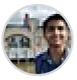
RAG vs Agentic RAG

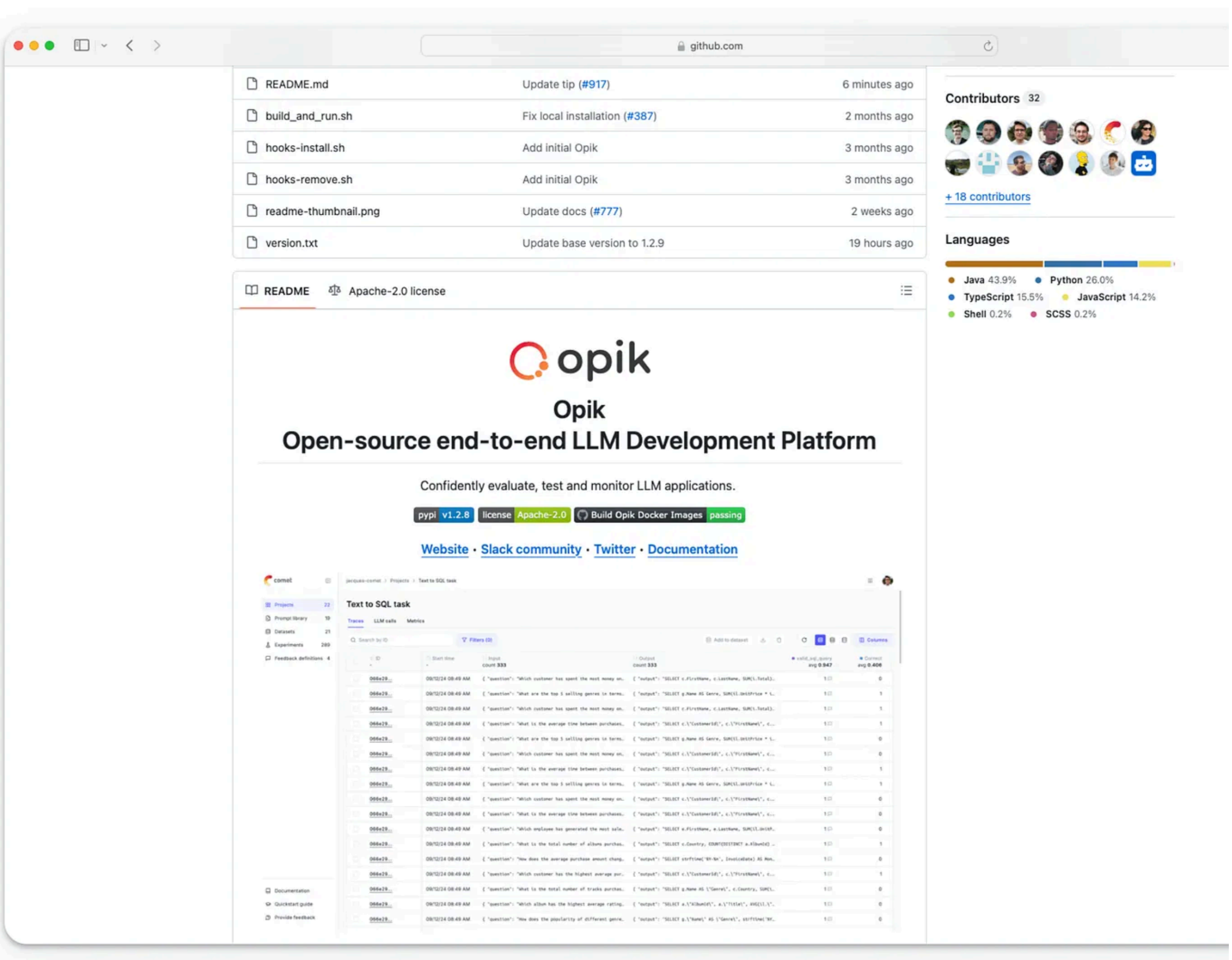
...explained visually.



AVI CHAWLA
DEC 20, 2024

29 2 Sha

[Confidently evaluate, test, and monitor LLM apps in production [OPEN-SOURCE]]



Monitoring and debugging LLMs is necessary but tricky and tedious.

Opik by CometML solves this.

