



Search



Write

Sign up

Sign in



★ Member-only story

Forget RAG, the Future is RAG-Fusion

The Next Frontier of Search: Retrieval Augmented Generation meets Reciprocal Rank Fusion and Generated Queries



Adrian H. Raudaschl · Follow

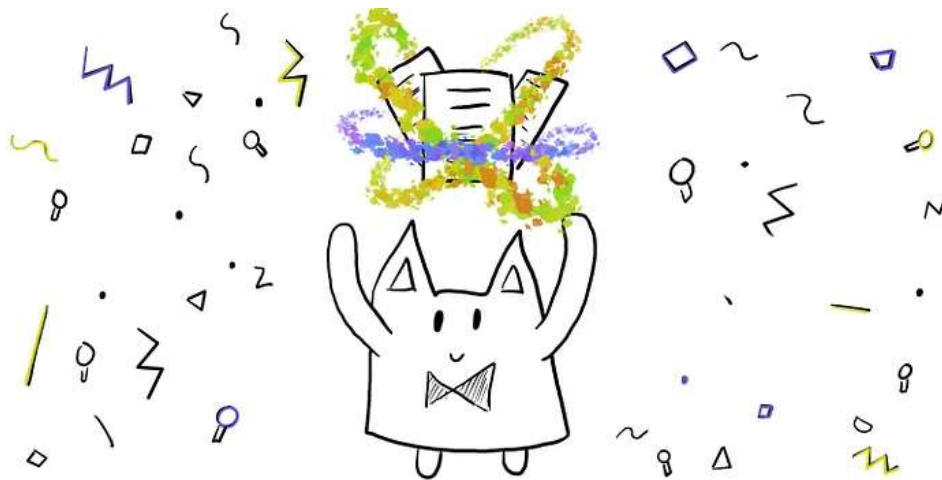
Published in Towards Data Science · 10 min read · Oct 6



2.5K



24

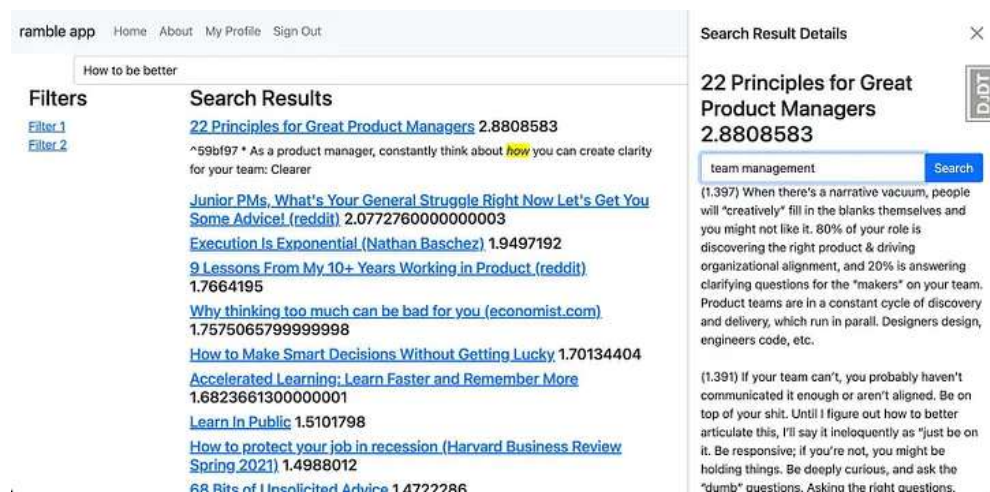


The Wonderful World of RAG Fusion. Illustration by author.

Having explored search technologies for almost a decade, I can honestly say nothing has been as disruptive as the recent rise of Retrieval Augmented Generation (RAG). This system is revolutionising search and information retrieval using vector search with generative AI to produce direct answers based on trusted data.

In my search projects, experimenting with RAG has led me to consider its potential enhancements; I believe RAG is still too limited to meet users'

needs and needs an upgrade.



My personal search system (Project Ramble), where I hooked up my Obsidian notes to a vector search combined with GPT-3 in 2022. Image by author.

