Progress® Agentic RAG

The RAG Cookbook:

Best Practices for Configuring our Retrieval-Augmented Generation models

WHITEPAPER

Overview: The RAG Cookbook

The rapid evolution of artificial intelligence (AI) has transformed the way we interact with information, creating new possibilities for more accurate, dynamic and context-aware systems. One of the most promising advancements in this field is Retrieval-Augmented Generation—often called RAG. It's a cutting-edge approach that combines the power of large language models (LLMs) with a data retrieval system. This hybrid model enables more informed, flexible and context-rich responses by integrating real-time access to controlled knowledge sources while generating human-like text.

RAG bridges the gap between traditional AI models, which rely solely on pre-existing knowledge stored within the model itself, and search-based systems that pull data directly from a database. By enhancing generation tasks with retrieval mechanisms, RAG enables more robust and contextually accurate outputs, addressing some of the core limitations of standalone generative models. This fusion opens the door to a wide range of applications, from improved conversational agents to advanced document summarization, personalized content creation, and even real-time decision support systems.

In this whitepaper, we will address the challenges and considerations in configuring RAG systems, providing insights into best practices.

At the end of this document, you'll find a list of useful options to help you configure your own RAG pipeline.

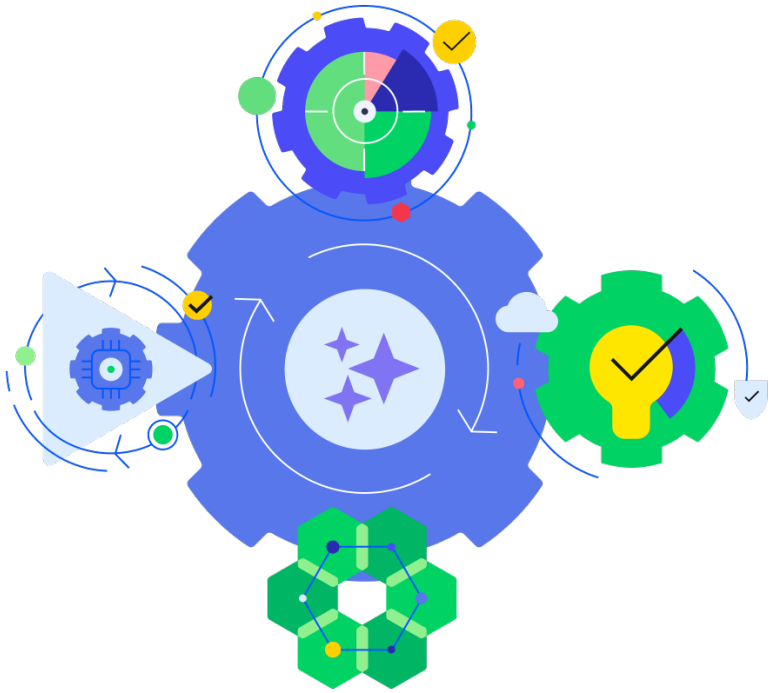
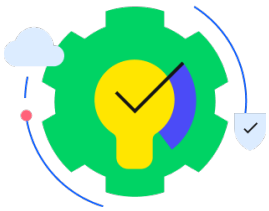


Table of Contents



Basic Principles of RAG

RAG is a two-step process that combines a retrieval system with a generative model.

The first step involves retrieving relevant information from a knowledge source based on the user's query. This retrieval process is typically performed using a semantic search algorithm that matches the query against the available data and returns the most relevant text blocks. It can also use results from a keyword search.

The second step is the generation phase, where a large language model processes the retrieved information along with the user's query to generate a coherent and contextually relevant response. The LLM leverages the retrieved context to provide more informed and accurate answers. The user query and the retrieved context are concatenated in a prompt that will explicitly mention that the LLM is expected to use only the retrieved context to generate the answer. That way, the LLM does not rely on its pre-existing knowledge.

These two steps can be tuned regarding different parameters:

- **Amount of context retrieved:** The more text blocks retrieved, the more information the LLM can draw on to generate the answer. However, too much context can lead to irrelevant or noisy information. You need to find the right balance, and it usually depends on the type of information you are dealing with. In some cases, questions have a very specific answer that can be found in a single text block. In others, the answer requires a broader context.

