underrepresented in the chosen LLM. While domain adapting or fine-tuning LLMs is always an option, it is expensive and ultimately undercuts some of the advantages of RAG. However, many domain-fine-tuned LLMs have emerged in the open-source space, and as such, generating domain-relevant test data for this purpose is becoming increasingly viable.

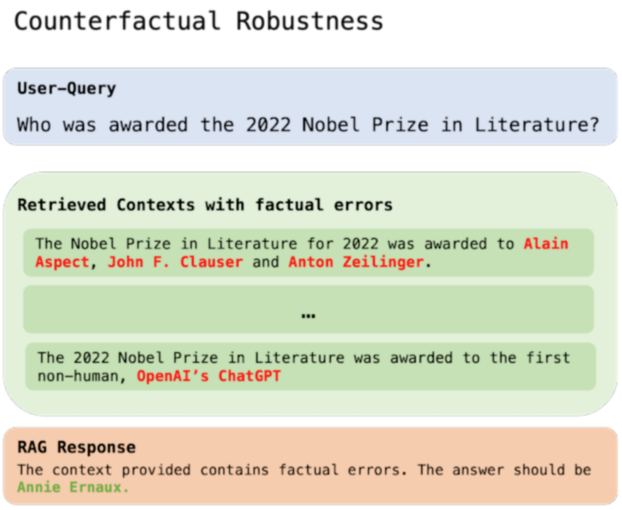
Counterfactual Robustness
User-Query
Who was awarded the 2022 Nobel Prize in Literature?
Retrieved Contexts with factual errors
The Nobel Prize in Literature for 2022 was awarded to Alain Aspect, John F. Clauser and Anton Zeilinger.
...
The 2022 Nobel Prize in Literature was awarded to the first non-human, OpenAI's ChatGPT
RAG Response
The context provided contains factual errors. The answer should be Annie Ernaux.

Fig. 7.11: Counterfactual robustness is the generator's ability to detect and highlight in its response that the context provided contains factual errors. This ability is grounded in the generator LLM's parametric knowledge, which can mean that it is challenging to assess when using an LLM without domain-relevant knowledge for a given application, or when the application relies on knowledge that arose after the LLM's knowledge cutoff.

**! Practical Tips**

To test this capability, the generator LLM is prompted to generate questions and answers solely on its parametric knowledge. This means that the LLM is prompted to generate questions to which it already knows the answers, independent of the in-

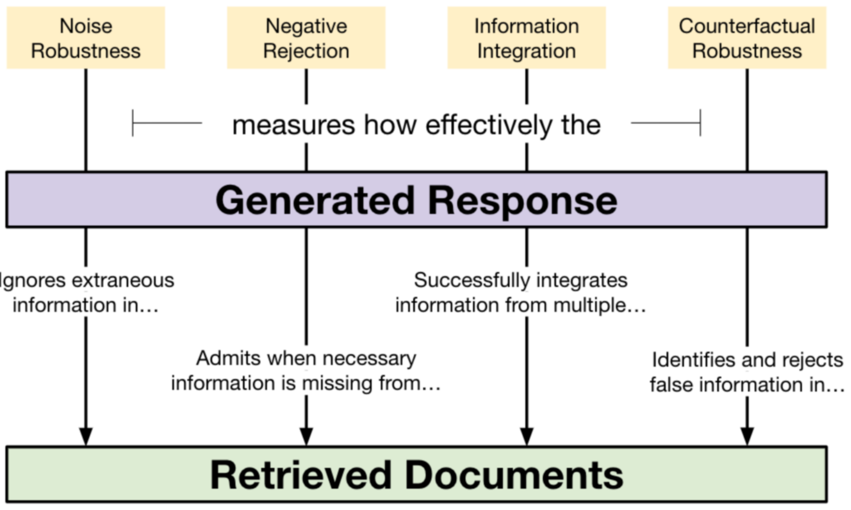
Noise Robustness
Negative Rejection
Information Integration
Counterfactual Robustness
measures how effectively the
Generated Response
Ignores extraneous information in...
Admits when necessary information is missing from...
Successfully integrates information from multiple...
Identifies and rejects false information in...
Retrieved Documents

Fig. 7.12: Graphical illustration of the properties measured by each RAG system capability metric. Each of the four metrics determines how well the generated response understands and correctly responds to the properties (positive or negative) of the retrieved documents.

formation in the external knowledge base. To establish the counterfactual supporting information, the generated answers are first verified to be correct, and subsequently, supporting information is retrieved from external knowledge. Supporting contexts are then manually modified to replace correct supporting facts with incorrect supporting facts. This dataset assesses two key metrics: the *error detection rate* and the *error correction rate*.

---

The error detection rate relies on the LLM generator responding with specified content in the event that supporting contexts contain factual errors. This metric indicates how well the LLM can evaluate the factuality of the retrieved contexts against its parametric knowledge. Similarly, the error correction rate measures how frequently the LLM generator can provide the correct answer despite the supporting information containing errors.