Don't get me wrong, RAG is excellent and is absolutely a step in the right direction for information retrieval technologies. I've used RAG since the advent of GPT-2 in 2021, which has significantly helped boost my productivity when looking for valuable information from my own notes or work documents. RAG has many **advantages**:

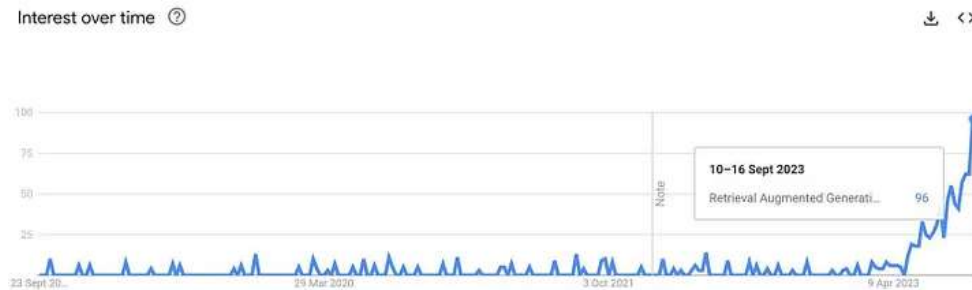
- **Vector Search Fusion:** RAG introduces a novel paradigm by integrating vector search capabilities with generative models. This fusion enables the generation of richer, more context-aware outputs from large language models (LLMs).
- **Reduced Hallucination:** RAG significantly diminishes the LLM's propensity for hallucination, making the generated text more grounded in data.
- **Personal and Professional Utility:** From personal applications like sifting through notes to more professional integrations, RAG showcases versatility in enhancing productivity and content quality while being based on a trustworthy data source.

However, I'm finding more and more **limitations** of RAG:

- **Constraints with Current Search Technologies:** RAG is limited by the same things limiting our retrieval-based lexical and vector search technologies.
- **Human Search Inefficiencies:** Humans are not great at writing what they want into search systems, such as typos, vague queries, or limited

vocabulary, which often lead to missing the vast reservoir of information that lies beyond the obvious top search results. While RAG assists, it hasn't entirely solved this problem.

- **Over-Simplification of Search:** Our prevalent search paradigm linearly maps queries to answers, lacking the depth to understand the multi-dimensional nature of human queries. This linear model often fails to capture the nuances and contexts of more complex user inquiries, resulting in less relevant results.



Searches for RAG (Retrieval Augmented Generation) skyrocketing in 2023. Screenshot by author from Google Trends Sept 2023.

So, what can we do to address these issues? We need a system that doesn't just retrieve what we ask but grasps the nuance behind our queries without needing ever-more advanced LLMs. Recognising these challenges and inspired by the possibilities, I developed a more refined solution: RAG-Fusion.

Why RAG-Fusion?

- **Addressing Gaps:** It tackles the constraints inherent in RAG by generating multiple user queries and reranking the results.
- **Enhanced Search:** Utilises Reciprocal Rank Fusion and custom vector score weighting for comprehensive, accurate results.

RAG-Fusion aspires to bridge the gap between what users explicitly ask and what they intend to ask, inching closer to uncovering the transformative knowledge that typically remains hidden.

Embarking on this journey with RAG years ago, I regret not sharing those initial experiments. But it's time to make amends. Let's delve deep into the technical intricacies of RAG-Fusion.

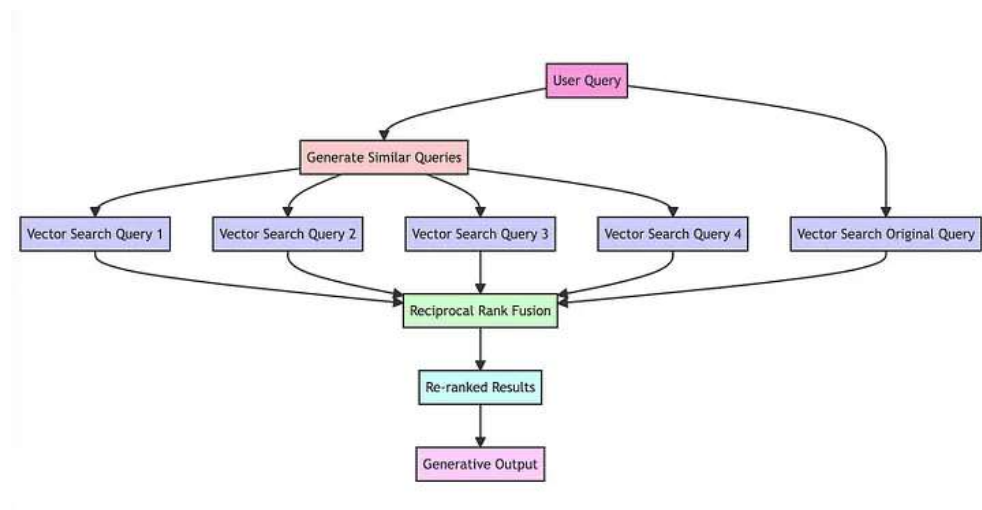
A Deep Dive into the Mechanics of RAG-Fusion