It's an open-source, production-ready end-to-end LLM evaluation platform tha allows developers to test their LLM applications in development, before a relea (CI/CD), and in production.

Here are some key features:

- Record and understand the LLM response generation process.
- Compare many LLM responses in a user-friendly table.
- Log traces during LLM development and production.
- Use built-in LLM judges to detect hallucinations.
- Test the LLM pipeline with different prompts.
- Use its pre-configured evaluation pipeline.

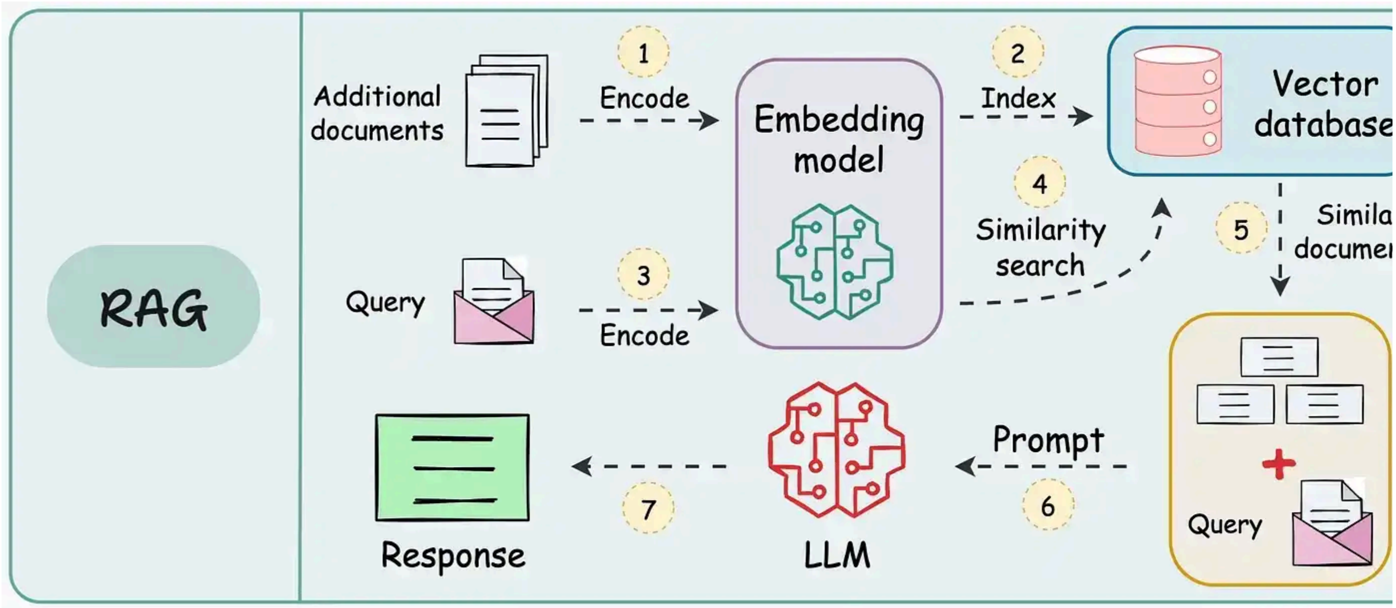
Opik is fully compatible with most LLMs and LLM development frameworks—OpenAI, Pinecone, LlamaIndex, Pinecone, you name it.

Start monitoring your LLM apps in production today →

*Thanks to **CometML** for partnering with us today.*

RAG vs Agentic RAG

These are some issues with the traditional RAG system:



