

Introduction#

What are agents?#

[Agents](#) are LLM-powered knowledge assistants that use tools to perform tasks like research, data extraction, and more. Agents range from simple question-answering to being able to sense, decide and take actions in order to complete tasks.

LlamaIndex provides a framework for building agents including the ability to use RAG pipelines as one of many tools to complete a task.

What are workflows?#

[Workflows](#) are multi-step processes that combine one or more agents, data connectors, and other tools to complete a task. They are event-driven software that allows you to combine RAG data sources and multiple agents to create a complex application that can perform a wide variety of tasks with reflection, error-correction, and other hallmarks of advanced LLM applications. You can then [deploy these agentic workflows](#) as production microservices.

What is context augmentation?#

LLMs offer a natural language interface between humans and data. LLMs come pre-trained on huge amounts of publicly available data, but they are not trained on **your** data. Your data may be private or specific to the problem you're trying to solve. It's behind APIs, in SQL databases, or trapped in PDFs and slide decks.

Context augmentation makes your data available to the LLM to solve the problem at hand. LlamaIndex provides the tools to build any of context-augmentation use case, from prototype to production. Our tools allow you to ingest, parse, index and process your data and quickly implement complex query workflows combining data access with LLM prompting.

The most popular example of context-augmentation is [Retrieval-Augmented Generation or RAG](#), which combines context with LLMs at inference time.

LlamaIndex is the framework for Context-Augmented LLM Applications#

LlamaIndex imposes no restriction on how you use LLMs. You can use LLMs as auto-complete, chatbots, agents, and more. It just makes using them easier. We provide tools like:


- **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 flow).
 - 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.
- **Workflows** allow you to combine all of the above into an event-driven system far more flexible than other, graph-based approaches.

Use cases#

Some popular use cases for LlamaIndex and context augmentation in general include:

- [Question-Answering](#) (Retrieval-Augmented Generation aka RAG)
- [Chatbots](#)
- [Document Understanding and Data Extraction](#)
- [Autonomous Agents](#) that can perform research and take actions
- [Multi-modal applications](#) that combine text, images, and other data types
- [Fine-tuning](#) models on data to improve performance

Check out our [use cases](#) documentation for more examples and links to tutorials.

 Who is LlamaIndex for?#

LlamaIndex provides tools for beginners, advanced users, and everyone in between.

Our high-level API allows beginner users to use LlamaIndex to ingest and query their data in 5 lines of code.

For more complex applications, our lower-level APIs allow advanced users to customize and extend any module -- data connectors, indices, retrievers, query engines, and reranking modules -- to fit their needs.

Getting Started#

LlamaIndex is available in Python (these docs) and [Typescript](#). If you're not sure where to start, we recommend reading [how to read these docs](#) which will point you to the right place based on your experience level.

30 second quickstart#

Set an environment variable called OPENAI_API_KEY with an [OpenAI API key](#). Install the Python library:

```
pip install llama-index
```

Put some documents in a folder called data, then ask questions about them with our famous 5-line starter:

```
from llama_index.core import VectorStoreIndex, SimpleDirectoryReader
```

```
documents = SimpleDirectoryReader("data").load_data()
```

```
index = VectorStoreIndex.from_documents(documents)
```

```
query_engine = index.as_query_engine()
```

```
response = query_engine.query("Some question about the data should go here")
```

```
print(response)
```

If any part of this trips you up, don't worry! Check out our more comprehensive starter tutorials using [remote APIs like OpenAI](#) or [any model that runs on your laptop](#).

LlamaCloud#

If you're an enterprise developer, check out [LlamaCloud](#). It is an end-to-end managed service for data parsing, ingestion, indexing, and retrieval, allowing you to get production-quality data for your production LLM application. It's available both hosted on our servers or as a self-hosted solution.

LlamaParse#

LlamaParse is our state-of-the-art document parsing solution. It's available as part of LlamaCloud and also available as a self-serve API. You can [sign up](#) and parse up to 1000 pages/day for free, or enter a credit card for unlimited parsing. [Learn more](#).