- **Ranking of the retrieved text blocks:** You need to rank the text blocks to make sure the most relevant ones are used by the LLM. Semantic results and keyword results come with different scores which cannot be compared directly. A good practice is to use a re-ranking that evaluates the relevance of the text blocks according to the user query.

- **Type of LLM used:** The more powerful the LLM, the more accurate the answers, but that often makes the computation more expensive. You need to do a cost/benefit analysis between the quality of the answers and the cost of the computation, then find the right balance.

- **Prompt quality:** The prompt is the text the LLM will use to generate the answer and should be carefully crafted to guide the LLM in the right direction. Clear and concise prompts also explicitly mention that the LLM should use only the retrieved context to generate the answer.

Evaluating the Quality of the RAG Pipeline

Once you have played with these parameters and configured your RAG pipeline, you need to evaluate its quality. It is not a one-shot process, but an ongoing task that requires regular monitoring and fine-tuning. Indeed, the content of the knowledge box can change as you ingest new resources, meaning questions originally answered correctly can become incorrect. Also, the user questions evolve, it's not uncommon for them to expand to topics not insufficiently covered by your resources.

As it is a complex and critical task, Progress offers RAG Evaluation Metrics, or REMi, an efficient open-source fine-tuned LLM that simplifies the assessment of RAG pipelines.

REMi considers the main inputs/outputs in the RAG pipeline, which are:

- Query: The user's question, which the model will try to answer
- Context: The information retrieved by the retrieval step, which aims to be relevant to the user's query.
- Answer: The response generated by the language model after receiving the query and context pieces.

REMi defines the following metrics to assess the quality of the RAG pipeline:

- Answer relevance: How relevant is the generated answer to the user query?
- Context relevance: Is the retrieved context relevant to the user query?
- Groundedness: How well is the generated answer grounded in the retrieved context?

By combining these metrics, REMi provides a comprehensive view of the quality of the RAG pipeline.

Potential outcomes from pipeline quality assessments:

- Scenario 1: **Context relevance is high but answer relevance and groundedness are low.** This means the model is generating evasive answers. The semantic search successfully retrieves relevant context pieces, but the model fails to generate a relevant and grounded answer.

  Suggested Fix: Use a different LLM.

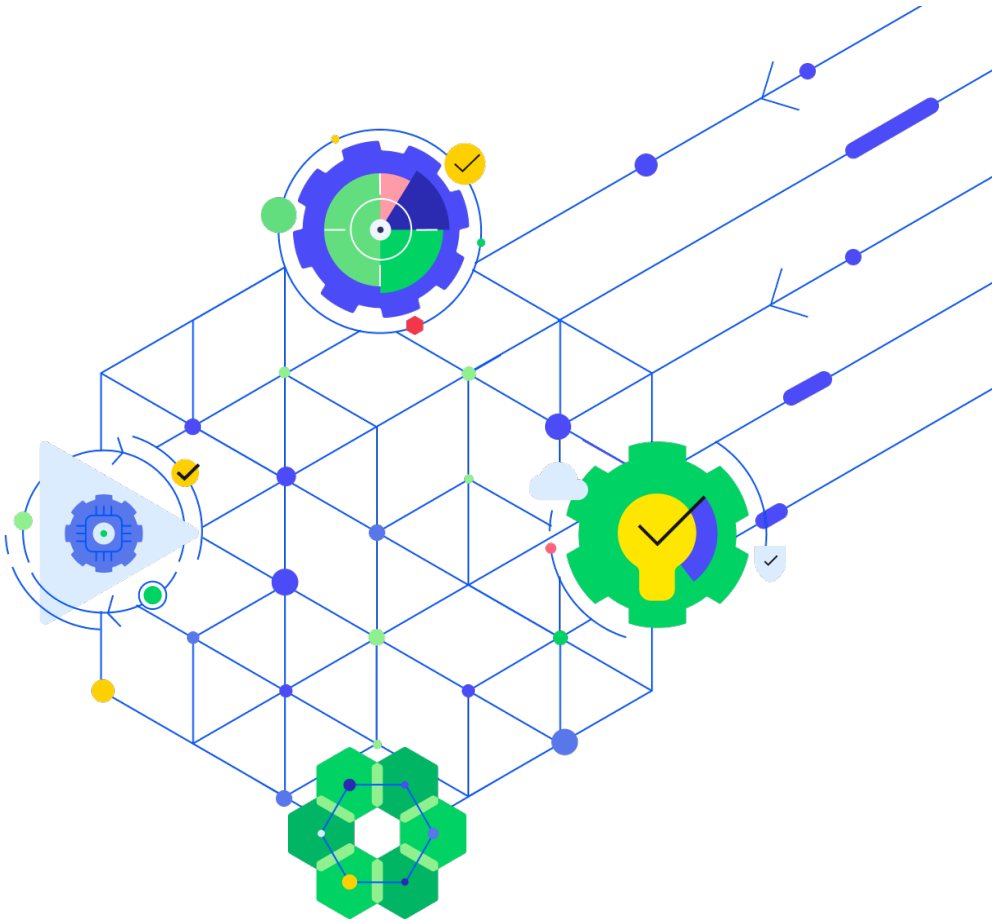
- Scenario 2: **Answer relevance is high, but the context relevance and groundedness are low.** This indicates that the model is generating unverifiable answers. The LLM generates a relevant answer, but it's not based on the information stored in your knowledge box.