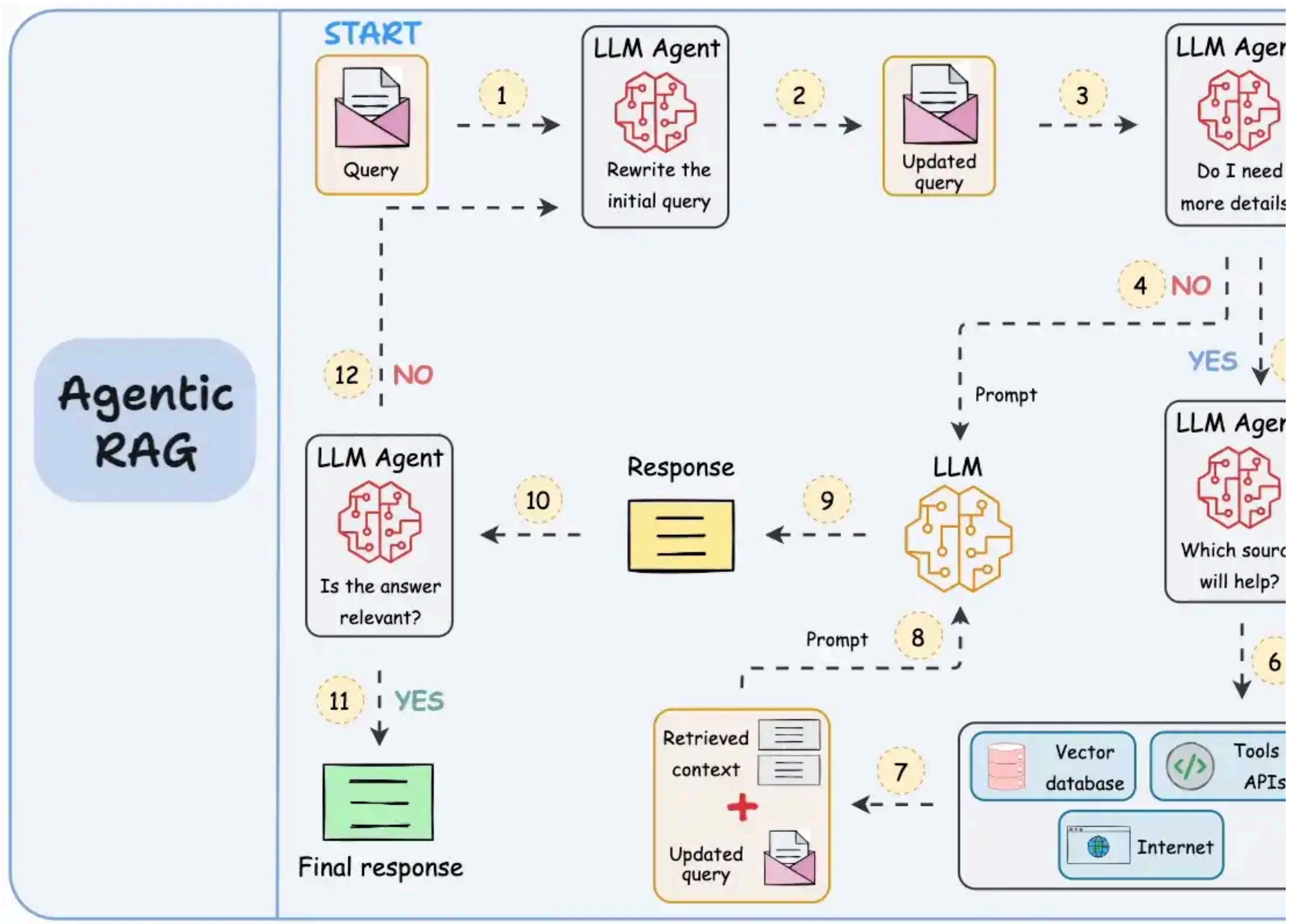
1. These systems retrieve once and generate once. This means if the retrieved context isn't enough, the LLM **can not** dynamically search for more information.
2. RAG systems may provide relevant context but don't reason through compl queries. If a query requires multiple retrieval steps, traditional RAG falls short.
3. There's little adaptability. The LLM can't modify its strategy based on the problem at hand.

Due to this, Agentic RAG is becoming increasingly popular. Let's understand tl today in more detail.

> *On a side note, we started a beginner-friendly crash course on RAGs recently with implementations:*
>
> *Read the first six parts here →*

Agentic RAG

The workflow of agentic RAG is depicted below:



As shown above, the idea is to introduce agentic behaviors at each stage of RAG

> *Think of agents as someone who can actively think through a task—planning, adapting, and iterating until they arrive at the best solution, rathar than just following a defined set of instructions. The powerful capabilities of LLMs make this possible.*

Let's understand this step-by-step:

Steps 1-2) The user inputs the query, and an agent rewrites it (removing spelling mistakes, simplifying it for embedding, etc.)