### 7.5.3 Summarizing RAG Evaluation

The evaluation of RAG applications is complex, thanks to the multiple functional components with fuzzy abstractions. Ensuring that the vector database's indexing is optimal, the chunking used for external knowledge documents, the retrieval relevance and ordering, and the final generation step are all concerns of the RAG application developer. Thankfully, much research and innovation has occurred to simplify and streamline this complex process. From conceptually useful frameworks, such as the RAG Triad from the TruLens team, to practically efficient implementation frameworks such as LlamaIndex, much of this complexity is simplified for users of these tools and frameworks to enable rapid prototyping and robust production-grade development.

In Chapter 8, we will explore the operational concepts, frameworks, tools, and challenges of using LLMs in production, much of which will apply to RAG application development. However, before we explore these issues in depth, we present a tutorial of RAG development and evaluation.

## 7.6 Tutorial: Building Your Own Retrieval-Augmented Generation System

### 7.6.1 Overview

Retrieval-augmented generation is a promising avenue for combining the semantic understanding capabilities of LLMs with the factual accuracy of direct source materials. In this chapter, we have discussed many aspects of these systems, from basic approaches to optimization to enhancements to the core RAG functionality and methods for evaluating RAG performance. This section will present a practical, hands-on example of building and augmenting a RAG system using a popular open-source RAG application.

In recent years, a number of such open-source libraries have been developed and released. These RAG libraries present high-level functions for implementing many different RAG approaches and allow great customization for constructing one's own system. For this tutorial, we will use LlamaIndex (Liu, 2022) to build a RAG system, experiment with a few of the many tunable parameters, and evaluate the system's performance.

**Goals:**

- Demonstrate how to set up a basic RAG application with low effort using LlamaIndex.
- Explore the wide range of possibilities for customizing and improving a RAG application.
- Evaluate context relevance, answer relevance, and groundedness for a RAG application.

Please note that this is a condensed version of the tutorial. The full version is available at `https://github.com/springer-llms-deep-dive/llms-deep-dive-tutorials`.

## 7.6.2 Experimental Design

This exercise walks through the steps to build an experimental RAG application. For our document corpus, we use the OpenAI terms and policies, taken from `https://openai.com/policies`, as they appeared in late January 2024. The tools we incorporate in our application are as follows:

- **LlamaIndex**: This framework handles document parsing, indexing, searching, and generation using an extensive catalog of modules that can be easily incorporated into a single RAG framework. Integrations with Hugging Face allow for great customization in the choice of embedding and generation models. (Liu, 2022)
- **BAAI/bge-small-en-v1.5**: This small English variant of the Beijing Academy of Artificial Intelligence (BAAI) line of text-embedding models is highly performant in text-similarity tasks, yet is small enough (33.4M parameters) to fine-tune easily.
- **OpenAI ChatGPT**: Throughout the tutorial, we use the `gpt-3.5-turbo` and `gpt-4` models from OpenAI as our generators. They will also provide a comparison of the output of our RAG systems.

The first step is to load each document from the OpenAI terms and conditions into LlamaIndex. Next, we choose a chunking strategy and an embedding model to generate our vector index. After this process is finished, LlamaIndex makes it easy to begin querying the RAG database using `gpt-3.5-turbo` as the generator LLM.

Starting from our initial basic application, we then go on to explore many of the design choices and improvements that can be made. Of these enhancements, the most notable are fine-tuning the embedding model and adding a document reranker. We continually compare results to see how our application responds as we introduce new ideas. Finally, we conduct a thorough evaluation of our end-stage RAG application against an earlier iteration without a fine-tuned embedding model.

## 7.6.3 Results and Analysis

There are many different approaches to evaluation, but we will consider here only the three quality metrics given in Section 7.5.1:
They are:

1. Context Relevance: Is the retrieved context relevant to the query?
2. Answer Relevance: Is the generated answer relevant to the query?

3. Answer Faithfulness/Groundedness: Is the generated answer supported by the retrieved context?

These three metrics ensure that the query finds useful documents, that the generated response is faithful to the documents, and that the generated response answers the original query.

Looking first at context relevance, we construct a test set by generation questions from a large number of corpus chunks. We can then use RAG to retrieve documents from each generated question, and check whether the document it was derived from is selected. We measure this with two metrics: “mrr”, or mean reciprocal rank (= 1 divided by the rank of the correct document), and “hit_rate”, which is equal to 1 if the correct document is among the top 5 returned, or 0 otherwise (see Sect. 7.5.1.1). Looking at one example question:

```
Total MRR =  1.0 /  1
# Hits =  1.0 /  1
Expected ID =  ['f712129a-a58d-4e36-b62f-22ebfeda56a8']
Retrieved IDs =  ['f712129a-a58d-4e36-b62f-22ebfeda56a8',
                  '1f95b363-1002-4f2d-bf17-ab4842714072']
```

Listing 7.2: ”Context relevance example”

We can see that the expected document ID was first in the retrieval list, and thus MRR and # hits are both 1/1. Looking now to a sample of 50 validation QA pairs:

```
- Base model:
  - Total MRR =  36.5 / 50
  - # Hits =  42.0 / 50
- Fine-tuned model:
  - Total MRR =  40.0 / 50
  - # Hits =  46.0 / 50
```

Listing 7.3: ”Context relevance scores”

We see that the source document was returned in the majority of cases and was frequently (although not always) the top returned document, but the RAG system whose embedding model was previously fine-tuned on the OpenAI terms and conditions corpus does somewhat better.

