

Open in app ↗

Medium

🔍 Search



Level Up Coding · [Follow publication](#)

★ Member-only story

# Rankify: A Comprehensive Approach to Retrieval, Re-Ranking, and Beyond

An In-Depth Exploration of a Cutting-Edge Python Toolkit



AI TutorMaster · [Follow](#)

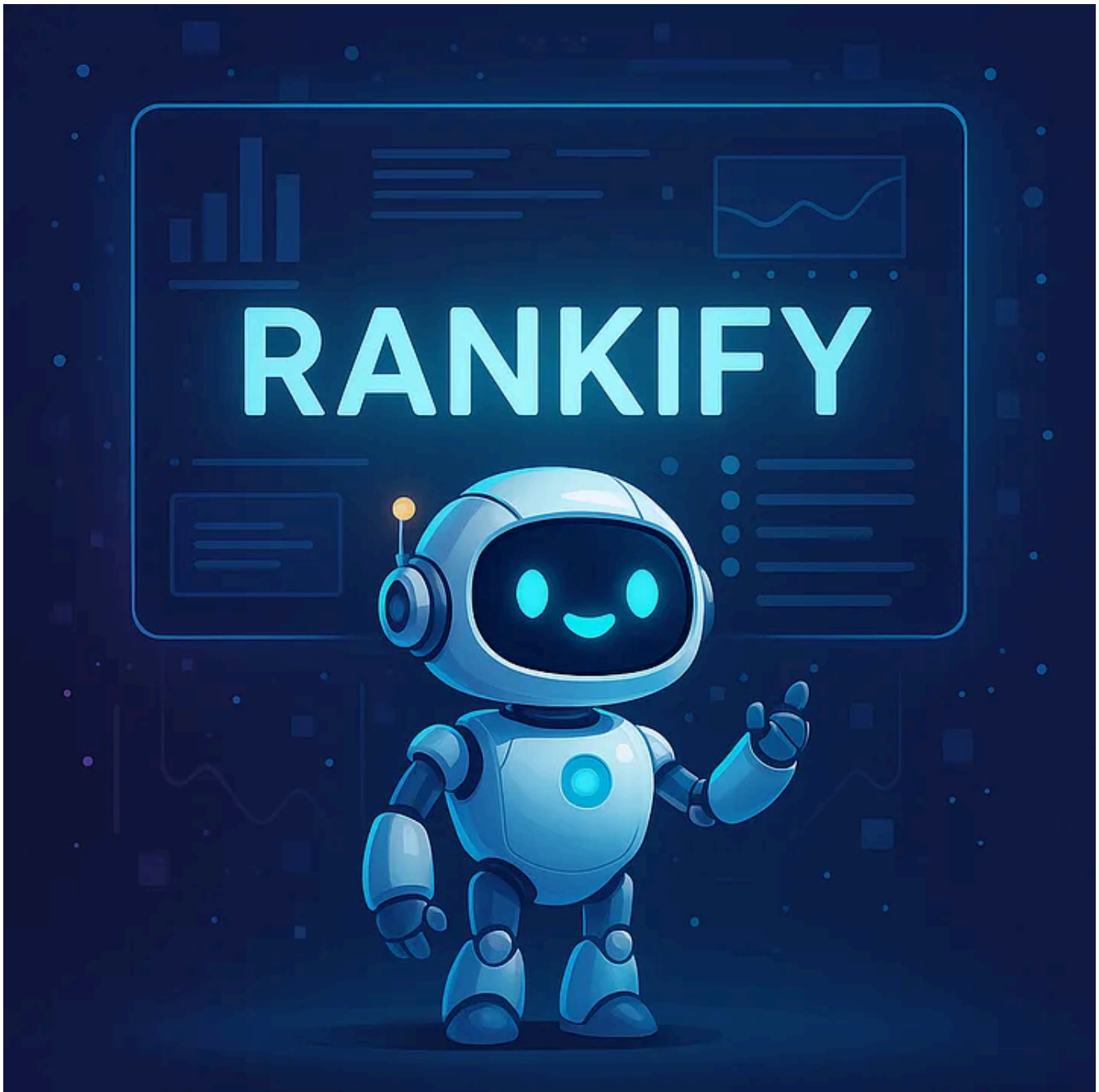
Published in Level Up Coding

9 min read · 18 hours ago

▶ Listen

📄 Share

⋮ More



Created by Dall-E3

## 1. Introduction

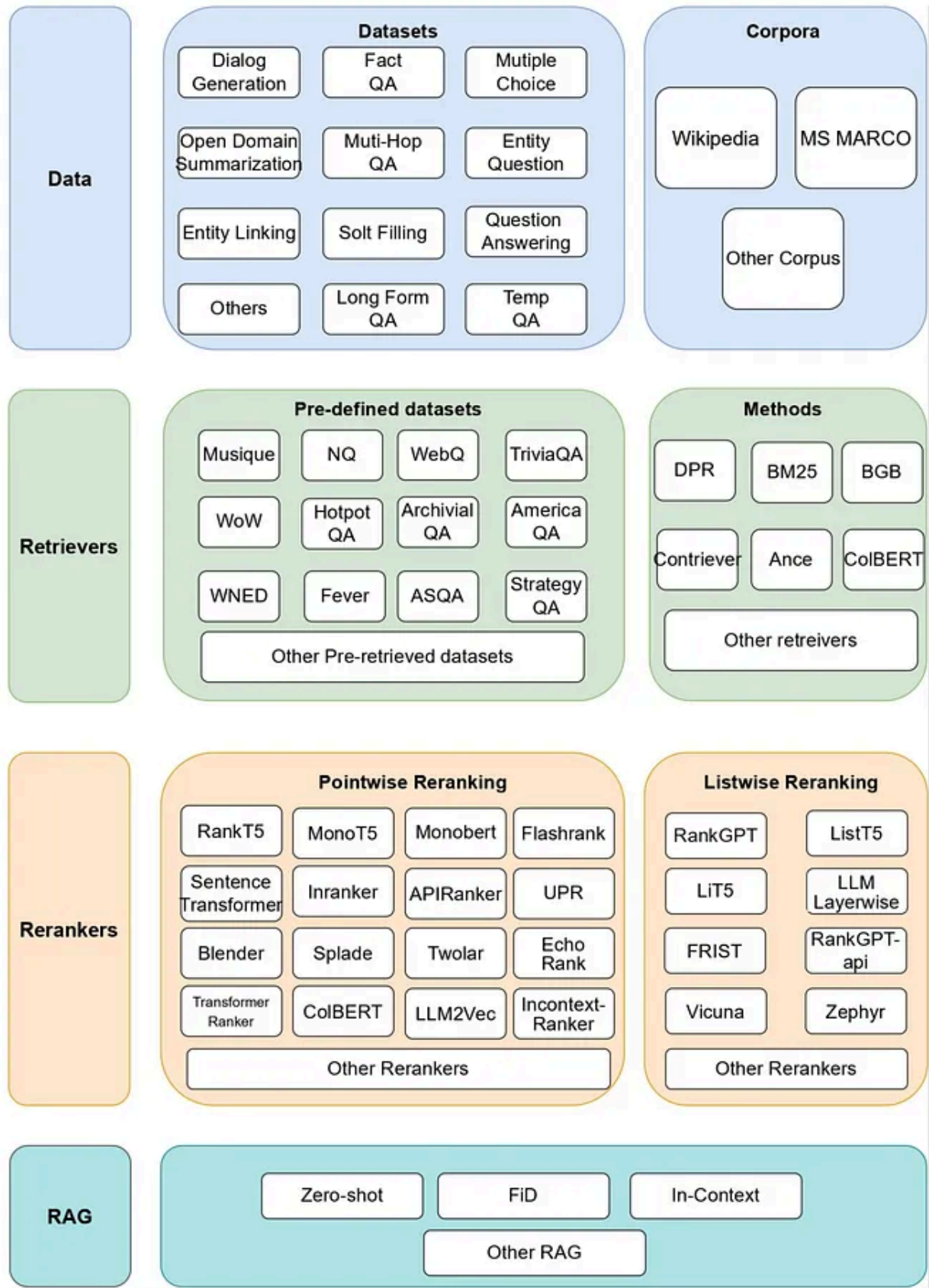
“Somewhere, something incredible is waiting to be known.”-Carl Sagan

**Rankify**, a Python toolkit that has been making waves in both academia and industry for its comprehensive, modular, and user-friendly approach to **retrieval**, **re-ranking**, and **retrieval-augmented generation (RAG)**.

We have witnessed an evolutionary shift from purely **lexical-based search** strategies to **dense embedding-based** approaches. Additionally, the surge in **re-ranking**

models has helped refine and reorder initial search results to bubble up the most relevant documents. Further, **retrieval-augmented generation** marries these techniques with text generation, enabling language models to answer questions with higher factual accuracy by consulting relevant documents on the fly.

Yet, as these fields have progressed, practitioners frequently found themselves piecing together multiple tools, each handling a specialized task. Rankify addresses this fragmentation with a **unified, modular, and robust** solution that streamlines the entire pipeline: from retrieving initial results and re-ranking them, to generating final, contextually-enriched answers.



