llama index

LlamaIndex is a framework for building **context-augmented** [LLM](https://en.wikipedia.org/wiki/Large_language_model) applications.

Context augmentation refers to any use case that applies LLMs on top of your private or domain-specific data.

LlamaIndex provides tooling to enable context augmentation. A popular example is [Retrieval-Augmented Generation (RAG)](https://docs.llamaindex.ai/en/stable/getting_started/concepts/) which combines context with LLMs at inference time. Another is [finetuning](https://docs.llamaindex.ai/en/stable/use_cases/fine_tuning/).

LlamaIndex provides the following tools to help you quickly standup production-ready LLM applications:

* **Data connectors** ingest your existing data from their native source and format. These could be APIs, PDFs, SQL, and (much) more.
* **Data indexes** structure your data in intermediate representations that are easy and performant for LLMs to consume.
* **Engines** provide natural language access to your data. For example:
  + Query engines are powerful interfaces for question-answering (e.g. a RAG pipeline).
  + Chat engines are conversational interfaces for multi-message, "back and forth" interactions with your data.
* **Agents** are LLM-powered knowledge workers augmented by tools, from simple helper functions to API integrations and more.
* **Observability/Evaluation** integrations that enable you to rigorously experiment, evaluate, and monitor your app in a virtuous cycle.

[**LlamaParse**](https://docs.llamaindex.ai/en/stable/llama_cloud/llama_parse/) **:** LlamaParse is the world's first genAI-native document parsing platform - built with LLMs and for LLM use cases.   
It comes equipped with the following features:

* State-of-the-art table extraction
* Provide *natural language instructions* to parse the output in the exact format you want it.
* JSON mode
* Image extraction
* Support for 10+ file types (.pdf, .pptx, .docx, .html, .xml, and more)
* Foreign language support

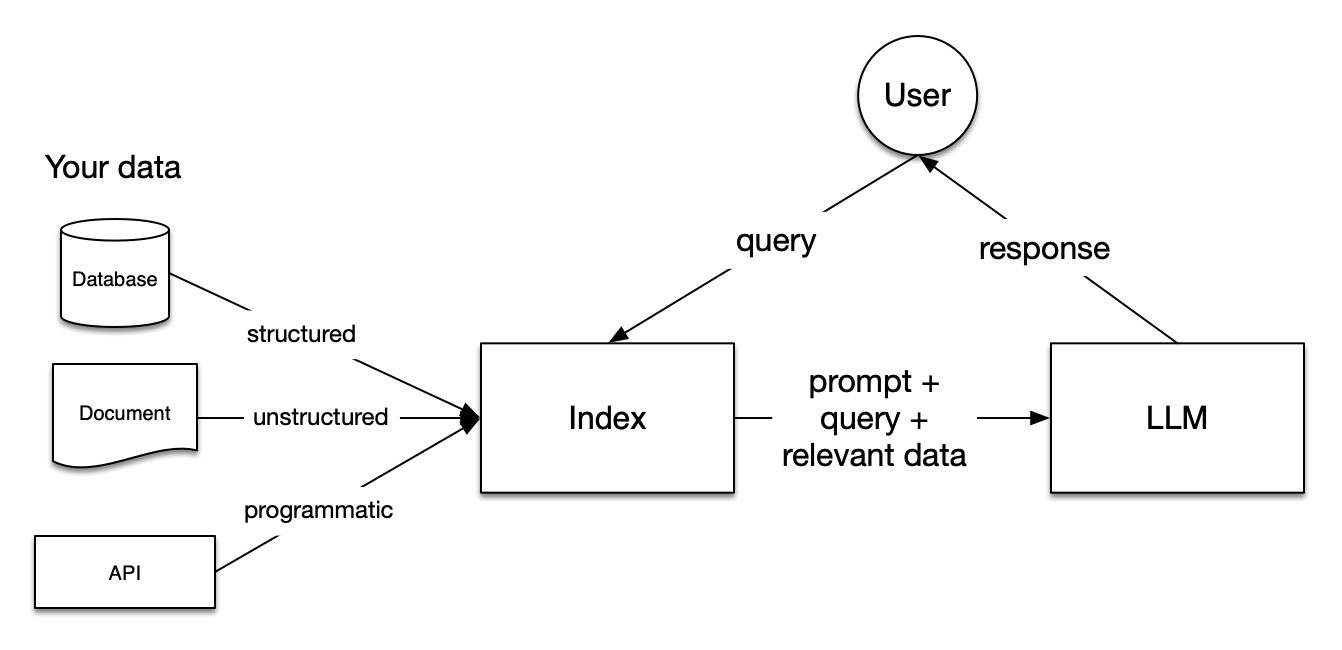
Link : <https://docs.llamaindex.ai/en/stable/llama_cloud/llama_parse/>

[LlamaHub](https://llamahub.ai/) | A large (and growing!) collection of custom data connectors.

## Retrieval Augmented Generation (RAG):

LLMs are trained on enormous bodies of data but they aren't trained on **your** data. Retrieval-Augmented Generation (RAG) solves this problem by adding your data to the data LLMs already have access to.