Turning to answer relevance, we can ask whether the RAG pipeline produces a reasonable answer to our queries. Here, we submit a query, receive a response, and then ask GPT-4 if the query is responsive to the question. In this case, we obtain a simple True or False response. Here is a test case:

```
query = "How can individuals request corrections for factually
    inaccurate information about themselves in ChatGPT output?"
results = run_answer_relevance_eval(index, [query,])

Response:
Individuals can request corrections for factually inaccurate
    information about themselves in ChatGPT output by submitting
    a correction request through privacy.openai.com or by sending
     an email to dsar@openai.com. If the inaccuracy cannot be
    corrected due to the technical complexity of the models, they
     can request the removal of their Personal Information from
    'ChatGPTs output by filling out a specific form.

Relevant:
True
```

Listing 7.4: "Answer relevance example"

This response does indeed answer the query. Evaluating 50 samples from the validation set, we find:

```
- Base model: 47 / 50
- Fine-tuned model: 49 / 50
```

Listing 7.5: "Answer relevance scores"

Once again, we see a slight improvement from fine-tuning, this time in arguably the most important metric: responsiveness of the query to the question.

The final evaluation metric is answer faithfulness, or "groundedness", where we ensure that the generated responses are grounded in the context. For our models, the transformation from context to response is done by GPT-4 instead of our vector index, so we should expect good performance and little difference between the two models. As expected, both models perform well, with only a minor difference:

```
- Base model: 48 / 50
- Fine-tuned model: 49 / 50
```

Listing 7.6: "Groundedness scores"

A summary of our results is given in Table 7.2, along with two additional model configurations – the base and fine-tuned versions combined with reranking (return top 20 > reranked top 2). Reranking significantly boosts context relevance, increasing the number of captured hits to nearly 100% while marginally improving total MRR score. However reranking has actually decreased the metrics for answer relevance and groundedness. Why is unclear, but suggests that care must be taken when incorporating reranking modules – their utility must be validated and not just taken as granted.

Table 7.2: Summary of evaluation results (out of 50) on the TruLens triad of RAG evaluations for four model setups: base, fine-tuned, base + reranking, fine-tuned + reranking.

| **Model** | **Context Relevance** | | **Answer Relevance** | **Groundedness** |
|---|---|---|---|---|
| | MRR | # Hits | | |
| Base | 36.5 | 42.0 | 47 | 48 |
| FT | 40.0 | 46.0 | 49 | 49 |
| Base RR | 37.5 | 49.0 | 43 | 47 |
| FT RR | 40.7 | 49.0 | 47 | 48 |

### 7.6.4 Conclusion

In this exercise, we discussed the basic steps of setting up a minimally functional RAG application. We then test more advanced methods to improve on the results, and demonstrated how to appropriately evaluate the responses to ensure that the application works as intended. It is clear that tools such as LlamaIndex are extraordinarily powerful in their ability to enrich the knowledge of LLMs without requiring a great deal of effort or model training.

## References

Vaibhav Adlakha, Parishad BehnamGhader, Xing Han Lu, Nicholas Meade, and Siva Reddy. Evaluating correctness and faithfulness of instruction-following models for question answering, 2023.

Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. Benchmarking large language models in retrieval-augmented generation, 2023a.

Jifan Chen, Grace Kim, Aniruddh Sriram, Greg Durrett, and Eunsol Choi. Complex claim verification with evidence retrieved in the wild, 2023b.

Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. Ragas: Automated evaluation of retrieval augmented generation, 2023.

Zhangyin Feng, Xiaocheng Feng, Dezhi Zhao, Maojin Yang, and Bing Qin. Retrieval-generation synergy augmented large language models, 2023.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1762–1777, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.99. URL https://aclanthology.org/2023.acl-long.99.

Yunfan Gao et al. Retrieval-augmented generation for large language models: A survey, 2024.

Or Honovich, Leshem Choshen, Roee Aharoni, Ella Neeman, Idan Szpektor, and Omri Abend. $q^2$: Evaluating factual consistency in knowledge-grounded dialogues via question generation and question answering. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7856–7870, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.619. URL https://aclanthology.org/2021.emnlp-main.619.

Yasuto Hoshi, Daisuke Miyashita, Youyang Ng, Kento Tatsuno, Yasuhiro Morioka, Osamu Torii, and Jun Deguchi. Ralle: A framework for developing and evaluating retrieval-augmented large language models, 2023.

Chenxu Hu, Jie Fu, Chenzhuang Du, Simian Luo, Junbo Zhao, and Hang Zhao. Chatdb: Augmenting llms with databases as their symbolic memory, 2023.

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

Jerry Liu. LlamaIndex, 11 2022. URL https://github.com/jerryjliu/llama_index.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting in retrieval-augmented large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5303–5315, Singapore, December 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.322. URL https://aclanthology.org/2023.emnlp-main.322.

