

NirDiamant / RAG_Techniques

<> Code

Issues

Pull requests 2

Actions

Projects

Security

Insights

This repository showcases various advanced techniques for Retrieval-Augmented Generation (RAG) systems. RAG systems combine information retrieval with generative models to provide accurate and contextually rich responses.

View license

☆ 7.4k stars

🔗 753 forks

👁 96 watching

🔑

📁

📈 Activity

Branches

Tags

🌐

 Public repository

1 Branch

0 Tags

🔍 Go to file

t

Go to file

+

Add file ▾

<> Code ▾

...

NirDiamant

updated README

0c414d6 · 1 hour ago 🕒

📁 .github	added workflows	last month
📁 all_rag_techniques	cleared some cell outputs	3 weeks ago
📁 all_rag_techniques_runnabl...	Fixes Update from llama_index.cor...	3 days ago
📁 data	merge	last month
📁 evaluation	feat: add test set meta evaluation a...	2 weeks ago
📁 images	feat: add mermaid schema	2 weeks ago
📁 tests	fix github action config	last month
📄 .gitignore	updated gitignore	2 weeks ago
📄 CONTRIBUTING.md	updated CONTRIBUTING.md	2 months ago
📄 LICENSE	updated LICENSE	last week
📄 README.md	updated README	1 hour ago
📄 helper_functions.py	Update helper_functions.py	last week
📄 requirements.txt	ollama removed	last month

📖 README

📄 License

PRs welcome

LinkedIn

Connect

X Follow @NirDiamantAI

Discord

Join our community

https://github.com/NirDiamant/RAG_Techniques

1/13

🌟 **Support This Project:** Your sponsorship fuels innovation in RAG technologies. [Become a sponsor](#) to help maintain and expand this valuable resource!

Advanced RAG Techniques: Elevating Your Retrieval-Augmented Generation Systems 🚀

Welcome to one of the most comprehensive and dynamic collections of Retrieval-Augmented Generation (RAG) tutorials available today. This repository serves as a hub for cutting-edge techniques aimed at enhancing the accuracy, efficiency, and contextual richness of RAG systems.

📧 Stay Updated!

Don't miss out on cutting-edge developments, new tutorials, and community insights!

[Subscribe to the RAG Techniques Newsletter of DiamantAI](#)

Introduction

Retrieval-Augmented Generation (RAG) is revolutionizing the way we combine information retrieval with generative AI. This repository showcases a curated collection of advanced techniques designed to supercharge your RAG systems, enabling them to deliver more accurate, contextually relevant, and comprehensive responses.

Our goal is to provide a valuable resource for researchers and practitioners looking to push the boundaries of what's possible with RAG. By fostering a collaborative environment, we aim to accelerate innovation in this exciting field.

Related Projects

✍️ Check out my [Prompt Engineering Techniques guide](#) for a comprehensive collection of prompting strategies, from basic concepts to advanced techniques, enhancing your ability to interact effectively with AI language models.

🤖 Explore my [GenAI Agents Repository](#) to discover a variety of AI agent implementations and tutorials, showcasing how different AI technologies can be combined to create powerful, interactive systems.

A Community-Driven Knowledge Hub

This repository grows stronger with your contributions! Join our vibrant Discord community — the central hub for shaping and advancing this project together 🤝

[RAG Techniques Discord Community](#)

Whether you're an expert or just starting out, your insights can shape the future of RAG. Join us to propose ideas, get feedback, and collaborate on innovative techniques. For contribution guidelines, please refer to our [CONTRIBUTING.md](#) file. Let's advance RAG technology together!

🔗 For discussions on GenAI, RAG, or custom agents, or to explore knowledge-sharing opportunities, feel free to [connect on LinkedIn](#).

Key Features

- 🧠 State-of-the-art RAG enhancements
- 📖 Comprehensive documentation for each technique
- 🔧 Practical implementation guidelines
- 🌟 Regular updates with the latest advancements

Advanced Techniques

Explore the extensive list of cutting-edge RAG techniques:

🌱 Foundational RAG Techniques

1. Simple RAG 🌱

- [LangChain](#)
- [LlamaIndex](#)
- [Runnable Script](#)

Overview 🔍

Introducing basic RAG techniques ideal for newcomers.

Implementation 🔧

Start with basic retrieval queries and integrate incremental learning mechanisms.

2. Simple RAG using a CSV file 🧩

- [LangChain](#)
- [LlamaIndex](#)

Overview 🔍

Introducing basic RAG using CSV files.