Source-[here](#)

## 2. Understanding Rankify

### 2.1 The Vision Behind Rankify

Rankify was conceived out of a necessity to unify different processes under one roof. Traditional IR toolkits often cater to either retrieval or re-ranking exclusively. Some frameworks handle retrieval-augmented generation but neglect to offer deeper granularity in ranking stages. Rankify bridges these gaps:

- **Retrieval:** Leverages multiple methods — from classic sparse retrieval like BM25 to advanced dense models like DPR, ANCE, BGE, Contriever, and ColBERT.
- **Re-Ranking:** Brings in a wide array of re-rankers, from MonoBERT to RankT5 and beyond, allowing flexible second-stage ranking with pointwise, pairwise, or listwise algorithms.
- **RAG:** Facilitates generating answers that are grounded in retrieved documents, thereby enhancing factual reliability.

## 2.2 Key Advantages Over Traditional Methods

- **Modularity:** With a single interface, users can mix and match retrieval methods, re-ranking models, and RAG approaches.
- **Pre-Retrieved Data:** Rankify provides pre-retrieved documents and ready-to-use indexes for large corpora like Wikipedia and MS MARCO.
- **Scalability:** Built around Python and PyTorch, Rankify efficiently scales to handle large datasets.
- **Comparative Experiments:** Because it consolidates diverse approaches, researchers can systematically compare performance across different retrieval or re-ranking strategies.

## 3. Differences from the Chain-of-Thought Paradigm

### 3.1 What Is Chain of Thought?

In large language model research, the term **chain of thought** often refers to a reasoning framework where the model (or method) openly deliberates intermediate steps before reaching a final conclusion. This approach attempts to mimic human-like reasoning sequences, ensuring each step is transparent.

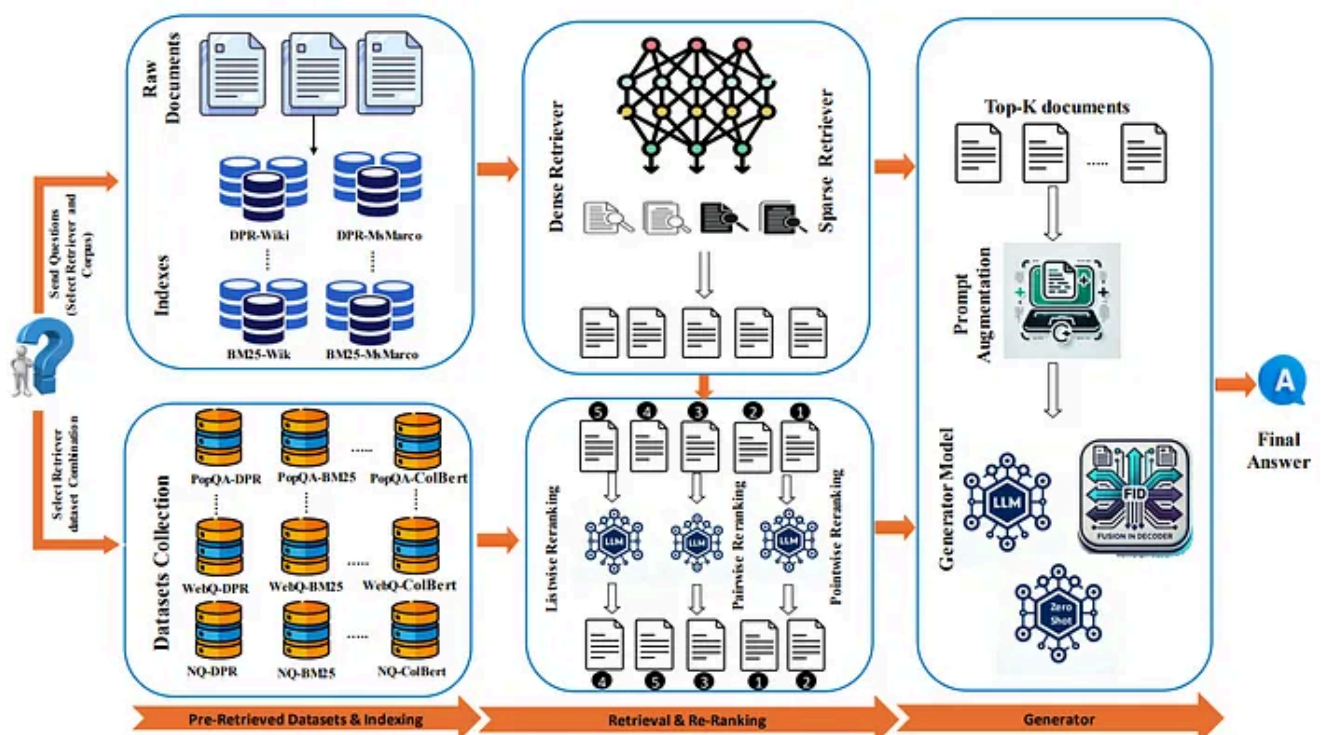
### 3.2 How Rankify Surpasses Chain-of-Thought Approaches