Yubo Ma, Yixin Cao, YongChing Hong, and Aixin Sun. Large language model is not a good few-shot information extractor, but a good reranker for hard samples! *arXiv preprint arXiv:2303.08559*, 2023b.

Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. Generation-augmented retrieval for open-domain question answering. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4089–4100, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.316. URL https://aclanthology.org/2021.acl-long.316.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark, 2023.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2020.

Josh Reini et al. truera/trulens: Trulens eval v0.25.1, 2024. URL https://zenodo.org/doi/10.5281/zenodo.4495856.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023a.

Liang Wang, Nan Yang, and Furu Wei. Query2doc: Query expansion with large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9414–9423, Singapore, December 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.585. URL https://aclanthology.org/2023.emnlp-main.585.

Xiao Wang, Guangyao Chen, Guangwu Qian, Pengcheng Gao, Xiao-Yong Wei, Yaowei Wang, Yonghong Tian, and Wen Gao. Large-scale multi-modal pre-trained models: A comprehensive survey. *Machine Intelligence Research*, pages 1–36, 2023c.

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

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. C-pack: Packaged resources to advance general chinese embedding, 2023.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. Recomp: Improving retrieval-augmented lms with compression and selective augmentation, 2023.

# Chapter 8
# LLMs in Production

**Abstract** The promise of LLMs has largely been driven through research efforts, where analytic performance is often prioritized over other practical aspects of their usage. Translating this promise into real-world production-grade applications is rapidly becoming a new research frontier, driven not through academic endeavors but through commercial efforts by firms aiming to differentiate themselves in the marketplace, optimize their operations, or develop unique value from applying LLMs. This chapter aims to bridge the gap from promise to practice by walking the reader through the most important aspects of applying LLMs in practice. From decisions such as which LLM to use to how to optimize LLM latency, the relevant tools and techniques are highlighted to help guide readers in their journey into LLM application development.

## 8.1 Introduction

In this chapter, we aim to synthesize the various factors developers should consider when building LLM-enabled applications for production. The goal is to arm the reader with the latest set of best-practice guidelines and knowledge to aid in robust, cost-effective, and safe development. As we have discussed elsewhere, LLMs represent immense promise and risk at the same time, so it is important that developers be able to navigate the various steps of the development lifecycle to maximize the realization of that promise while minimizing the risk.

We begin in Sect. 8.2 by exploring common applications for LLMs, in order to give the reader a sense of the types of use cases that the later sections contextualize. We also review the different high-level categories of LLMs available, providing the reader with an additional dimension to assess LLM suitability across different use cases. While there are many lower-level aspects of LLMs and their abilities, such as context length, number of parameters, and architecture, these have been discussed at

U. Kamath et al., *Large Language Models: A Deep Dive*,
https://doi.org/10.1007/978-3-031-65647-7_8

length elsewhere (e.g., Chapter 2), so they are not discussed here.

In Sect. 8.3 and 8.4, we introduce common metrics used for evaluating LLM applications, and provide an extensive list of canonical datasets employed for these evaluations across a broad range of use cases.

Sect. 8.5 looks at LLM selection from the perspective of open-source vs. closed-source considerations. Various LLM aspects, such as analytic quality, costs, and data security and licensing, are explored to give the reader a sense of the various trade-offs one might have to make when designing their applications. We also discuss inference latency and LLM customization in this context to help the reader understand the various constraints that the selection of an open-source or closed-source LLM might introduce to their project.

In Sect. 8.6, the aim is to provide the reader with details on the various tools, frameworks, and patterns within the rapidly evolving LLM application development ecosystem. We will discuss various details, such as the available LLM application development frameworks, prompt engineering tooling, vector storage and LLM customization.

Next, we delve into more details around inference in Sect. 8.7. This section discusses important details on model hosting options, performance optimization innovations, and, perhaps most importantly, cost optimization. The inference cost in LLMs is still a core research focus, as Sect. 4.4 in Chapter 4 outlines, so insight into the current state of optimization here is important.

The chapter finishes with an overview of an *LLMOps* perspective on LLM application development. Given the complexity of LLMs and their fledgling adoption in applications, rigorous frameworks must underpin these projects. This ensures that as the potential for change in LLMs and how they can be interacted with and customized remains high, these innovations can be sustainably integrated experimentally, evaluated, and deployed with efficiency and minimal disruption to users. In this early phase of LLM adoption, maintaining user confidence and credibility is essential; an LLMOps perspective is intended to help in this process.

## 8.2 LLM Applications

Before getting into the technical details about developing production-grade LLM-enabled applications, it is useful to understand some of the problems and use cases that LLMs have been applied to. To do this, we will briefly introduce the various types of generic use cases/applications for which LLMs help to improve outcomes (e.g., conversational chatbots), and then provide an overview of the different categories of LLMs available for these use cases/applications.

This overview of LLM utility will help the reader situate the more technical sections of the chapter so that they are as practically informative as possible from a development life-cycle perspective.

### 8.2.1 Conversational AI, chatbots and AI assistants

This category of use cases is by far the most common to which LLMs have contributed significant improvements. In chatbots and conversational AI, LLMs and their enhanced language understanding over traditional language models contribute several important new benefits. Perhaps the most significant is their natural language understanding (NLU) abilities (Wei et al., 2022). Within the context of these types of applications, the LLM's ability comprehend user intent behind a query, and synthesize this input with existing parametric knowledge to create a coherent response, heavily influence the application's utility.