Implementation 🔧

This uses CSV files to create basic retrieval and integrates with openai to create question and answering system.

3. [Reliable RAG](#) 📄

Overview 🔍

Enhances the Simple RAG by adding validation and refinement to ensure the accuracy and relevance of retrieved information.

Implementation

Check for retrieved document relevancy and highlight the segment of docs used for answering.

4. Choose Chunk Size

- [LangChain](#)
- [Runnable Script](#)

Overview

Selecting an appropriate fixed size for text chunks to balance context preservation and retrieval efficiency.

Implementation



Experiment with different chunk sizes to find the optimal balance between preserving context and maintaining retrieval speed for your specific use case.

5. [Proposition Chunking](#)

Overview

Breaking down the text into concise, complete, meaningful sentences allowing for better control and handling of specific queries (especially extracting knowledge).

Implementation

-  **Proposition Generation:** The LLM is used in conjunction with a custom prompt to generate factual statements from the document chunks.
-  **Quality Checking:** The generated propositions are passed through a grading system that evaluates accuracy, clarity, completeness, and conciseness.

Additional Resources

- [The Propositions Method: Enhancing Information Retrieval for AI Systems](#) - A comprehensive blog post exploring the benefits and implementation of proposition chunking in RAG systems.

Query Enhancement



6. Query Transformations

- [LangChain](#)
- [Runnable Script](#)

Overview

Modifying and expanding queries to improve retrieval effectiveness.

Implementation

-  **Query Rewriting:** Reformulate queries to improve retrieval.
-  **Step-back Prompting:** Generate broader queries for better context retrieval.

-  **Sub-query Decomposition:** Break complex queries into simpler sub-queries.

7. Hypothetical Questions (HyDE Approach)

- [LangChain](#)
- [Runnable Script](#)

Overview

Generating hypothetical questions to improve alignment between queries and data.

Implementation

Create hypothetical questions that point to relevant locations in the data, enhancing query-data matching.

Additional Resources

- [HyDE: Exploring Hypothetical Document Embeddings for AI Retrieval](#) - A short blog post explaining this method clearly.

Context and Content Enrichment

8. [Contextual Chunk Headers](#)

Overview

Contextual chunk headers (CCH) is a method of creating document-level and section-level context, and prepending those chunk headers to the chunks prior to embedding them.

Implementation

Create a chunk header that includes context about the document and/or section of the document, and prepend that to each chunk in order to improve the retrieval accuracy.

Additional Resources

[dsRAG](#): open-source retrieval engine that implements this technique (and a few other advanced RAG techniques)

9. [Relevant Segment Extraction](#)

Overview

Relevant segment extraction (RSE) is a method of dynamically constructing multi-chunk segments of text that are relevant to a given query.

Implementation

Perform a retrieval post-processing step that analyzes the most relevant chunks and identifies longer multi-chunk segments to provide more complete context to the LLM.

10. Context Enrichment Techniques

- [LangChain](#)
- [LlamaIndex](#)
- [Runnable Script](#)

Overview 🔍

Enhancing retrieval accuracy by embedding individual sentences and extending context to neighboring sentences.

Implementation 🛠️

Retrieve the most relevant sentence while also accessing the sentences before and after it in the original text.

11. Semantic Chunking 🧠

- [LangChain](#)
- [Runnable Script](#)

Overview 🔍

Dividing documents based on semantic coherence rather than fixed sizes.

Implementation 🛠️

Use NLP techniques to identify topic boundaries or coherent sections within documents for more meaningful retrieval units.

Additional Resources 📖

- [Semantic Chunking: Improving AI Information Retrieval](#) - A comprehensive blog post exploring the benefits and implementation of semantic chunking in RAG systems.

12. Contextual Compression 🗄️

- [LangChain](#)
- [Runnable Script](#)

Overview 🔍

Compressing retrieved information while preserving query-relevant content.

Implementation 🛠️

Use an LLM to compress or summarize retrieved chunks, preserving key information relevant to the query.

13. Document Augmentation through Question Generation for Enhanced Retrieval

- [LangChain](#)
- [Runnable Script](#)

Overview 🔍

This implementation demonstrates a text augmentation technique that leverages additional question generation to improve document retrieval within a vector database. By generating and incorporating various questions related to each text fragment, the system enhances the standard retrieval process, thus increasing the likelihood of finding relevant documents that can be utilized as context for generative question answering.

Implementation

Use an LLM to augment text dataset with all possible questions that can be asked to each document.