While chain-of-thought reasoning is insightful, it focuses primarily on how a model's internal logic unfolds rather than ensuring the **external, relevant context** is properly retrieved, weighted, and utilized. Rankify, on the other hand, is:

1. **Context-Focused:** Instead of counting on an internal chain-of-thought, Rankify actively retrieves relevant documents and systematically re-ranks them, ensuring that the language model's generation is rooted in accurate external data.
2. **Robust and Modular:** Chain-of-thought can be difficult to evaluate or scale because it's deeply integrated into the model's hidden layers. Rankify uses well-established external retrievers and re-rankers, making it easier to customize and extend.
3. **Better for Knowledge-Intensive Tasks:** When factual correctness is paramount, relying solely on a model's internal "thought chain" risks hallucination. Rankify's pipeline ensures relevant sources are always at the forefront.

"Information is the oil of the 21st century, and analytics is the combustion engine." — Peter Sondergaard

## 4. Core Components of Rankify



Source-[here](#)

### 4.1 Retrieval

Rankify supports both **sparse** (BM25) and **dense** (DPR, ANCE, BGE, Contriever, ColBERT) retrievers.

- **Sparse Retrieval (BM25):** Reliable for shorter queries and situations where exact keyword matches suffice.
- **Dense Retrieval (e.g., DPR):** Uses neural embeddings to capture semantic similarity, often outperforming sparse methods in more nuanced or ambiguous queries.

## 4.2 Re-Ranking

The second stage in a typical Rankify workflow is re-ranking. Once an initial set of documents is retrieved, these documents are re-ordered using more computationally complex but more precise models:

- **Pointwise** (e.g., MonoBERT, MonoT5)
- **Pairwise**
- **Listwise** (e.g., RankT5, RankGPT, LiT5)

This step is particularly essential when you need to extract high-precision results from an ample set of initially retrieved documents.

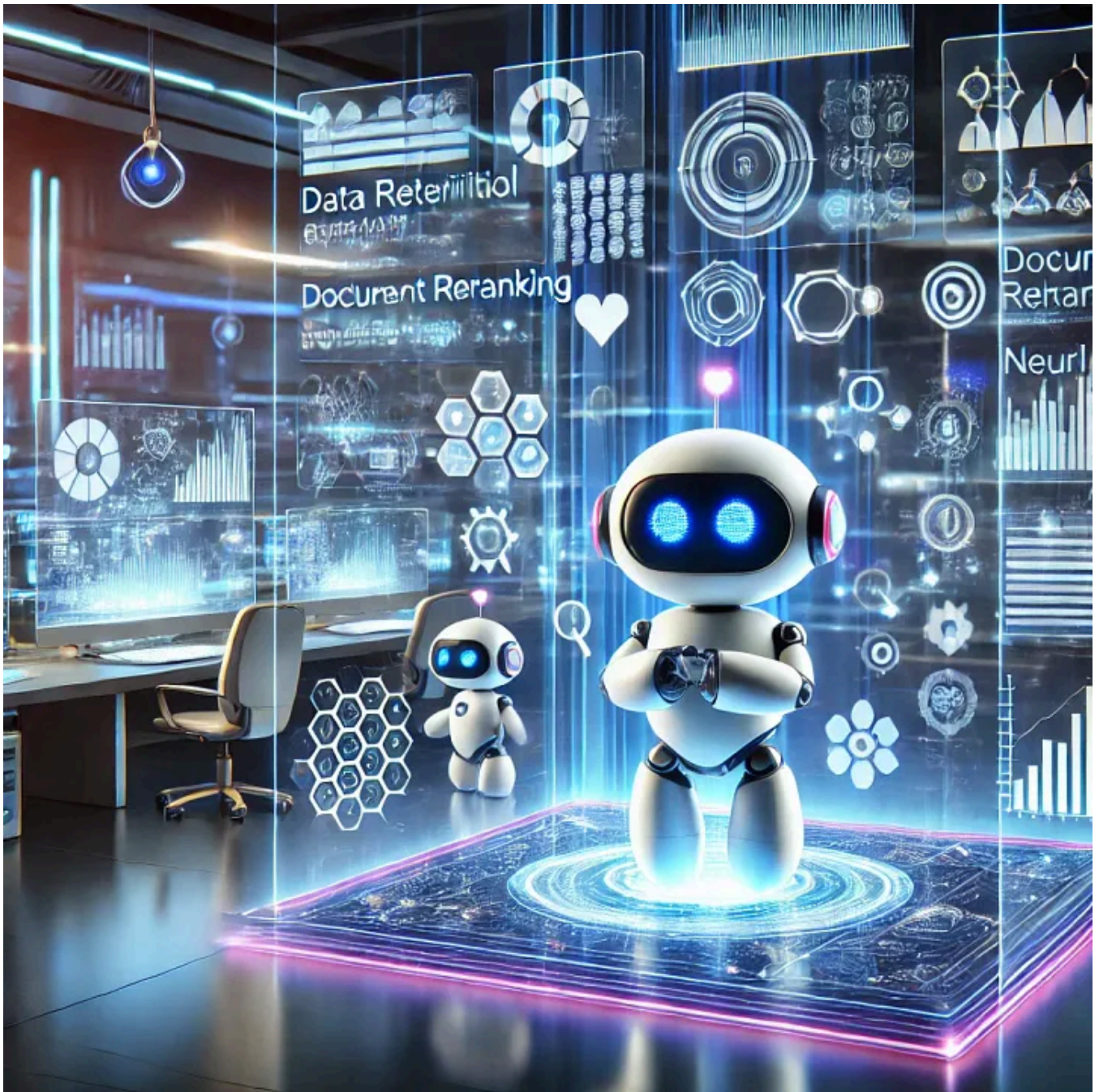
## 4.3 Retrieval-Augmented Generation (RAG)