In RAG, your data is loaded and prepared for queries or "indexed". User queries act on the index, which filters your data down to the most relevant context. This context and your query then go to the LLM along with a prompt, and the LLM provides a response.



## Stages within RAG[#](https://docs.llamaindex.ai/en/stable/getting_started/concepts/#stages-within-rag)

There are five key stages within RAG, which in turn will be a part of any larger application you build. These are:

* **Loading**: this refers to getting your data from where it lives -- whether it's text files, PDFs, another website, a database, or an API -- into your pipeline. [LlamaHub](https://llamahub.ai/) provides hundreds of connectors to choose from.
* **Indexing**: this means creating a data structure that allows for querying the data. For LLMs this nearly always means creating vector embeddings, numerical representations of the meaning of your data, as well as numerous other metadata strategies to make it easy to accurately find contextually relevant data.
* **Storing**: once your data is indexed you will almost always want to store your index, as well as other metadata, to avoid having to re-index it.
* **Querying**: for any given indexing strategy there are many ways you can utilize LLMs and LlamaIndex data structures to query, including sub-queries, multi-step queries and hybrid strategies.
* **Evaluation**: a critical step in any pipeline is checking how effective it is relative to other strategies, or when you make changes. Evaluation provides objective measures of how accurate, faithful and fast your responses to queries are.



## Important concepts within each step[#](https://docs.llamaindex.ai/en/stable/getting_started/concepts/#important-concepts-within-each-step)

There are also some terms you'll encounter that refer to steps within each of these stages.

### Loading stage[#](https://docs.llamaindex.ai/en/stable/getting_started/concepts/#loading-stage)

[**Nodes and Documents**](https://docs.llamaindex.ai/en/stable/module_guides/loading/documents_and_nodes/): A Document is a container around any data source - for instance, a PDF, an API output, or retrieve data from a database. A Node is the atomic unit of data in LlamaIndex and represents a "chunk" of a source Document. Nodes have metadata that relate them to the document they are in and to other nodes.

[**Connectors**](https://docs.llamaindex.ai/en/stable/module_guides/loading/connector/): A data connector (often called a Reader) ingests data from different data sources and data formats into Documents and Nodes.

### Indexing Stage[#](https://docs.llamaindex.ai/en/stable/getting_started/concepts/#indexing-stage)

[**Indexes**](https://docs.llamaindex.ai/en/stable/module_guides/indexing/): Once you've ingested your data, LlamaIndex will help you index the data into a structure that's easy to retrieve. This usually involves generating vector embeddings which are stored in a specialized database called a vector store. Indexes can also store a variety of metadata about your data.

[**Embeddings**](https://docs.llamaindex.ai/en/stable/module_guides/models/embeddings/) LLMs generate numerical representations of data called embeddings. When filtering your data for relevance, LlamaIndex will convert queries into embeddings, and your vector store will find data that is numerically similar to the embedding of your query.

### Querying Stage[#](https://docs.llamaindex.ai/en/stable/getting_started/concepts/#querying-stage)

[**Retrievers**](https://docs.llamaindex.ai/en/stable/module_guides/querying/retriever/): A retriever defines how to efficiently retrieve relevant context from an index when given a query. Your retrieval strategy is key to the relevancy of the data retrieved and the efficiency with which it's done.

[**Routers**](https://docs.llamaindex.ai/en/stable/module_guides/querying/router/): A router determines which retriever will be used to retrieve relevant context from the knowledge base. More specifically, the RouterRetriever class, is responsible for selecting one or multiple candidate retrievers to execute a query. They use a selector to choose the best option based on each candidate's metadata and the query.

[**Node Postprocessors**](https://docs.llamaindex.ai/en/stable/module_guides/querying/node_postprocessors/): A node postprocessor takes in a set of retrieved nodes and applies transformations, filtering, or re-ranking logic to them.

[**Response Synthesizers**](https://docs.llamaindex.ai/en/stable/module_guides/querying/response_synthesizers/): A response synthesizer generates a response from an LLM, using a user query and a given set of retrieved text chunks.

### Putting it all together[#](https://docs.llamaindex.ai/en/stable/getting_started/concepts/#putting-it-all-together)

There are endless use cases for data-backed LLM applications but they can be roughly grouped into three categories:

[**Query Engines**](https://docs.llamaindex.ai/en/stable/module_guides/deploying/query_engine/): A query engine is an end-to-end pipeline that allows you to ask questions over your data. It takes in a natural language query, and returns a response, along with reference context retrieved and passed to the LLM.

[**Chat Engines**](https://docs.llamaindex.ai/en/stable/module_guides/deploying/chat_engines/): A chat engine is an end-to-end pipeline for having a conversation with your data (multiple back-and-forth instead of a single question-and-answer).

[**Agents**](https://docs.llamaindex.ai/en/stable/module_guides/deploying/agents/): An agent is an automated decision-maker powered by an LLM that interacts with the world via a set of [tools](https://docs.llamaindex.ai/en/stable/module_guides/deploying/agents/tools/). Agents can take an arbitrary number of steps to complete a given task, dynamically deciding on the best course of action rather than following pre-determined steps. This gives it additional flexibility to tackle more complex tasks.