Advanced Retrieval Methods

14. Fusion Retrieval

- [LangChain](#)
- [LlamaIndex](#)
- [Runnable Script](#)

Overview

Optimizing search results by combining different retrieval methods.

Implementation

Combine keyword-based search with vector-based search for more comprehensive and accurate retrieval.




15. Intelligent Reranking

- [LangChain](#)
- [LlamaIndex](#)
- [Runnable Script](#)

Overview

Applying advanced scoring mechanisms to improve the relevance ranking of retrieved results.

Implementation

-  **LLM-based Scoring:** Use a language model to score the relevance of each retrieved chunk.
-  **Cross-Encoder Models:** Re-encode both the query and retrieved documents jointly for similarity scoring.
-  **Metadata-enhanced Ranking:** Incorporate metadata into the scoring process for more nuanced ranking.

Additional Resources

- [Relevance Revolution: How Re-ranking Transforms RAG Systems](#) - A comprehensive blog post exploring the power of re-ranking in enhancing RAG system performance.

16. Multi-faceted Filtering

Overview 🔍

Applying various filtering techniques to refine and improve the quality of retrieved results.

Implementation 🛠️

- 📁 **Metadata Filtering:** Apply filters based on attributes like date, source, author, or document type.
- 🇮🇱 **Similarity Thresholds:** Set thresholds for relevance scores to keep only the most pertinent results.
- 📄 **Content Filtering:** Remove results that don't match specific content criteria or essential keywords.
- 🌈 **Diversity Filtering:** Ensure result diversity by filtering out near-duplicate entries.

17. Hierarchical Indices 📁

- [LangChain](#)
- [Runnable Script](#)

Overview 🔍

Creating a multi-tiered system for efficient information navigation and retrieval.

Implementation 🛠️

Implement a two-tiered system for document summaries and detailed chunks, both containing metadata pointing to the same location in the data.

Additional Resources 📖

- [Hierarchical Indices: Enhancing RAG Systems](#) - A comprehensive blog post exploring the power of hierarchical indices in enhancing RAG system performance.

18. Ensemble Retrieval 🤖

Overview 🔍

Combining multiple retrieval models or techniques for more robust and accurate results.

Implementation 🛠️

Apply different embedding models or retrieval algorithms and use voting or weighting mechanisms to determine the final set of retrieved documents.

19. Multi-modal Retrieval 🧑‍🎤

Overview 🔍

Extending RAG capabilities to handle diverse data types for richer responses.

Implementation 🛠️

- [Multi-model RAG with Multimedia Captioning](#) - Caption and store all the other multimedia data like pdfs, ppts, etc., with text data in vector store and retrieve them together.
- [Multi-model RAG with Colpali](#) - Instead of captioning convert all the data into image, then find the most relevant images and pass them to a vision large language model.

Iterative and Adaptive Techniques

20. Retrieval with Feedback Loops

- [LangChain](#)
- [Runnable Script](#)

Overview

Implementing mechanisms to learn from user interactions and improve future retrievals.

Implementation

Collect and utilize user feedback on the relevance and quality of retrieved documents and generated responses to fine-tune retrieval and ranking models.

21. Adaptive Retrieval

- [LangChain](#)
- [Runnable Script](#)

Overview

Dynamically adjusting retrieval strategies based on query types and user contexts.

Implementation

Classify queries into different categories and use tailored retrieval strategies for each, considering user context and preferences.

22. Iterative Retrieval

Overview

Performing multiple rounds of retrieval to refine and enhance result quality.

Implementation

Use the LLM to analyze initial results and generate follow-up queries to fill in gaps or clarify information.

Evaluation

23. [DeepEval Evaluation](#)

Overview

Performing evaluations Retrieval-Augmented Generation systems, by covering several metrics and creating test cases.

Implementation

Use the `deepeval` library to conduct test cases on correctness, faithfulness and contextual relevancy of RAG systems.

24. [GroUSE Evaluation](#)

Overview

Evaluate the final stage of Retrieval-Augmented Generation using metrics of the GroUSE framework and meta-evaluate your custom LLM judge on GroUSE unit tests.

Implementation

Use the `grouse` package to evaluate contextually-grounded LLM generations with GPT-4 on the 6 metrics of the GroUSE framework and use unit tests to evaluate a custom Llama 3.1 405B evaluator.

Explainability and Transparency

25. Explainable Retrieval

- [LangChain](#)
- [Runnable Script](#)

Overview

Providing transparency in the retrieval process to enhance user trust and system refinement.