RAG in Rankify integrates the best of both retrieval and generation:

1. **Retriever** fetches the top documents relevant to a user query.
2. **Generator** then takes these top documents and produces a contextually grounded answer.

Supported methods include zero-shot generation, **Fusion-in-Decoder (FiD)** approaches, in-context learning (RALM), and more. By anchoring language models to real, external data, the risk of factual hallucination diminishes significantly.





Created by Dall-E3

## 5. Installation and Setup

### 5.1 Virtual Environment Setup

Before installing Rankify, it's best practice to create a dedicated **conda environment** that prevents dependency conflicts:

```
conda create -n rankify python=3.10  
conda activate rankify
```

### 5.2 Installing PyTorch



## Rankify works seamlessly with PyTorch 2.5.1:

```
pip install torch==2.5.1 torchvision==0.20.1 torchaudio==2.5.1 --index-url http
```

If you have a GPU and want optimized training/retrieval performance, install the **CUDA version (12.4 or 12.6)** for PyTorch.

## 5.3 Installing Rankify

### Basic Installation:

```
pip install rankify
```

This equips you with the base functionalities for retrieval, re-ranking, and retrieval-augmented generation.

### Recommended Installation:

```
pip install "rankify[all]"
```

This command installs the entire suite — covering advanced retrievers, re-rankers, and generator modules.

### Optional Dependencies

If you only want to install specific components:

- **Retriever only:**

```
pip install "rankify[retriever]"
```

- **Reranking only**

```
pip install "rankify[reranking]"
```

- **Installation from GitHub** (for the latest dev version):

```
git clone https://github.com/DataScienceUIBK/rankify.git cd rankify pip install
```

## 5.4 Using ColBERT Retriever

If you plan to use ColBERT with Rankify, additional setup is required:

1. **Install GCC and required libraries:**

```
conda install -c conda-forge gcc=9.4.0 gxx=9.4.0 conda install -c conda-forge l
```

2. **Export environment variables:**

```
export LD_LIBRARY_PATH=$CONDA_PREFIX/lib:$LD_LIBRARY_PATH export CC=gcc export
```

## 6.1 Pre-Retrieved Datasets

Rankify provides 1,000 pre-retrieved documents per dataset — an immense help for those who want to hit the ground running without the overhead of building or maintaining an index.

## 6.2 Downloading and Handling Datasets in Rankify

Users can download pre-retrieved documents for popular QA datasets with a single script. For instance, to download BM25-retrieved documents for Natural Questions:

```
from rankify.dataset.dataset import Dataset
dataset = Dataset(retriever="bm25", dataset_name="nq-dev", n_docs=100)
```

```
documents = dataset.download(force_download=False)
```

You can swap out “bm25” with “dpr”, “ance”, “colbert”, “bge”, “contriever”, or “mss” to explore how different retrievers retrieve the same dataset.