Step 3) Another agent decides whether it needs more details to answer the quer

- Step 4) If not, the rewritten query is sent to the LLM as a prompt.
- Step 5-8) If yes, another agent looks through the relevant sources it has acc to (vector database, tools & APIs, and the internet) and decides which sourc should be useful. The relevant context is retrieved and sent to the LLM as a prompt.

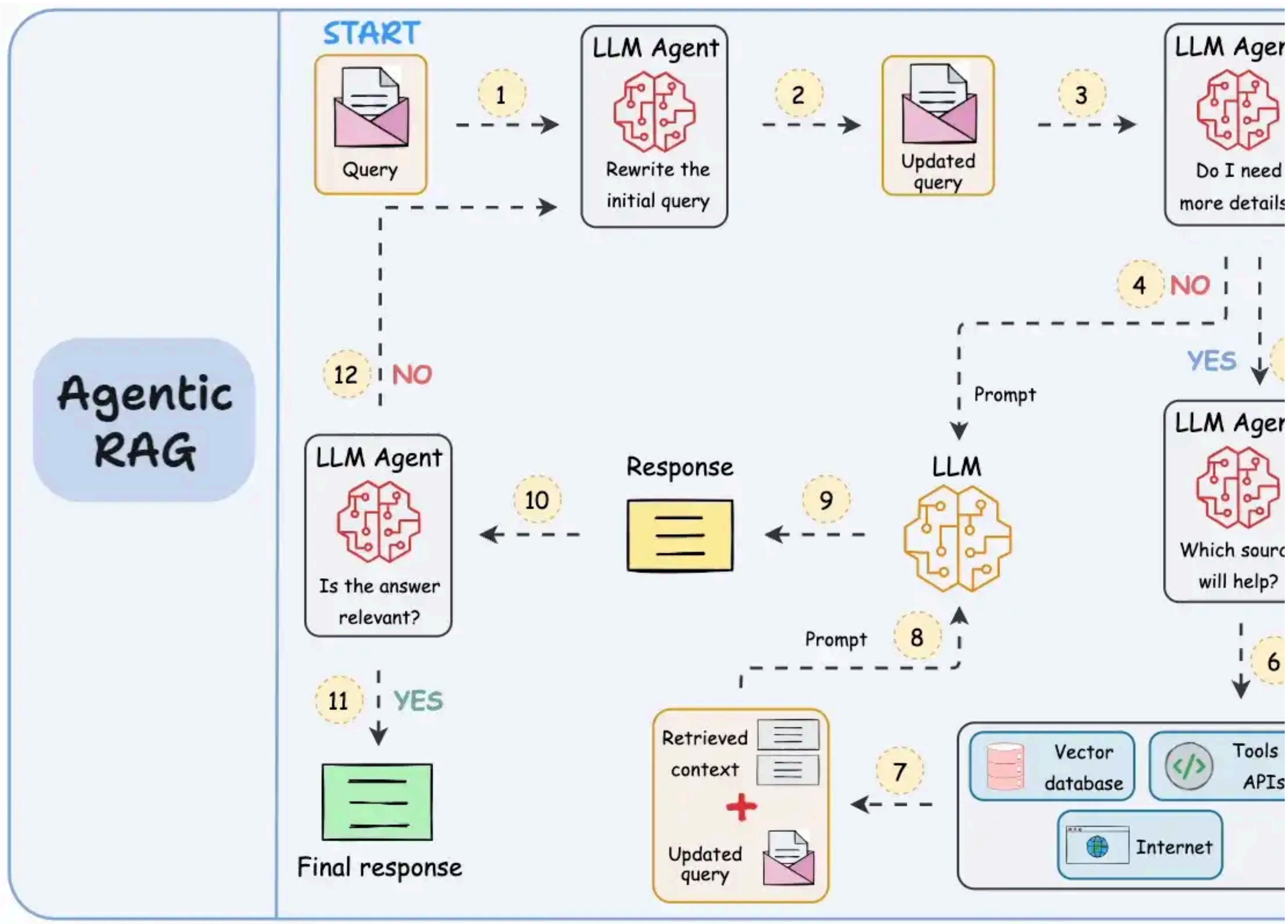
Step 9) Either of the above two paths produces a response.

Step 10) A final agent checks if the answer is relevant to the query and context.

- Step 11) If yes, return the response.
- Step 12) If not, go back to Step 1. This procedure continues for a few iterati until the system admits it cannot answer the query.