Similarly, since users of these applications often hold open-ended conversations that may span various knowledge domains or topics, the LLM's ability to track context is also critical to ensure coherent responses throughout the conversation session. In line with this, in the context of multi-turn dialogues, where the user and the application engage in back-and-forth conversation, LLMs leveraged the need to have the ability to selectively incorporate earlier queries or responses in the conversation to provide useful and coherent responses throughout the dialogue. Recent improvements in input context length have further advanced this specific ability in LLMs (Pawar et al., 2024) by effectively elongating the input range over which the LLM can reason.

Many of the use cases within this category of LLM application have a strong requirement for response factuality, meaning that the inherent tendency for LLMs to hallucinate is a significant challenge to be mitigated during development. The most popular way this risk is mitigated is through external knowledge bases from which relevant context can be extracted and used to condition the LLM response to verified knowledge. Integrating this knowledge base into the application architecture and the knowledge itself into the LLM input to elicit the appropriate response introduces another set of application development challenges to consider.

### 8.2.2 Content Creation

"Content is king", as the saying goes in the content marketing and digital media domains. Traditionally, the generation of content, in the form of stories, blog posts, newsletters, social media content, and many more, was performed by skilled humans versed in the art of identifying the types of content that would resonate with their audience, producing that content, and disseminating it through the most efficient channels. Today, however, LLMs have taken over much of the content production step within this domain. Applications exist that allow marketing professionals to curate demographic context, provide relevant content, and provide detailed guidance for LLM-enabled systems to generate highly engaging content and disseminate that content across channels according to a planned publication schedule.

Similarly, LLMs can be prompted to generate entire essays and stories about factual or fictional topics and events. This content is often indistinguishable from human-generated content by human readers, opening up new avenues for content creators, especially regarding the scale and diversity of content generated. However, these developments are not without their negative consequences, none more so than in educational settings, where students have quickly adopted LLMs such as OpenAI's ChatGPT to complete their assignments, leading to insufficient knowledge mastery (Lo, 2023). Nonetheless, LLMs have greatly improved the efficiency with which educators design, plan, and produce their curricula and serve as handy learning aids for students when leveraged productively.

### 8.2.3 Search, Information Retrieval, and Recommendation Systems

This category of LLM applications is fundamentally about providing relevant information to users as efficiently as possible. Typically, this information retrieval is performed in the context of the *needle in a haystack*, where not having mechanisms to identify and surface relevant content would mean that the user is perpetually inundated with off-target or irrelevant information. Consider a company with 10 years of product documentation mapping to multiple versions of products and their features and the complexity of navigating such a knowledge base. In the age of LLMs, users can now converse with AI assistants underpinned by innovative techniques for integrating user queries, knowledge bases, and metadata around the knowledge base to make understanding such product functionality more efficient than ever.

Again, owing to their improved NLU, entity extraction, summarization, and semantic embedding capabilities, the precision and recall with which relevant knowledge can be identified, reasoned over, and integrated into highly personalized responses is a revolution in these use cases. Much methodological, architectural, and tooling innovation around LLMs has been critical to these improvements, such as those reviewed in Chapter 7 in the context of retrieval-augmented generation and elsewhere.

### 8.2.4 Coding

Not surprisingly, LLMs are highly competent at generating computer programming language and natural language. The most popular solution in this space is Github Copilot, which was designed to assist human programmers in developing software using computer code. Since it is the most popular solution in this space, below we will look at its core capabilities as an exemplar of the types of benefits that these types of LLM-enabled applications provide.

- **Code auto-completion**, which can provide functionality as simple as traditional tab completion solutions for function/method and variable name completions and as complex as recommending entire code blocks based on real-time analysis of the existing code in a given script.
- **Multiple programming language support** allows developers to interoperate across coding languages efficiently. This capability is most useful in full-stack or specialist-domain application development, where multiple programming languages are used for different solution components. Imagine a full-stack developer writing data handling routines in JavaScript for the user interface. At the same time, Copilot suggests code blocks in Python for the back-end API that serves the data to the front-end. As of the time of writing, Github Copilot supports all programming languages available within public Github repositories. However, Copilot's competency in these languages is a function of that language's representation in Github public repository code. Interested readers are encouraged to explore GitHut[1] to understand better the relative proportions of different coding languages on Github.
- **Natural language understanding** enables users to specify the functionality or capabilities they would like computer code for through natural language prompting. This can often be achieved by simply writing comments describing what the subsequent code does. If Github Copilot is active for that script, it will recommend code to achieve the descriptions in the comments. These recommendations can provide surprisingly elegant code solutions to many problems and benefit from the wider context of the script being developed, especially if it is well commented/documented. Such functionality has clear benefits from an efficiency perspective. However, as always with LLM generation, users should validate and test recommended code carefully to safeguard against LLM fail-states.
- **Code refactoring and debugging** is another efficiency-improving capability of Github Copilot. Refactoring can be achieved thanks to the scale of code on which the system has been trained. Invariably, during this process, the LLM has learned many variants of code solutions for the same or similar problems, allowing it to provide alternative patterns for users to consider. Similarly, repetitive code, say for defining a class in Python, can be provided by Copilot as a boilerplate so that the developer can focus on only the functional components of the code, saving additional time. From a debugging perspective, Github Copilot can interpret execution errors or descriptions of unexpected outputs from code in the context of the code itself and the developer's description of what the code is intended to do to help identify candidate issues within the code. At a higher level, Copilot can also provide natural language explanations of execution flows and code functionality, further assisting the developers in exploring potential root causes for execution issues.