## 6.3 Running Retrieval

Rankify provides a simple, uniform interface for a wide spectrum of retrieval methods. Here’s how to retrieve documents with BM25 on Wikipedia:

```
from rankify.dataset.dataset import Document, Question, Answer
from rankify.retrievers.retriever import Retriever
documents = [
    Document(question=Question("Who wrote Hamlet?"),
              answers=Answer(["Shakespeare"]), contexts=[])
]
bm25_retriever_wiki = Retriever(method="bm25", n_docs=5, index_type="wiki")
retrieved_docs = bm25_retriever_wiki.retrieve(documents)
for doc in retrieved_docs:
    print(doc)
```

## 6.4 Running Re-Ranking

Once you’ve retrieved the top documents, feed them into a re-ranker to refine the order. Rankify supports more than 20 re-ranking models:

```
from rankify.dataset.dataset import Document, Question, Answer, Context
from rankify.models.reranking import Reranking
question = Question("When did Thomas Edison invent the light bulb?")
answers = Answer(["1879"])
contexts = [
    Context(text="Lightning strike at Seoul National University", id=1),
    Context(text="Thomas Edison invented the light bulb in 1879", id=2),
]
doc = Document(question=question, answers=answers, contexts=contexts)
reranker = Reranking(method="monot5", model_name="monot5-base-msmarco")
reranker.rank([doc])
print(doc.reorder_contexts)
```

After re-ranking, `doc.reorder_contexts` will reflect the new ordering, emphasizing passages that best match the user's query.

## 6.5 Using Generator Module

Rankify's generator module integrates retrieval-augmented generation (RAG) into the workflow. Here's an example of retrieving some contexts and then using a generative model to produce a final answer:

```
from rankify.dataset.dataset import Document, Question, Answer, Context
from rankify.generator.generator import Generator
question = Question("What is the capital of France?")
answers = Answer(["Paris"])
contexts = [
    Context(text="The capital of France is Paris.", id=1),
    Context(text="Berlin is the capital of Germany.", id=2),
]
doc = Document(question=question, answers=answers, contexts=contexts)
generator = Generator(method="in-context-ralm", model_name='meta-llama/Llama-3.1-70B-Instruct')
output = generator.generate([doc])
print(output)
```

This approach is invaluable when you want to ensure your language model not only has the question but also the relevant context at hand.

## 7. Evaluation Metrics

### 7.1 Measuring Retrieval Performance

Rankify provides top-k accuracy measures for retrieval:

```
from rankify.metrics.metrics import Metrics
m = Metrics(documents)
before_rank = m.calculate_retrieval_metrics(ks=[1,5,10,20,50,100], use_reorder=True)
print(before_rank)
```

For each k (1, 5, 10, 20, 50, 100), Rankify checks if the correct answer is found in the top k retrieved passages.

### 7.2 Assessing Re-Ranking Effectiveness

In many experiments, you'll want to see whether re-ranking models actually improve the outcome. Simply toggle `use_reordered=True` to see the difference:

```
after_rank = m.calculate_retrieval_metrics(ks=[1,5,10,20,50,100], use_reordered=True)
print(after_rank)
```

## 7.3 Evaluating Retrieval-Augmented Generation

Rankify also calculates metrics like **Exact Match (EM)**, **Precision**, **Recall**, and **F1** for your final, generated answers. Such metrics are crucial for QA and knowledge-grounded generation tasks:

```
gen_metrics = m.calculate_generation_metrics(generated_answers)
print(gen_metrics)
```

## 8. Use Cases and Best Practices

### 8.1 QA Systems

If you're building an FAQ or answering open-domain queries — like medical, scientific, or enterprise knowledge bases — Rankify excels in pulling the right documents, re-ranking them with high precision, and optionally generating a concise, accurate response.

### 8.2 Knowledge-Based Assistants

Many chatbots fail because they rely purely on large language models without anchored references. Rankify reduces the risk of “hallucination” by merging external references from retrieval and re-ranking with generative abilities. The end result is a more grounded and reliable assistant.

### 8.3 Academic Research and Benchmarking

Researchers can leverage Rankify's pre-retrieved datasets to conduct controlled experiments quickly. They can swap in new retrieval or re-ranking models and directly compare metrics without re-inventing the wheel each time.



“If we knew what we were doing, it wouldn’t be called research, would it?” — Albert Einstein

Rankify embodies this philosophy: it frees researchers from mundane tasks of implementing standard retrieval pipelines, letting them focus on the novelty of their work.