Tools and Tech Stack

FYI, for those of you who just want to see code and get straight into playing with RAG-Fusion, [check out the GitHub repo here](#).

The foundational triad of RAG Fusion is similar to RAG and lies in the same three key technologies:

- A general-purpose programming language, often Python.
- A dedicated vector search database, such as Elasticsearch or Pinecone, steering the document retrieval.
- A potent large language model, like ChatGPT, crafting the text.



An illustrative representation of RAG-Fusion's working mechanism. Image by author.

However, unlike RAG, RAG-Fusion differentiates itself with a few additional steps — query generation and a reranking of the results.

RAG-Fusion's Workflow:

1. **Query Duplication with a Twist:** Translate a user's query into similar, yet distinct queries via an LLM.
2. **Vector Search Unleashed:** Perform vector searches for the original and its newly generated query siblings.
3. **Intelligent Reranking:** Aggregate and refine all the results using reciprocal rank fusion.
4. **Eloquent Finale:** Pair the cherry-picked results with the new queries, guiding the large language model to a crafted output that considers all the queries and the reranked list of results.

```

main.py X
Users > raudaschl > main.py > reciprocal_rank_fusion

31
32 # Reciprocal Rank Fusion algorithm
33 # Reciprocal Rank Fusion algorithm
34 def reciprocal_rank_fusion(search_results_dict, k=60):
35     fused_scores = {}
36     print("Initial individual search result ranks:")
37     for query, doc_scores in search_results_dict.items():
38         print(f"for query '{query}': {doc_scores}")
39
40     for query, doc_scores in search_results_dict.items():
41         for rank, (doc, score) in enumerate(sorted(doc_scores.items(), key=lambda x: x[1], reverse=True)):
42             if doc not in fused_scores:
43                 fused_scores[doc] = 0
44             previous_score = fused_scores[doc]
45             fused_scores[doc] += 1 / (rank + k)
46             print(f"Updating score for {doc} from {previous_score} to {fused_scores[doc]} based on rank {rank} in query '{query}'")
47
48     reranked_results = {doc: score for doc, score in sorted(fused_scores.items(), key=lambda x: x[1], reverse=True)}
49     print("Final reranked results:", reranked_results)
50     return reranked_results
51
52 # Dummy function to simulate generative output
53 def generate_output(reranked_results, queries):
54     return f"Final output based on {queries} and reranked documents: {list(reranked_results.keys())}"
55
56
57 self._interpret_response_line
58 File ~/Users/raudaschl/.pyenv/versions/3.8.0/Lib/python3.8/site-packages/openai/api_requestor.py, line 687, in _interpret_response_line
59     raise self.handle_error_response(
60 openai.error.AuthenticationError: Incorrect API key provided: your-ope*****here. You can find your API key at https://platform.openai.com/account/api-keys.
61
62 Adrians-Mac: raudaschl$ python main.py
63 Initial individual search result ranks:
64 For query '1. Current and future impacts of climate change on ecosystems': {'doc8': 0.82, 'doc9': 0.74}
65 For query '2. Economic consequences of climate change': {'doc10': 0.88, 'doc2': 0.84, 'doc3': 0.81, 'doc8': 0.71}
66 For query '3. Social and cultural impacts of climate change': {'doc8': 0.85, 'doc9': 0.81, 'doc5': 0.79, 'doc2': 0.73}
67 For query '4. Solutions to mitigate the impact of climate change': {'doc9': 0.87, 'doc2': 0.79, 'doc4': 0.75, 'doc5': 0.73, 'doc8': 0.71}
68 Updating score for doc8 from 0 to 0.016666666666666666 based on rank 0 in query '1. Current and future impacts of climate change on ecosystems'
69 Updating score for doc10 from 0 to 0.016666666666666666 based on rank 0 in query '2. Economic consequences of climate change'
70 Updating score for doc2 from 0 to 0.01639344262295882 based on rank 1 in query '2. Economic consequences of climate change'
71 Updating score for doc3 from 0 to 0.016129832258064516 based on rank 2 in query '2. Economic consequences of climate change'
72 Updating score for doc5 from 0.016466666666666666 to 0.032539682539682535 based on rank 3 in query '2. Economic consequences of climate change'
73 Updating score for doc8 from 0.032539682539682535 to 0.0452863452863452 based on rank 0 in query '3. Social and cultural impacts of climate change'
74 Updating score for doc9 from 0.01639344262295882 to 0.03278688524590164 based on rank 1 in query '3. Social and cultural impacts of climate change'
75 Updating score for doc2 from 0.01639344262295882 to 0.03226443459596666 based on rank 3 in query '3. Social and cultural impacts of climate change'
76
77 Ln 36, Col 27 Spaces: 4 UTF-8 LF Python 3.8.0 64-bit (3.8.0: pyenv)