The aspects listed above contribute to increased efficiency in software development. Using a coding copilot can reduce the effort required to achieve effective and

[1] `https://madnight.github.io/githut/#/`

functional code, which traditionally might involve the use of reference textbooks, many visits to websites such as Stack Overflow or Github Gists, and code reviews by peers. Thanks to coding copilots, developers can achieve similar learning and feedback through a single intuitive interface. This is especially true thanks to some of the efforts to integrate coding copilots into popular Integrated Development Environments, such as Visual Basic Code, Vim, and JetBrains.

### 8.2.5 Categories of LLMs

There has been an explosion in the development of individual LLMs, especially within the open-source domain (Gao and Gao, 2023). With tweaks to the training data, the fine-tuning approach, the prompts, the scale of computing, learning objectives, and many other influential development aspects used, a proliferation of innovation has resulted in relentless improvements in many abilities. This explosion is impossible to survey within the constraints of a single book chapter. However, many online resources are available to readers to help them identify and understand the performance of specific LLMs on specific tasks and benchmarks, such as HuggingFace's `open_llm_leaderboard`, which consolidates LLM analytic performance in real-time as new models are developed and published[2]. These tools often allow users to filter comparisons based on their factors of interest, such as only comparing LLMs with a specific architecture type, or with a specific weight precision.

While comparative analysis of individual LLMs is essential for choosing the right model for your use case, understanding higher-level categories of LLMs is also important, as some categories are better aligned regarding outcomes to different use cases (e.g., chat vs. coding vs. retrieval). Below we highlight the most important LLM category types to enable a more informed selection process.

#### 8.2.5.1 General-Purpose LLMs

*General-purpose LLMs* are typically trained on large corpora of web-sourced content. In some cases that content is curated for quality, but by and large these training sets represent a comprehensive sampling of the various topics and domains represented on the web. As such, they are highly competent in generic language tasks (e.g. NLU, entity extraction, CoT, and question-answering), but may lack competencies in domain-specific tasks (e.g. financial numerical reasoning). This category of LLM is certainly the most versatile and can be a good starting point for many applications. Many also enable task specialization if they are lacking in specific areas; however, this is more true for open-source options than for closed-source models. In summary, these models should be considered for their text generation, NLU, QA,

[2] https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

translation, classification, summarization, and conversational abilities, where their out-of-the-box performance may be sufficient for most use cases.

#### 8.2.5.2 Multimodal LLMs

*Multimodal LLMs* have been trained on data from more than one "modality". Common modalities includes text, audio, video, and image data (Yin et al., 2024). Training models on these different modalities enables a new set of cross-modal use cases and is rapidly becoming the new frontier of generative AI (see Chapter 9 for an extensive overview). In line with some of the use-cases for LLMs discussed in Sect. 8.2, multimodal LLMs extend their use into applications such as image retrieval based on natural language descriptions or audio generation based on natural language instruction. Multimodal content generation, such as storytelling or product specifications, where text and image generation provide a richer and more expressive user experience, is rapidly becoming an area of interest for model developers. In general, these models are very large relative to traditional LLMs, and this scale introduces its own set of challenges for adoption and integration. However, their capabilities are truly impressive, and research is ongoing to improve their analytic and computational performance.

#### 8.2.5.3 Multilingual LLMs

*Multilingual LLMs* are trained on text data across more than one natural language. These types of models have received significant research attention and are useful for tasks that involve translation, multilingual reasoning, multilingual content generation, etc. Indeed, some multilingual LLMs support a large number of languages, such as the open-source model BLOOM developed by BigScience (Workshop et al., 2023), which is a 196B-parameter model trained on text across 46 natural languages and 13 programming languages. This model category's promise is clearly aligned with cross-lingual tasks, such as reasoning over text from multiple languages (e.g.Ranaldi et al. (2024)). In terms of applications, multilingual LLMs have been leveraged for customer service and other communication use cases where code-switching, the linguistic practice of alternating between natural language in communication, is commonly exhibited (Yong et al., 2023).

#### 8.2.5.4 Domain-Specific LLMs

In contrast with general-purpose LLMs, *domain-specific LLMs* have been trained on highly selective data from a narrow industry, field, or specialization. The general motivation when training this type of LLM is to adapt its abilities to the idiosyncrasies of the domain. This can be especially important when the domain has much jargon that does not translate in the more general context or when content within the domain is expected to be skewed relative to the general context. For example, in biomedical science, the domain-specific BioMistral LLM was developed (Labrak et al., 2024). This model was built by adaptively pre-training a Mistral model on PubMed Central, one of the largest repositories of biomedical research literature available on the web. By adapting the General Purpose Mistral 7B-parameter LLM, the domain-specific BioMistral models outperformed the general-purpose model in 9/10 biomedical tasks. Domain-specific LLMs also exist for education, legal, economic, political, scientific, and financial fields, among others. This can be a valuable starting point for many domain-specific LLM-enabled applications.