This makes the RAG much more robust since, at every step, agentic behavior ensures that individual outcomes are aligned with the final goal.

That said, it is also important to note that building RAG systems typically boils down to design preferences/choices.

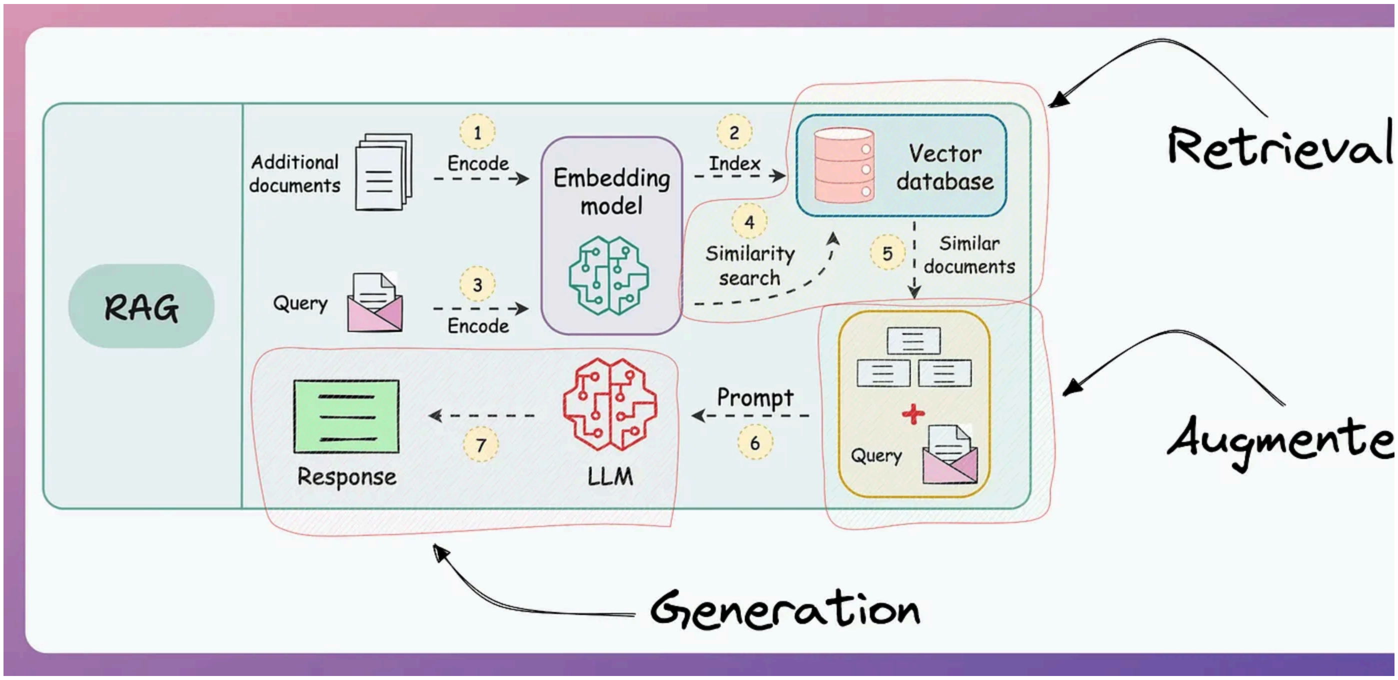


The diagram above is one of many blueprints that an agentic RAG system may possess. You can adapt it according to your specific use case.

Going ahead, we shall cover RAG-focused agentic workflows in our ongoing R crash course in much more detail.

Why care about RAG?

RAG is a key NLP system that got massive attention due to one of the key challenges it solved around LLMs.



More specifically, if you know how to build a reliable RAG system, you can byp the challenge and cost of fine-tuning LLMs.

That's a considerable cost saving for enterprises.