  Suggested Fix: Check whether information is missing from your knowledge box. If the information is present, change your search and RAG strategy parameters.

- Scenario 3: **Groundedness is high,** but the **answer relevance and context relevance are low.** When this happens, the model is generating **unrelated answers.** The LLM generates an answer based on the context, but the answer is not relevant to the user query. This can happen if the wrong context pieces are retrieved, but the LLM still generates an answer based on the available information, disregarding the nuances of the query.

  Suggested Fix: Review your retrieval strategy. Pay close attention to both semantic thresholds as well as prompts. Be sure that they explicitly call out things such as "only use the provided context" and "do not use any external knowledge."

This paper will focus here on cases where the information is present in the knowledge box, but the retrieval step is not providing the relevant context to the LLM.



Enriching the Context

RAG is about retrieving the best text blocks to answer the question. The way it finds the "best" text blocks is by using semantic search, sometimes combined with keyword search. It assumes that the best text blocks are the most semantically similar to the user query and will thus produce the most relevant answer.

Some critical information might be located around the semantic matches without being retrieved. That's why it is useful to enrich the context by adding extra content to the initial retrieval.

Neighboring Text Blocks or Full Documents

When information is scattered across multiple text blocks, results can be improved by retrieving neighboring text blocks to the ones that are semantically similar to the user query.

For example, in a user support context, imagine a long introduction presenting a typical problem, followed by a list of solutions. It is very likely that the user will describe their problem (as they probably do not know about the solution), so the introduction will have a very good semantic score, while the text blocks describing the solution might not.

In this case, if you know your documents are structured this way, you could consider systematically adding the two or three text blocks following the ones that are semantically similar to the user query. This will help provide the LLM with the information needed to generate the answer.

Adding too many text blocks can amplify noise and lead to irrelevant information, so it's important to find the right balance.

Note that if your documents are small, you could consider adding the full document to provide sufficient query context.

Metadata

Another way to enrich the context is to append metadata. The most common metadata is the title of the document. Let's imagine your documents are articles about famous scientists for the following case:

- Question: "Who discovered that the universe is mostly made of hydrogen?"
- Context: The document containing the answer is titled "Cecilia Payne-Gaposchkin"
- Reference: One of its text blocks contains the following sentence: "She is the genius who discovered that the Universe is mostly made of hydrogen".

This text block will be retrieved, as it is semantically very close to the question. Yet, it does not contain the answer to the question. The title does, but it will not be retrieved because it does not mention the universe nor hydrogen.

In this case, you can add the title of the corresponding document of each retrieved text block to the context, and the LLM will be able to provide a good answer.

Note, the title is not the only metadata you can use. Consider including the author, publication date, labels, tags, etc.

Data Augmentation

Data augmentation is a technique used to automatically generate extra content from the existing ones. A very common type is summarization. You can use a summarization model to generate a summary of each document at ingestion time, then add it to the context whenever a text block from this document is retrieved.