Community#

Need help? Have a feature suggestion? Join the LlamaIndex community:

- [Twitter](#)
- [Discord](#)
- [LinkedIn](#)

Getting the library#

- LlamaIndex Python
 - [LlamaIndex Python Github](#)
 - [Python Docs](#) (what you're reading now)
 - [LlamaIndex on PyPi](#)
- LlamaIndex.TS (Typescript/Javascript package):
 - [LlamaIndex.TS Github](#)
 - [TypeScript Docs](#)
 - [LlamaIndex.TS on npm](#)

Contributing#

We are open-source and always welcome contributions to the project! Check out our [contributing guide](#) for full details on how to extend the core library or add an integration to a third party like an LLM, a vector store, an agent tool and more.

LlamaIndex Ecosystem#

There's more to the LlamaIndex universe! Check out some of our other projects:

- [llama_deploy](#) | Deploy your agentic workflows as production microservices
- [LlamaHub](#) | A large (and growing!) collection of custom data connectors
- [SEC Insights](#) | A LlamaIndex-powered application for financial research
- [create-llama](#) | A CLI tool to quickly scaffold LlamaIndex projects

High-Level Concepts#

This is a quick guide to the high-level concepts you'll encounter frequently when building LLM applications.

Large Language Models (LLMs)#

LLMs are the fundamental innovation that launched LlamaIndex. They are an artificial intelligence (AI) computer system that can understand, generate, and manipulate natural language, including answering questions based on their training data or data provided to them at query time. You can [learn more about using LLMs](#).

Agentic Applications#

When an LLM is used within an application, it is often used to make decisions, take actions, and/or interact with the world. This is the core definition of an **agentic application**.

While the definition of an agentic application is broad, there are several key characteristics that define an agentic application:

- **LLM Augmentation:** The LLM is augmented with tools (i.e. arbitrary callable functions in code), memory, and/or dynamic prompts.
- **Prompt Chaining:** Several LLM calls are used that build on each other, with the output of one LLM call being used as the input to the next.
- **Routing:** The LLM is used to route the application to the next appropriate step or state in the application.
- **Parallelism:** The application can perform multiple steps or actions in parallel.
- **Orchestration:** A hierarchical structure of LLMs is used to orchestrate lower-level actions and LLMs.
- **Reflection:** The LLM is used to reflect and validate outputs of previous steps or LLM calls, which can be used to guide the application to the next appropriate step or state.

In LlamaIndex, you can build agentic applications by using the Workflow class to orchestrate a sequence of steps and LLMs. You can [learn more about workflows](#).

Agents#

We define an agent as a specific instance of an "agentic application". An agent is a piece of software that semi-autonomously performs tasks by combining LLMs with other tools and memory, orchestrated in a reasoning loop that decides which tool to use next (if any).

What this means in practice, is something like: - An agent receives a user message - The agent uses an LLM to determine the next appropriate action to take using the previous chat history, tools, and the latest user message - The agent may invoke one or more tools to assist in the users request - If tools are used, the agent will then interpret the tool outputs and use them to inform the next action - Once the agent stops taking actions, it returns the final output to the user

You can [learn more about agents](#).

Retrieval Augmented Generation (RAG)#

Retrieval-Augmented Generation (RAG) is a core technique for building data-backed LLM applications with LlamaIndex. It allows LLMs to answer questions about your private data by providing it to the LLM at query time, rather than training the LLM on your data. To avoid sending **all** of your data to the LLM every time, RAG indexes your data and selectively sends only the relevant parts along with your query. You can [learn more about RAG](#).

Use cases#

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

Agents: An agent is an automated decision-maker powered by an LLM that interacts with the world via a set of [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.

Workflows: A Workflow in LlamaIndex is a specific event-driven abstraction that allows you to orchestrate a sequence of steps and LLMs calls. Workflows can be used to implement any agentic application, and are a core component of LlamaIndex.

Structured Data Extraction Pydantic extractors allow you to specify a precise data structure to extract from your data and use LLMs to fill in the missing pieces in a type-safe way. This is useful for extracting structured data from unstructured sources like PDFs, websites, and more, and is key to automating workflows.

Query Engines: A query engine is an end-to-end flow 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: A chat engine is an end-to-end flow for having a conversation with your data (multiple back-and-forth instead of a single question-and-answer).

Installation and Setup#

The LlamaIndex ecosystem is structured using a collection of namespaced python packages.

What this means for users is that pip install llama-index comes with a core starter bundle of packages, and additional integrations can be installed as needed.

