

Briefing Doc: Improving RAG Systems with Advanced Retrieval and Evaluation Techniques

Executive Summary

This document synthesizes key insights on optimizing Retrieval-Augmented Generation (RAG) systems, based on an expert discussion with Andrew Drov, a research scientist at Databricks. The central thesis is that the quality of the retrieval stage is the most significant bottleneck in RAG applications, and improvements here yield the most substantial gains in overall system performance.

The primary takeaways are:

- **Retrieval is Paramount:** The success of a RAG system is overwhelmingly dependent on its ability to find the correct piece of information. As Drov states, "if you find like the right piece of information that's like 99%... you're basically there."
- **Fine-Tuning Embeddings is Critical:** Off-the-shelf embeddings capture general similarity but often fail on domain-specific relevance. Fine-tuning an embedding model on custom query-document pairs is a high-impact, surprisingly straightforward technique to drastically improve retrieval accuracy.
- **Rerankers Offer Significant Uplift:** Implementing a two-stage retrieval process, where a fast initial search is followed by a more powerful reranking model, can boost recall and other key metrics by 10-20%. However, research indicates that rerankers can underperform if given too large a set of documents to rank, highlighting a nuance in their application.
- **LLMs are Multi-Purpose Tools:** Large Language Models (LLMs) can be leveraged across the RAG pipeline—not just for generation. They can be used to rewrite user queries for better retrieval, generate synthetic data for evaluation and fine-tuning, and act as powerful rerankers themselves.
- **Meaningful Evaluation is Essential but Difficult:** Generic benchmarks are insufficient for enterprise applications. Creating meaningful, custom evaluation sets is crucial. LLMs can assist in this by automatically generating relevant

questions from a corpus of documents, a technique known as "question generation."

1. Introduction to Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is a technique designed to enhance the capabilities of Large Language Models (LLMs). LLMs are trained on vast but static datasets, meaning they lack knowledge of recent events or private, proprietary information. RAG addresses this by "injecting useful information into the prompts" so that LLMs can provide accurate, up-to-date, and contextually relevant answers.

The basic RAG pipeline consists of the following steps:

1. **Ingestion:** A collection of documents (e.g., internal documentation, knowledge bases) is processed and indexed. This typically involves chunking the documents and creating vector embeddings for each chunk.
2. **Retrieval:** When a user asks a question, the question (query) is also embedded. The system then performs a similarity search to find the most relevant document chunks from the indexed collection.
3. **Augmentation:** The text from the retrieved document chunks is combined with the original user query into an expanded prompt.
4. **Generation:** This augmented prompt is fed to an LLM, which generates an answer grounded in the provided context.

2. The Centrality of Retrieval: The Primary Bottleneck

The most critical factor determining the quality of a RAG application is the retrieval step. The entire system's performance hinges on its ability to find the correct context.

- **The "99%" Rule:** Drov emphasizes that for many questions, the answer exists directly within a single document. The core challenge is not complex reasoning across multiple sources, but simply locating that one crucial piece of information.
- **Garbage In, Garbage Out:** An eloquent and powerful generator model is rendered useless if it receives irrelevant or incorrect context. As noted in the discussion, "it doesn't matter how amazing your... generative model is if you can't retrieve the right context."

3. Techniques for Improving Retrieval Quality

3.1. Query Enhancement

A common failure point is the initial user query. Users often interact with RAG systems in a conversational manner, which is suboptimal for information retrieval.

- **Problem:** Conversational queries ("Hi, how are you doing? I have this question today...") contain noise that can confuse retrieval systems.
- **Solution:** Use an LLM as a preliminary step to rewrite the user's conversational input into a concise, keyword-focused query that is better suited for search.

3.2. Fine-Tuning Embedding Models

Standard retrieval methods, including traditional keyword search (like BM25) and off-the-shelf neural embeddings, have inherent limitations. Fine-tuning the embedding model is presented as a high-impact solution.

Method	Strengths	Weaknesses
BM25	Excellent with specific keywords, product codes, and identifiers. Simple, effective, and a very strong baseline.	Fails on synonyms and misspellings. Relies on exact word overlap.
Unsupervised Embeddings	Robust to synonyms and paraphrasing; captures semantic similarity.	Captures a generic definition of similarity which may not align with domain-specific relevance. Can miss key identifiers that BM25 would catch.