```

RAG-Fusion Code [Example](#). Image by author.

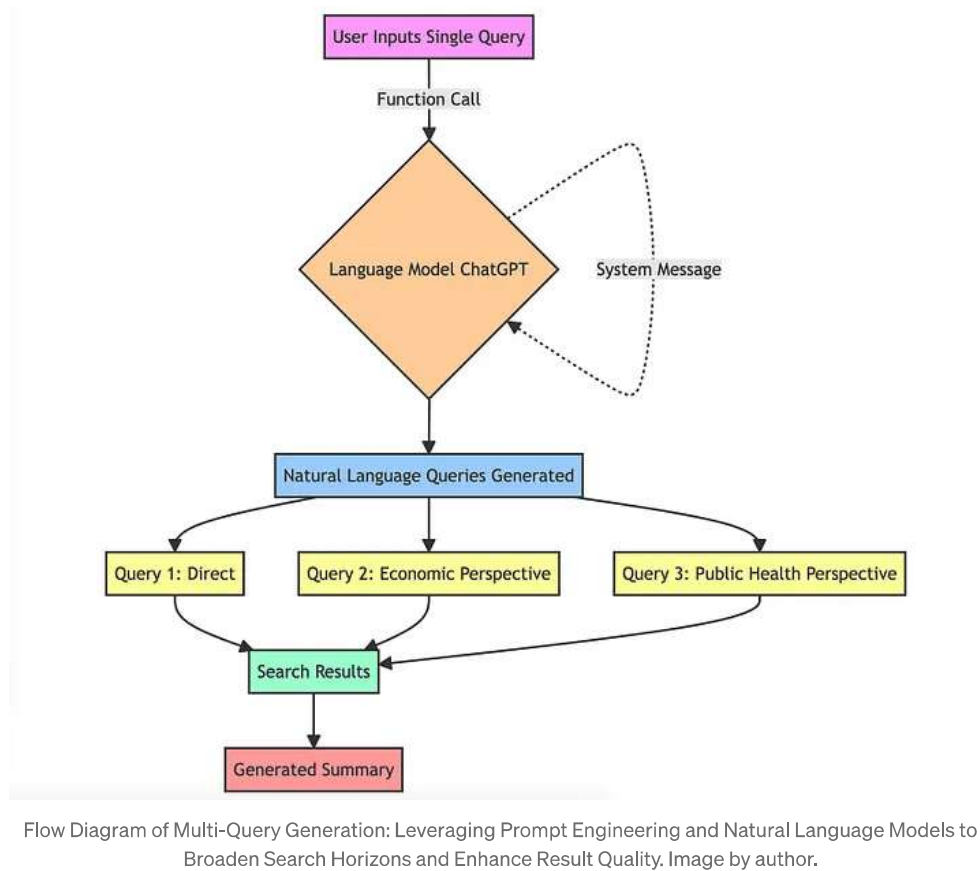
Let's walk through each of these steps in more detail.

Multi-Query Generation

Why Multiple Queries?

In traditional search systems, users often input a single query to find information. While this approach is straightforward, it has limitations. A single query may not capture the full scope of what the user is interested in, or it may be too narrow to yield comprehensive results. This is where generating multiple queries from different perspectives comes into play.

Technical Implementation (Prompt Engineering)



The use of prompt engineering is crucial to generate multiple queries that are not only similar to the original query but also offer different angles or perspectives.