Implementation

Explain why certain pieces of information were retrieved and how they relate to the query.

Advanced Architectures

26. Knowledge Graph Integration (Graph RAG)

- [LangChain](#)
- [Runnable Script](#)

Overview

Incorporating structured data from knowledge graphs to enrich context and improve retrieval.

Implementation

Retrieve entities and their relationships from a knowledge graph relevant to the query, combining this structured data with unstructured text for more informative responses.

27. GraphRag (Microsoft)

- [GraphRag](#)

Overview

Microsoft GraphRAG (Open Source) is an advanced RAG system that integrates knowledge graphs to improve the performance of LLMs

Implementation

- Analyze an input corpus by extracting entities, relationships from text units. generates summaries of each community and its constituents from the bottom-up.

28. RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval

- [LangChain](#)
- [Runnable Script](#)

Overview

Implementing a recursive approach to process and organize retrieved information in a tree structure.

Implementation

Use abstractive summarization to recursively process and summarize retrieved documents, organizing the information in a tree structure for hierarchical context.

29. Self RAG

- [LangChain](#)
- [Runnable Script](#)

Overview

A dynamic approach that combines retrieval-based and generation-based methods, adaptively deciding whether to use retrieved information and how to best utilize it in generating responses.

Implementation

- Implement a multi-step process including retrieval decision, document retrieval, relevance evaluation, response generation, support assessment, and utility evaluation to produce accurate, relevant, and useful outputs.

30. Corrective RAG

- [LangChain](#)
- [Runnable Script](#)

Overview

A sophisticated RAG approach that dynamically evaluates and corrects the retrieval process, combining vector databases, web search, and language models for highly accurate and context-aware responses.

Implementation

- Integrate Retrieval Evaluator, Knowledge Refinement, Web Search Query Rewriter, and Response Generator components to create a system that adapts its information sourcing strategy based on relevance scores and combines multiple sources when necessary.

🌟 Special Advanced Technique 🌟

31. [Sophisticated Controllable Agent for Complex RAG Tasks](#) 🤖

Overview 🔍

An advanced RAG solution designed to tackle complex questions that simple semantic similarity-based retrieval cannot solve. This approach uses a sophisticated deterministic graph as the "brain" 🧠 of a highly controllable autonomous agent, capable of answering non-trivial questions from your own data.

Implementation 🛠️

- Implement a multi-step process involving question anonymization, high-level planning, task breakdown, adaptive information retrieval and question answering, continuous re-planning, and rigorous answer verification to ensure grounded and accurate responses.

Getting Started

To begin implementing these advanced RAG techniques in your projects:

1. Clone this repository:

```
git clone https://github.com/NirDiamant/RAG_Techniques.git
```



2. Navigate to the technique you're interested in:

```
cd all_rag_techniques/technique-name
```



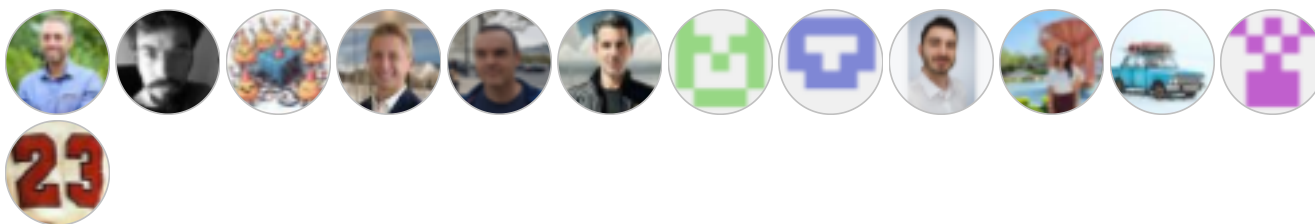
3. Follow the detailed implementation guide in each technique's directory.

Contributing

We welcome contributions from the community! If you have a new technique or improvement to suggest:

1. Fork the repository
2. Create your feature branch: `git checkout -b feature/AmazingFeature`
3. Commit your changes: `git commit -m 'Add some AmazingFeature'`
4. Push to the branch: `git push origin feature/AmazingFeature`
5. Open a pull request

Contributors



License

This project is licensed under a custom non-commercial license - see the [LICENSE](#) file for details.



Releases

No releases published

Sponsor this project

 NirDiamant

 Sponsor

[Learn more about GitHub Sponsors](#)

Packages

No packages published

Contributors 13



Languages