A complete list of packages and available integrations is available on [LlamaHub](#).

Quickstart Installation from Pip#

To get started quickly, you can install with:

```
pip install llama-index
```

This is a starter bundle of packages, containing

- llama-index-core
- llama-index-llms-openai
- llama-index-embeddings-openai
- llama-index-program-openai
- llama-index-question-gen-openai
- llama-index-agent-openai
- llama-index-readers-file
- llama-index-multi-modal-llms-openai

NOTE: LlamaIndex may download and store local files for various packages (NLTK, HuggingFace, ...). Use the environment variable "LLAMA_INDEX_CACHE_DIR" to control where these files are saved.

Important: OpenAI Environment Setup#

By default, we use the OpenAI gpt-3.5-turbo model for text generation and text-embedding-ada-002 for retrieval and embeddings. In order to use this, you must have an OPENAI_API_KEY set up as an environment variable. You can obtain an API key by logging into your OpenAI account and [creating a new API key](#).

Tip

You can also [use one of many other available LLMs](#). You may need additional environment keys + tokens setup depending on the LLM provider.

[Check out our OpenAI Starter Example](#)

Custom Installation from Pip#

If you aren't using OpenAI, or want a more selective installation, you can install individual packages as needed.

For example, for a local setup with Ollama and HuggingFace embeddings, the installation might look like:

```
pip install llama-index-core llama-index-readers-file llama-index-llms-ollama llama-index-embeddings-huggingface
```

[Check out our Starter Example with Local Models](#)

A full guide to using and configuring LLMs is available [here](#).

A full guide to using and configuring embedding models is available [here](#).

Installation from Source#

Git clone this repository: `git clone https://github.com/run-llama/llama_index.git`. Then do the following:

- [Install poetry](#) - this will help you manage package dependencies
- If you need to run shell commands using Poetry but the shell plugin is not installed, add the plugin by running:
 - `poetry self add poetry-plugin-shell`
- `poetry shell` - this command creates a virtual environment, which keeps installed packages contained to this project
- `pip install -e llama-index-core` - this will install the core package
- (Optional) `poetry install --with dev,docs` - this will install all dependencies needed for most local development

From there, you can install integrations as needed with pip, For example:

```
pip install -e llama-index-integrations/readers/llama-index-readers-file
```

```
pip install -e llama-index-integrations/lms/llama-index-lms-ollama
```

How to read these docs#

Welcome to the LlamaIndex documentation! We've tried hard to make these docs approachable regardless of your experience level with LlamaIndex and with LLMs and generative AI in general.

Before you start#

LlamaIndex is a Python library, so you should have Python [installed](#) and a basic working understanding of how to write it. If you prefer JavaScript, we recommend trying out our [TypeScript package](#).

Many of our examples are formatted as Notebooks, by which we mean Jupyter-style notebooks. You don't have to have Jupyter installed; you can try out most of our examples on a hosted service like [Google Colab](#).

Structure of these docs#

Our docs are structured so you should be able to roughly progress simply by moving across the links at the top of the page from left to right, or just hitting the "next" link at the bottom of each page.

1. **Getting started:** The section you're in right now. We can get you going from knowing nothing about LlamaIndex and LLMs. [Install the library](#), write your first demo in [five lines of code](#), learn more about the [high level concepts](#) of LLM applications, and then see how you can [customize the five-line example](#) to meet your needs.
2. **Learn:** Once you've completed the Getting Started section, this is the next place to go. In a series of bite-sized tutorials, we'll walk you through every stage of building a production LlamaIndex application and help you level up on the concepts of the library and LLMs in general as you go.
3. **Use cases:** If you're a dev trying to figure out whether LlamaIndex will work for your use case, we have an overview of the types of things you can build.
4. **Examples:** We have rich notebook examples for nearly every feature under the sun. Explore these to find and learn something new about LlamaIndex.
5. **Component guides:** Arranged in the same order of building an LLM application as our Learn section, these are comprehensive, lower-level guides to the individual components of LlamaIndex and how to use them.
6. **Advanced Topics:** Already got a working LlamaIndex application and looking to further refine it? Our advanced section will walk you through the [first things you should try optimizing](#) like your embedding model and chunk size through progressively more complex and subtle customizations all the way to [fine tuning](#) your model.