Here's how it works:

- 1. Function Call to Language Model:** The function calls a language model (in this case, chatGPT). This method expects a specific instruction set, often described as a “system message”, to guide the model. For example, the system message here instructs the model to act as an “AI assistant.”
- 2. Natural Language Queries:** The model then generates multiple queries based on the original query.
- 3. Diversity and Coverage:** These queries aren't just random variations. They are carefully generated to offer different perspectives on the original question. For instance, if the original query was about the “impact of climate change,” the generated queries might include angles like “economic consequences of climate change,” “climate change and public health,” etc.

This approach ensures that the search process considers a broader range of information, thereby increasing the quality and depth of the generated

summary.

Reciprocal Rank Fusion (RRF)

Why RRF?

Reciprocal Rank Fusion (RRF) is a technique for combining the ranks of multiple search result lists to produce a single, unified ranking. Developed in collaboration with the University of Waterloo (CAN) and Google, RRF, in the words of its authors, “yields better results than any individual system, and better results than standard” reranking methods.

$$RRFscore(d \in D) = \sum_{r \in R} \frac{1}{k + r(d)},$$

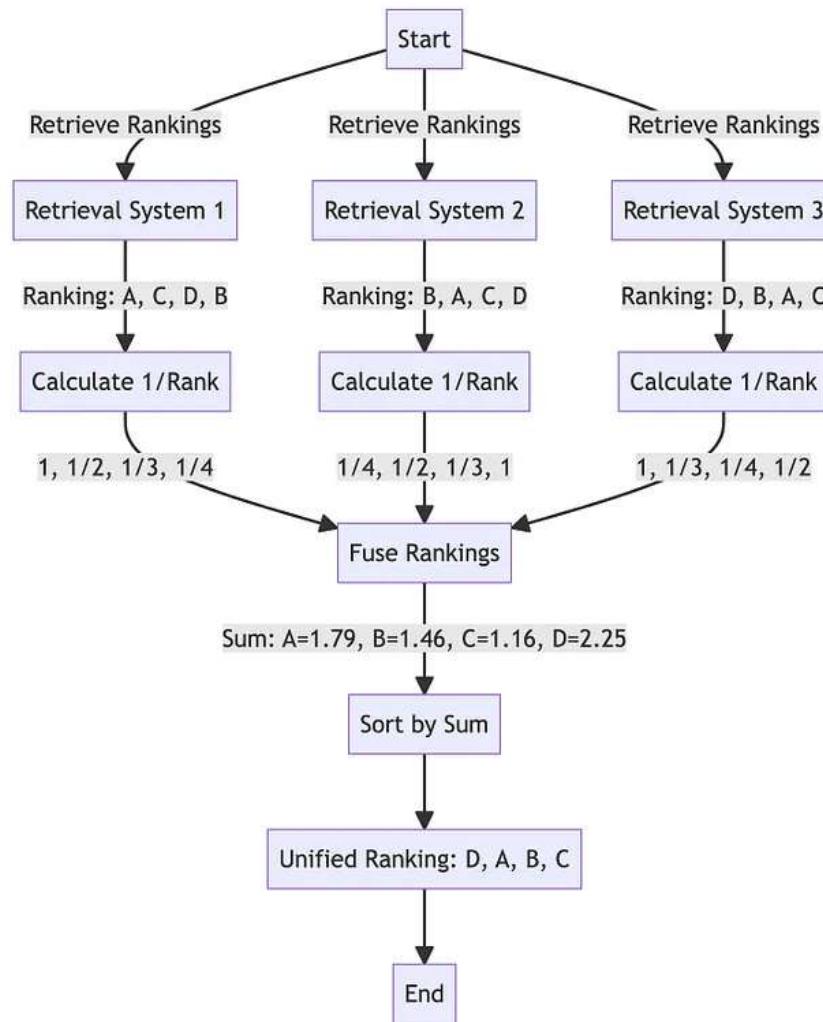
RRF algorithm where k=60. Image from — [Reciprocal Rank Fusion outperforms Condorcet and individual Rank Learning Methods](#)

By combining ranks from different queries, we increase the chances that the most relevant documents will appear at the top of the final list. RRF is particularly effective because it doesn't rely on the absolute scores assigned by the search engine but rather on the relative ranks, making it well-suited for combining results from queries that might have different scales or distributions of scores.