## 8.3 LLM Evaluation Metrics

Evaluating LLMs is a critical process involving systematic measurements to assess how effectively these models perform specific tasks. Evaluation metrics for LLMs can be classified across multiple dimensions based on their application and methodological approach. We describe here a number of these dimensions:

1. **With References vs. Without References**:
   - **With References**: Metrics that compare the model's output to predefined correct answers. Common in tasks such as translation and summarization.
   - **Without References**: Metrics that assess quality based on the model's internal consistency and linguistic properties.
2. **Character-based vs. Word-based vs. Embeddings-based**:
   - **Character-based**: Focus on character-level accuracy, which is useful in specific text generation tasks.
   - **Word-based**: Evaluate the presence, frequency, and order of words.
   - **Embeddings-based**: Leverage vector representations to assess semantic similarity beyond exact word matches.
3. **Task-Agnostic vs. Task-Specific**:
   - **Task-Agnostic**: Metrics that can be applied across different types of tasks without modifications.
   - **Task-Specific**: Metrics designed for specific applications that are directly related to the quality of task outputs.
4. **Human Evaluation vs. LLM Evaluation**:
   - **Human Evaluation**: Involves human judges assessing the quality or relevance of model outputs. Although inherently subjective, human evaluation is invaluable for gauging natural language fluency, ensuring coherence, and verifying relevance within the specific context of use. This form of assessment provides critical insights that automated metrics might overlook, par-

ticularly in terms of the text's contextual appropriateness and the subtleties of human language understanding.
- **LLM-based Evaluation**: Involves using a second LLM to evaluate LLM outputs, often through automated metrics or model-based judgments.

5. **Traditional vs. Non-Traditional**

   - **Traditional Metrics**: These metrics are concerned with the lexical and syntactic accuracy of the model's output. Common traditional metrics include exact string matching, string edit-distance, BLEU, and ROUGE, which prioritize the order and accuracy of words and phrases.
   - **Non-Traditional Metrics**: These metrics exploit the advanced capabilities of language models to assess the quality of generated text more holistically. Examples include embedding-based methods such as BERTScore, which utilizes embeddings to compare semantic similarity, and LLM-assisted methods such as G-Eval, where another powerful LLM is used to assess the quality of the generated text.

Next, we will explore some of these metrics that are commonly employed in the evaluation of LLMs, detailing their methodologies and specific applications.

### 8.3.1 Perplexity

*Perplexity* serves as a measure of how uniformly a model predicts the set of tokens in a corpus. A lower perplexity score indicates that a model can predict the sequence more accurately, exhibiting less surprise when encountering actual data. Conversely, a higher perplexity score implies that the sequence is unexpected from the perspective of next-token probabilities generated by the model.

Given a tokenized sequence $X = (x_0, x_1, \dots, x_N)$, where $N$ is the number of tokens, the perplexity of $X$ is calculated as follows:

$$\mathrm{PPL}(X) = \exp\left\{-\frac{1}{N}\sum_{i=0}^{N} \log p_\theta(x_i \mid x_{<i})\right\} \tag{8.1}$$

Here, $\log p_\theta(x_i \mid x_{<i})$ represents the log-likelihood of the $i$-th token, conditioned on all preceding tokens $x_{<i}$, as determined by the model. This value reflects the model's predictive accuracy per token within the sequence.

### 8.3.2 BLEU

One of the predominant metrics in this category is the *Bilingual Evaluation Understudy* (BLEU) score, which was introduced by Papineni et al. (2002), primarily for evaluating the quality of text translated from one natural language to another.

BLEU assesses the closeness of machine-generated text to one or more reference translations by examining the frequency and presence of consecutive words, known as n-grams, in both texts.

The mathematical representation of BLEU involves several steps:

- **Precision Calculation**: Precision is computed for n-grams of different lengths. For a given n-gram length $n$, precision $p_n$ is the ratio of the number of n-grams in the generated text that match the reference text to the total number of n-grams in the generated text. This count is clipped by the maximum number of times an n-gram appears in any reference text, which avoids over-counting.

$$p_n = \frac{\text{Number of clipped matching n-grams}}{\text{Total number of n-grams in generated text}} \tag{8.2}$$

- **Geometric Mean of Precision**: After calculating precision for various n-gram lengths, a final BLEU score, referred to as BLEU-N, is determined using the geometric mean of these precision values across all considered n-gram lengths.

$$\text{BLEU-N} = \left(\prod_{n=1}^{N} p_n\right)^{\frac{1}{N}} \tag{8.3}$$

- **Brevity Penalty**: To address the limitation of precision favoring shorter text (since shorter texts are likely to have a higher precision by virtue of fewer opportunities for error), BLEU incorporates a brevity penalty. This penalty is applied if the length of the generated text is shorter than the reference, thus discouraging overly concise translations.