For example, a user asks, "What is the best hammer for laying fabric on furniture?" The corresponding document is "Upholstery hammer", but the retrieved text does not describe the hammer. It simply mentions it's effective at laying fabric on furniture. By adding the title as described in the previous section, the LLM would answer "Upholstery hammer", but it would not be very informative. If you add the summary, the LLM would be able to generate a more detailed answer.

Data augmentation techniques can create content with insights about a given topic, such as a security warning highlighting the risks of a given practice. It can also produce JSON structured data, like a list of brands or prices, etc., that can also be valuable to the LLM.

Advanced Strategies for RAG

Here is a list of interesting examples of advanced strategies we have developed, specific to certain use cases. Use them to inspire your own RAG design strategy.

Filtering on Titles

Documents come in all kinds. Some are very general and discuss many different topics, while others are specific, concentrating on a single subject.

If you want the LLM to use a specific document, and know the title contains keyword-rich hints about its relevance, include the title in your filter for the retrieved text blocks.

Usually in that case, it is more efficient to perform a keyword search on titles only, then retrieve the text blocks from the documents that were found using a combination of semantic and keyword search.

Rephrasing

Rephrasing the user query is a common technique to improve the quality of semantic search results. It is usually not recommended for keyword search as it can produce unexpected results.

If the user query is not well formulated, the semantic search might not find the best text blocks during retrieval. Regular rephrasing can help determine the intent, optimize the query and improve the likelihood that the best text blocks are retrieved from your source files.

However, you can also use a specific prompt to make rephrasing more specific.

A great example relates to multilingual context, where documents are all in English, but you want to enable users to ask questions in their own language. For this, using a multilingual embedding model to index your content should work well. However, there are cases where an English embedding model might prove more efficient. In that case, use a prompt that rephrases the user query in English, so English embeddings can be retrieved.

Automatic Filtering

Semantic search can make strong matches on content that may not be explicitly

relevant to the user query, like “What was the biggest success of The Beatles in 2003?” You may have a content suggesting “Yellow Submarine” was a huge success, while failing to mention it happened in 1968. On the surface, it did match most of the query parameters.

To avoid situations like this, you need to evaluate your query phrasing and extract anything that could be used as a filter. You can then repurpose it to narrow your semantic search results to content matching the metadata filters.

Boosting Specific Sources

When available sources are not equally reliable, it’s best to point the LLM to the most reliable ones to generate the answer.

Unfortunately, semantic search might find the best text blocks in less reliable sources because they are the most semantically similar to the user query. Valuable information is then drowned in noise or even not surfaced at all.

To ensure text blocks are retrieved from the most reliable sources, try duplicating your query by running it on:

- Reliable sources only, using the appropriate filtering
- The rest of sources, with a reverse filter
- Merge the two results sets by applying a higher weight to the valuable sources

This gives reliable sources a better ranking, so the LLM uses them to generate the answer

Besides filtering to target a specific kind of source, you can also consider a specific rephrasing prompt to fit the language of the sources when running the two queries. For example, if the reliable sources are legal documents, you can use legal vocabulary in the prompt, while preserving general vocabulary for the rest..

Multi-Step Queries

Sometimes you want an answer from a specific part of your information corpus, if this answer exists there. If not, you want to draw from the rest of available data and files.

For example, if your corpus contains contracts, amendments and meeting minutes, you want to prioritize the amendments, then the contracts and finally the meeting minutes when answering the question.

A multi-step query will accomplish this:

- Run the query on the amendments only
- If there is no valuable result, run it on the contracts only
- If contracts returns no valuable results, run it on the meeting minutes

Based on the collected results, you can let the LLM generate the answer. It might also be combined with automatic filtering, so when the user asks, "What do the amendments say about the new policy?", you would run the first-step query only.

Injecting Caveats

Language models can't always distinguish domain-specific terms that might be lexically similar but have different meanings. That's where fine-tuned LLMs can be very useful. But what if you must use a generic LLM? Let's look at an example of how to handle this situation by using caveats.