Typically, RRF has been used to blend lexical and vector results. And although that method can help make up for the lack of specificity of vector search when looking up specific terms like acronyms, for example, I've been unimpressed by the results, which tend to be more of a patchwork of multiple result sets as the same results rarely come up for the same query for lexical and vector search.

Think of RRF as that person who insists on getting everyone's opinion before making a decision. Only in this case, it's not annoying but helpful. The more, the merrier — or, in this case, the more accurate.

Technical Implementation



Reciprocal Rank Fusion Positional Reranking System. Image by author.

The function `reciprocal_rank_fusion` takes a dictionary of search results, where each key is a query, and the corresponding value is a list of document IDs ranked by their relevance to that query. The RRF algorithm then calculates a new score for each document based on its ranks in the different lists and sorts them to create a final reranked list.

After calculating the fused scores, the function sorts the documents in descending order of these scores to get the final reranked list, which is then returned.

Generative Output

User Intent Preservation

One of the challenges in using multiple queries is the potential dilution of the user's original intent. To mitigate this, we instruct the model to give more weight to the original query in the prompt engineering.

Technical Implementation

Finally, the reranked documents and all queries are fed into an LLM prompt to produce the generative output in a typical RAG way, like asking for a response or summary.

By layering these technologies and techniques, RAG Fusion offers a powerful, nuanced approach to text generation. It leverages the best of search technology and generative AI to produce high-quality, reliable outputs.

Strengths and Shortcomings of RAG-Fusion

Strengths

1. Superior Source Material Quality

When you use RAG Fusion, the depth of your search isn't merely 'enhanced' — it's amplified. The reranked list of relevant documents means that you're not just scraping the surface of information but diving into an ocean of perspectives. The structured output is easier to read and feels intuitively trustworthy, which is crucial in a world sceptical of AI-generated content.

2. Enhanced User Intent Alignment

At its core, RAG Fusion is designed to be an empathic AI that brings to light what users are striving to express but perhaps can't articulate. Leveraging a multi-query strategy captures a multifaceted representation of the user's informational needs, thus delivering holistic outputs and resonating with user intent.

3. Structured, Insightful Outputs

By drawing from a diverse set of sources, the model crafts well-organised and insightful answers, anticipating follow-up questions and preemptively addressing them.

4. Auto-Correcting User Queries

The system not only interprets but also refines user queries. Through the generation of multiple query variations, RAG Fusion performs implicit spelling and grammar checks, thereby enhancing search result accuracy

5. Navigating Complex Queries

Human language often falters when expressing intricate or specialised thoughts. The system acts as a linguistic catalyst, generating variations that may incorporate the jargon or terminologies required for more focused and relevant search results. It can also take longer, more complex queries and break them down into smaller, manageable chunks for the vector search.

6. Serendipity in Search

Consider the “unknown unknowns” — information you don’t know you need until you encounter it. RAG Fusion allows for this serendipitous discovery. By employing a broader query spectrum, the system engenders the likelihood of unearthing information that, while not explicitly sought, becomes a eureka moment for the user. This sets RAG Fusion apart from other traditional search models.

Challenges

1. The Risk of Being Overly Verbose

RAG-Fusion’s depth can sometimes lead to a deluge of information. Outputs might be detailed to the point of being overwhelming. Think of RAG-Fusion as that friend who over-explains things — informative, but occasionally, you might need them to get to the point.

2. Balancing the Context Window

The inclusion of multi-query input and a diversified document set can stress the language model’s context window. Picture a stage crowded with actors, making it challenging to follow the plot. For models with tight context constraints, this could lead to less coherent or even truncated outputs.

Ethical and User Experience Considerations

With great power comes great responsibility. And with RAG Fusion, the power to manipulate user queries to improve results feels like it’s crossing into some kind of moral grey zone. Balancing the improved search results with the integrity of user intent is crucial, and I’ve got some thoughts you should consider when implementing this solution:

Ethical Concerns:

- **User Autonomy:** The manipulation of user queries can sometimes deviate from the original intent. It’s essential to consider how much control we’re ceding to AI and at what cost.

- **Transparency:** It's not just about better results; users should be aware of and how their queries are adjusted. This transparency is essential to maintain trust and respect user intent.

User Experience (UX) Enhancements:

- **Preserving Original Query:** RAG Fusion prioritises the initial user query, ensuring its importance in the generative process. This acts as a safeguard against misinterpretations.
- **Visibility of Process:** Displaying generated queries alongside final results provides users with a transparent look at the search's scope and depth. It aids in building trust and understanding.