## 9. Why Rankify Stands Out

### 9.1 Unified Framework

Instead of juggling half a dozen tools, you can handle retrieval, re-ranking, and RAG from one interface. This cohesiveness cuts development time and mitigates integration bugs.

### 9.2 Adaptability and Modularity

Rankify is designed in a plug-and-play manner. Adding a new retriever or re-ranker typically involves implementing an interface that can seamlessly slot into the existing pipeline. This modularity fosters innovation because users can quickly experiment with fresh ideas.

### 9.3 Community and Ecosystem

As an open-source project, Rankify benefits from community contributions and active maintenance. With thorough documentation and an expanding user base, it stands ready to evolve with the latest advances in NLP.

“Coming together is a beginning, staying together is progress, and working together is success.” — Henry Ford

In the spirit of Ford’s words, Rankify thrives because it unifies the IR community under one flexible, powerful framework.

## 10. Famous Quotes for Inspiration

- **Walt Disney:** “All our dreams can come true, if we have the courage to pursue them.”

*Context:* Encourages bold, forward-thinking research in IR.

- **Thomas Edison:** “There’s a way to do it better — find it.”

*Context:* Symbolic of how Rankify aims to continually push the boundaries of retrieval and re-ranking.

## 11. Conclusion

Rankify epitomizes a **next-generation toolkit** that unifies the entire pipeline of retrieval, re-ranking, and retrieval-augmented generation. It addresses the complexities and fragmentation in modern IR systems by offering a cohesive, modular platform. Users can easily experiment with an array of retrievers, from the venerable BM25 to advanced neural approaches like DPR and ColBERT. They can further refine results with sophisticated re-rankers (MonoT5, RankT5, LiT5, RankGPT, and many more) and then produce contextually relevant answers using RAG.

Unlike the **chain-of-thought** paradigm that relies primarily on an internal, opaque mechanism, Rankify cements its answers in **explicit external context**. This distinction is critical for tasks demanding verifiable accuracy, factual correctness, and robust performance. By focusing on how and where the relevant data is sourced and organized, Rankify sidesteps pitfalls of hallucination, effectively bridging the gap between theoretical AI capabilities and tangible, real-world reliability.



Created by Dall-E3

Rankify

Reranking

Retrieval Augmented Gen

Artificial Intelligence

Data Retrieval



Follow

Published in Level Up Coding

220K Followers · Last published 18 hours ago

Coding tutorials and news. The developer homepage [gitconnected.com](#) && [skilled.dev](#) && [levelup.dev](#)



Follow

## Written by AI TutorMaster

3.1K Followers · 5.3K Following

Senior Data Scientist | Data Engineer| PhD UX Design | Tech Writer | Chatbot Development