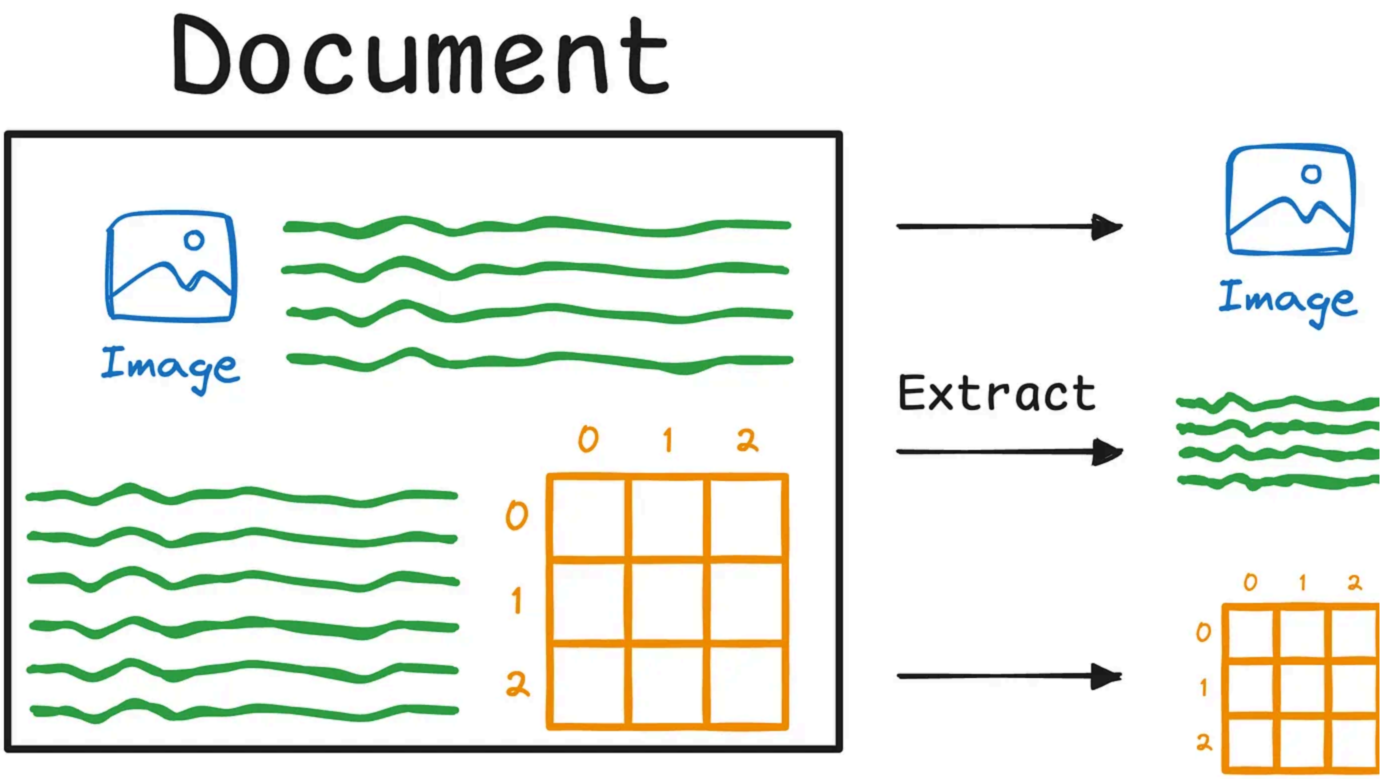
And at the end of the day, all businesses care about *impact*. That's it!

- Can you reduce costs?
- Drive revenue?
- Can you scale ML models?
- Predict trends before they happen?

Thus, the objective of this crash course is to help you **implement** reliable RAG systems, understand the underlying challenges, and develop expertise in buildi RAG apps on LLMs, which every industry cares about now.

- In Part 1, we explored the foundational components of RAG systems, the typical RAG workflow, and the tool stack, and also learned the implementation.

- In Part 2, we understood how to evaluate RAG systems (with implementati
- In Part 3, we learned techniques to optimize RAG systems and handle millions/billions of vectors (with implementation).
- In Part 4, we understood multimodality and covered techniques to build R/ systems on complex docs—ones that have images, tables, and texts (with implementation):



- In Part 5, we understood the fundamental building blocks of multimodal R systems that will help us improve what we built in Part 4.
- In Part 6, we utilized the learnings from Part 5 to build a more extensive ar capable multimodal RAG system.

Of course, if you have never worked with LLMs, that's okay. We cover everythir in a practical and beginner-friendly way.

👉 Over to you: What does your system design look like for Agentic RAG?

Thanks for reading!

← Previous Next

Discussion about this post

Comments Restacks

Write a comment...