UX/UI Implementation Tips:

- **User Control:** Offer users an option to toggle RAG Fusion, allowing them the choice between manual control and enhanced AI assistance.
- **Guidance & Clarity:** A tooltip or brief explanation about RAG Fusion's workings can help set clear user expectations.

If I had to encapsulate the value of RAG Fusion, it would be this: it moves us closer to what AI was always supposed to do — amplify human potential.

RAG Fusion is not merely an advancement; it's a clarion call to all innovators. It beckons us to step beyond conventional frameworks and reimagine the tapestry of "search".

To those in the realm of search, I pose a challenge: Let's not merely create search systems; let's architect interpreters of inquiry.

Hopefully, RAG-Fusion inspires you to take up that challenge with me.

Dive into the [GitHub repo](#), get your hands dirty with the code, and join the revolution.

You have to start with the customer experience and work backwards to the technology. — Steve Jobs

Bibliography

- If want to read more on the subject, I strongly recommend reading “[AI-Powered Search](#)” by Trey Grainger, Doug Turnbull, and Max Irwin.

Large Language Models

Vector Search

Search Engines

ChatGPT

Machine Learning



2.5K



24



tds

Written by Adrian H. Raudaschl

1.4K Followers · Writer for Towards Data Science

The thoughts and lessons of a physician turned product manager driving search and generative AI innovations.

Follow



More from Adrian H. Raudaschl and Towards Data Science



Adrian H. Raudaschl in UX Collective

Taming the tiger: a product team's guide to risky projects

Be opinionated, embrace criticism, and improve the arguments of others.

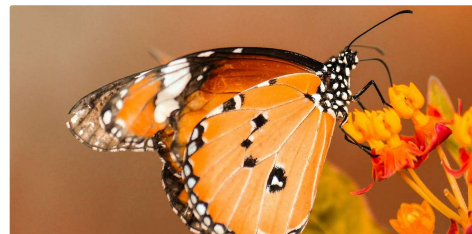
★ · 13 min read · Oct 31



490



4



Marco Peixeiro in Towards Data Science

TimeGPT: The First Foundation Model for Time Series Forecasting

Explore the first generative pre-trained forecasting model and apply it in a project...

★ · 12 min read · Oct 24



2.3K



21





 Rahul Nayak in Towards Data Science


How to Convert Any Text Into a Graph of Concepts

A method to convert any text corpus into a Knowledge Graph using Mistral 7B.

12 min read · Nov 10

 1.5K  25



 Adrian H. Raudaschl in UX Collective

TARS: A product metric game changer

Balance your business goals and user needs by working out if your features are any good

★ · 14 min read · Apr 13

 418  4




See all from Adrian H. Raudaschl

See all from Towards Data Science

Recommended from Medium



 Ignacio de Gregorio

OpenAI Just Killed an Entire Market in 45 Minutes

The Story Everyone Should Have Seen Coming

★ · 6 min read · Nov 9

 13.1K  182



 Ahmed Besbes in Towards Data Science

Why Your RAG is Not Reliable in a Production Environment

And how you should tune it properly 🛠️

★ · 7 min read · Oct 12

 832  13



Lists



The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 202 saves



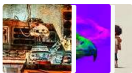
ChatGPT prompts

30 stories · 686 saves



Predictive Modeling w/ Python

20 stories · 612 saves



What is ChatGPT?

9 stories · 220 saves



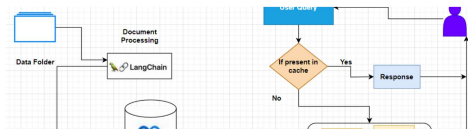
Paul Rose

I Found A Very Profitable AI Side Hustle

And it's perfect for beginners

6 min read · Oct 18

9.1K 169



Kris Ograbek in Towards AI

AutoGen is Mindblowing: 4 Features that Make AutoGen the...

How to Build Your Custom AI Assistant Teams in Minutes.

8 min read · Oct 14

1.2K 6



Plaban Nayak in AI Planet

Advanced RAG Implementation on Custom Data Using Hybrid Searc...

What is RAG ?

40 min read · Oct 8

335 5



Vishal Rajput in AIGuys

RetNet: Transformer killer is here

Can RetNet replace the Transformers? Early results looks very promising.

8 min read · Sep 14

1.2K 9



See more recommendations