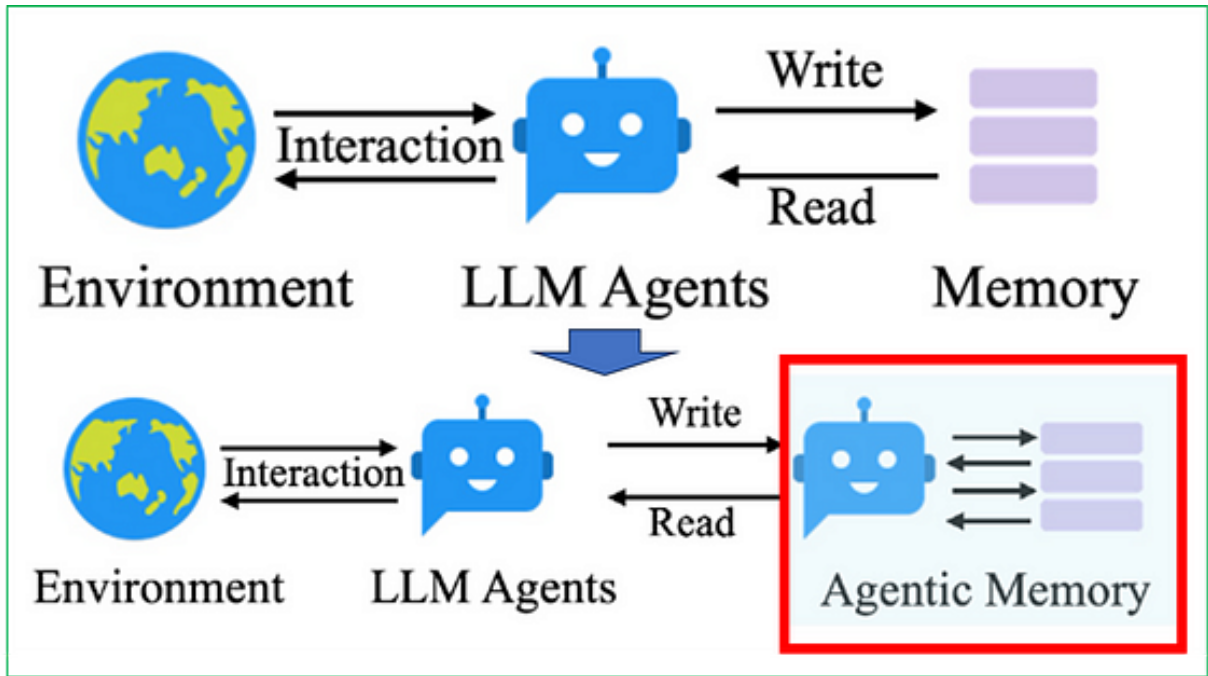
### No responses yet




navneet chaudhary

What are your thoughts?

### More from AI TutorMaster and Level Up Coding



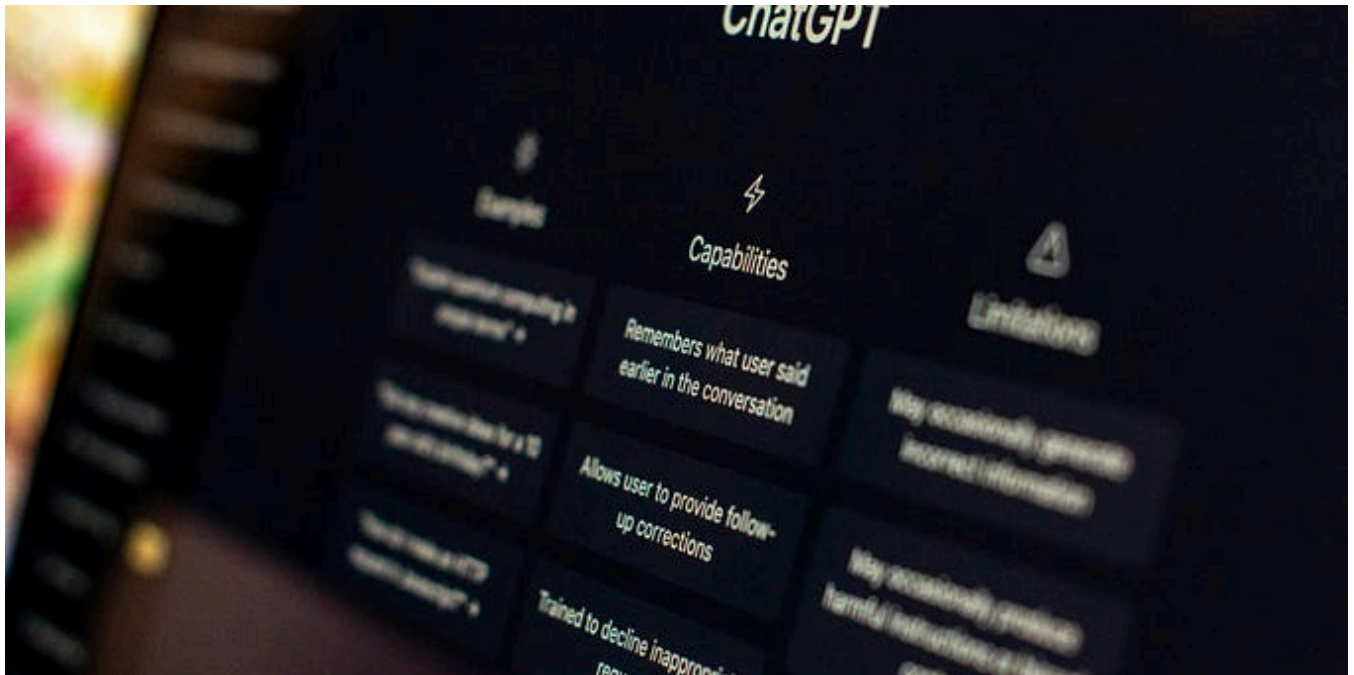
 In Level Up Coding by AI TutorMaster


## Agentic Memory(A-MEM): Long-Term Knowledge Management for LLM Agents

Adaptive Knowledge Networks for Smarter, More Context-Aware LLMs

★ Mar 9 🖱️ 21 💬 1

🔖 ⋮



 In Level Up Coding by Crafting-Code

## The ChatGPT Hack Top 1% Developers Use to Write Code 10x Faster

Stop Wasting Time on Basic Prompts—Here's the Framework Nobody Shares