In a financial context, the LLM might be confused between "qualitative easing" and "quantitative easing." To help the LLM, you can inject a caveat in the prompt, providing the definition of both terms and the important differences between them, and direct the LLM to mind the difference when generating the answer.

But there will likely be many more terms that might cause confusion in a financial context beyond the two listed above. But listing each individually in the prompt is likely a heavy lift.

Instead, create a "caveats" resource that contains all the caveats in one place. Then when a user asks a question, you can run a normal query plus a keyword search for the caveats resource. Once located, the retrieved text blocks can be added to the context.

Note: A keyword search is recommended here because we can assume the user question is not about the difference between "qualitative easing" and "quantitative easing," so the corresponding caveat text block is not likely to be found in the semantic search results.

Useful Options for Progress Agentic RAG

The strategies described in this whitepaper can be implemented in the Progress Agentic RAG platform using the following options.

Find more details about these options in the product documentation.

Search Options

- **features** (**semantic** and/or **keyword**): To choose the type of search to perform
- **filters:** To narrow the search to specific sources according to their metadata
- **keyword_filters:** To narrow the search to sources containing a given keyword
- **autofilter:** To automatically filter search results based on the user query
- **top_k:** To limit the number of text blocks retrieved
- **reranker:** To re-rank the retrieved text blocks according to the user query
- **rephrase:** To rephrase the user query for better semantic search results
- **prompt.rephrase:** To provide a custom rephrasing prompt that will be used to rephrase the user query

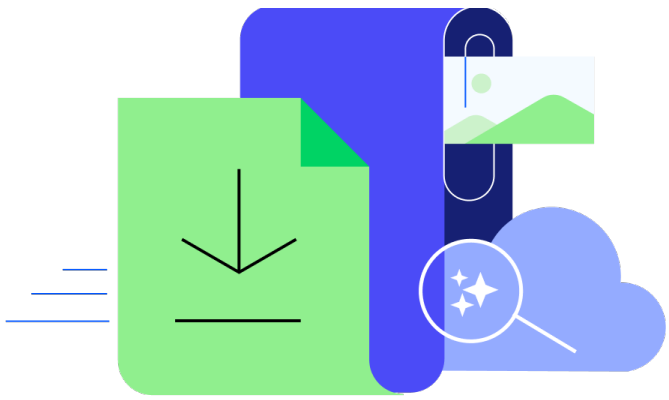
RAG & Generation Options

- **generative_model:** To choose the LLM
- **prompt.user:** To provide a custom user prompt that will be used by the LLM to generate the answer
- **prompt.system:** To provide a custom system prompt that will be used by the LLM to generate the answer
- **rag_strategies:**
    - **full_resource:** To retrieve the full document when a text block is retrieved
    - **field_extension:** to retrieve some additional fields from the document when a text block is retrieved
    - **hierarchy:** To retrieve the title and the summary when a text block is retrieved
    - To retrieve the neighboring text blocks when a text block is retrieved
    - **metadata_extension:** To retrieve some metadata when a text block is retrieved
    - **prequeries:** To run extra queries before the main one to either filter the sources or add more context from different parts of the corpus

- **rag_images_strategies:** To pass either images contained in the retrieved text blocks to the LLM, or a full screenshot of the page containing the text block

Results Options

- **answer_json_schema:** To produce a JSON structured answer, computing specific attributes based on the context
- **citations:** To provide the sources of the retrieved text blocks
- **citation_threshold:** To increase or decrease the accuracy of the citations



What to know more? **Try it for Free!**

About Progress Software

Progress Software (Nasdaq: PRGS) empowers organizations to achieve transformational success in the face of disruptive change. Our software enables our customers to develop, deploy and manage responsible AI-powered applications and digital experiences with agility and ease. Customers get a trusted provider in Progress, with the products, expertise and vision they need to succeed. Over 4 million developers and technologists at hundreds of thousands of enterprises depend on Progress. Learn more at www.progress.com

Worldwide Headquarters

Progress Software Corporation
15 Wayside Rd, Suite 400, Burlington, MA 01803, USA
Tel: +1-800-477-6473

facebook.com/progresssw
twitter.com/progresssw
youtube.com/progresssw
linkedin.com/company/progress-software
progress_sw_

Progress