The Case for Fine-Tuning:

- **Purpose:** To teach the embedding model what "relevance" means for a specific domain. By training on query-document pairs, the model learns to connect queries to the documents that actually answer them, even if they don't share keywords or look textually similar.

- **Process:** Fine-tuning uses supervised learning (often contrastive learning) on a dataset of known good query-document matches. This data can be generated automatically using an LLM via question generation.
- **Recommendation:** Drov states that embedding fine-tuning should be "very high on the list" of techniques to try for improving a RAG application. It is a surprisingly straightforward process with readily available libraries and can be done effectively even with a few thousand model-generated examples.

3.3. Implementing Rerankers

Reranking is a two-stage retrieval architecture that can significantly boost performance.

- **Stage 1 (Retrieval):** A fast, efficient model (e.g., BM25 or a fine-tuned dense embedding) retrieves a large pool of candidate documents (e.g., 100 candidates).
- **Stage 2 (Reranking):** A more expensive and powerful model examines this smaller candidate pool and re-sorts it to produce a more accurate final ranking.

Key Characteristics of Rerankers:

- **Improved Accuracy:** Rerankers jointly encode the query and each document, allowing for a deeper, more contextual understanding of relevance than is possible with faster, single-pass embedding models. This can lead to a **10-20% improvement** on metrics like recall and NDCG.
- **Cost-Quality Tradeoff:** While adding a component, a reranker can lead to net savings. By providing a more accurate, shorter list of documents to the final generator LLM, it reduces the length of the final prompt, which in turn lowers cost and latency at the generation stage.
- **LLMs as Rerankers:** LLMs themselves can be used as effective rerankers. A simple method is to prompt the LLM to output a relevance score for a given query-document pair. A more advanced technique involves using the model's logits and few-shot prompting with difficult examples to improve performance.

An Important Nuance: Recent research from Drov's team reveals a surprising limitation. Rerankers perform well on small candidate sets (e.g., 10-50 documents) but their performance can degrade and even become worse than the initial embedding model when asked to rank very large sets (e.g., 1,000+ documents). This finding suggests that the number of documents being reranked is a critical hyperparameter.

4. The Critical Role of Meaningful Evaluation

Generic academic benchmarks (e.g., HELM, MMLU) are not representative of the challenges in specific enterprise use cases. Creating custom, meaningful evaluations is essential for progress.

- **The Challenge:** Building a high-quality evaluation set is a significant amount of work, which discourages many teams. Drov notes, "it's just like so hard to convince people to make their own eval."
- **LLM-Assisted Evaluation:** LLMs can automate and scale the creation of evaluation data. The primary technique is **question generation**:
 - Take a document from the corpus.
 - Prompt an LLM to generate several questions that can be answered by that document.
 - (Optional) To improve quality, repeat the generation process multiple times and select the most frequently generated questions (self-consistency).
 - This creates a ground-truth pair: a question linked to a specific document that should be retrieved.
- **Pitfalls:** This process can create noise. For example, an LLM might generate an "underspecified" question like "Who did it?" for a mystery story, which is too generic to be a useful retrieval query.

5. Advanced Prompting and Application

Few-Shot Prompting

Including examples (shots) in a prompt can "teach" a model how to perform a task without updating its parameters.

- **Starting Point:** Simply including a few randomly sampled examples in the prompt is a great starting point for tasks like question generation.
- **Improving Diversity:** To get more diverse and robust outputs, run the process multiple times with different sets of randomly sampled examples.
- **Advanced Technique:** For even better performance, focus on finding "hard demonstrations"—examples that the model would typically get wrong—to include in the prompt, as this provides a more useful signal.

Practical Case Study

Brooke Wenning shared a customer example that successfully applied these principles:

- **Problem:** A RAG application using a large Llama 3.1 70B model for generation was too slow because it was being passed too much context.
- **Solution:** A smaller, faster 8B parameter model was implemented as a reranker/filter. Its sole job was to assess the relevance of the retrieved context before it was passed to the generator.
- **Result:** The overall system became **cheaper, faster, and higher quality** by intelligently reducing the amount of context fed to the expensive final generation step.