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \tag{8.4}$$

- **Final BLEU Score**: The overall BLEU score combines the geometric mean of the precision scores with the brevity penalty to produce a final score between 0 and 1, where 1 indicates a perfect match with the reference texts.

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} \frac{1}{N} \log p_n\right) \tag{8.5}$$

Despite its widespread use, it is important to note BLEU's limitations. It does not account for the semantic accuracy or grammatical correctness of the generated text. Therefore, while BLEU is useful for a preliminary assessment of translation quality, it should be supplemented with other metrics or human evaluations to capture the nuances of language generation more comprehensively.

### 8.3.3 ROUGE

Another metric, called the *Recall-Oriented Understudy for Gisting Evaluation* (ROUGE), differs from BLEU in that it is recall-oriented. It primarily assesses how many words from the reference texts are also present in the machine-generated output, making it especially useful for evaluating automatic summarization tasks.

ROUGE includes several variants, each with a specific focus:

- **ROUGE-N**: ROUGE encompasses a collection of metrics designed to assess the effectiveness of summaries and translations by contrasting generated text against a set of human-crafted reference summaries.
  As it is focused on recall, ROUGE primarily evaluates the extent to which words and phrases from the reference summaries are reproduced in the generated text. This focus makes ROUGE especially valuable in scenarios where capturing as much of the reference content as possible is crucial.

$$\text{ROUGE-N} = \frac{\sum_{S \in \{\text{Reference Summaries}\}} \sum_{\text{gram}_n \in S} \text{Count}_{\text{match}}(\text{gram}_n)}{\sum_{S \in \{\text{Reference Summaries}\}} \sum_{\text{gram}_n \in S} \text{Count}(\text{gram}_n)} \tag{8.6}$$

  In this formula, $\text{gram}_n$ denotes n-grams of length $n$, and $\text{Count}_{\text{match}}(\text{gram}_n)$ is the maximum number of times that an n-gram occurs in both a candidate summary and the set of reference summaries.
  Examples include:
  - ROUGE-1 for unigrams.
  - ROUGE-2 for bigrams.
- **ROUGE-L**: Focuses on the longest common subsequence (LCS) between the generated and reference texts. Unlike n-gram overlap, LCS does not require the sequence to be contiguous, thereby capturing more flexible matches.
- **ROUGE-W**: An extension of ROUGE-L, this variant incorporates the length of the texts into its evaluation to counter the length bias.
- **ROUGE-S**: Measures the skip-bigram co-occurrence, which accounts for any pair of words in their sentence order, regardless of gaps. This metric emphasizes the order in which content is mentioned, regardless of intervening content:
- **ROUGE-SU**: Enhances ROUGE-S by including both skip-bigrams and unigrams in the evaluation:

### 8.3.4 BERTScore

Introduced by Zhang et al. (2020), *BERTScore* represents an advanced methodology for evaluating text quality by utilizing deep contextual embeddings from the BERT model. Unlike traditional metrics such as BLEU and ROUGE, which assess token or n-gram overlap, BERTScore calculates a similarity score for each token in the candidate text against each token in the reference text using these contextual embeddings.

BERTScore employs greedy matching to ensure that each token from the candidate text is aligned with the most similar token from the reference text, optimizing the overall similarity score. The evaluation includes three key metrics:

- **Recall** ($R_{\text{BERT}}$): This metric is calculated by taking the maximum similarity score for each token in the reference text, summing these scores, and then normalizing by the number of tokens in the reference. It reflects the extent to which the candidate text captures the content of the reference.

$$R_{\text{BERT}} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} \langle x_i, \hat{x}_j \rangle \tag{8.7}$$

- **Precision** ($P_{\text{BERT}}$): Similar to recall, precision sums the maximum similarity scores for each token in the candidate text and normalizes by the number of tokens in the candidate. It measures the extent to which tokens in the candidate text are represented in the reference.

$$P_{\text{BERT}} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} \langle \hat{x}_j, x_i \rangle \tag{8.8}$$

- **F1 score** ($F_{\text{BERT}}$): The harmonic mean of precision and recall, providing a balanced measure of both completeness and precision.

$$F_{\text{BERT}} = 2\frac{P_{\text{BERT}} \cdot R_{\text{BERT}}}{P_{\text{BERT}} + R_{\text{BERT}}} \tag{8.9}$$

BERTScore, offers semantic awareness and robustness to paraphrasing, making it highly effective for evaluating translations or summaries. However, it demands substantial computational resources and may not always correspond with human judgments, especially in evaluating the structure and coherence of text.

### 8.3.5 MoverScore

*MoverScore* evaluates the semantic similarity between a system's predicted text and a reference text using the concept of *Word Mover's Distance* (WMD) (Kusner et al., 2015). This metric helps capture semantic distances between words and phrases, making it particularly useful for text evaluation tasks. Unlike BERTScore, which utilizes one-to-one matching (or "hard alignment") of tokens, MoverScore incorporates many-to-one matching (or "soft alignment"), allowing for more flexible token alignments.

The key components of MoverScore include the following: