# LlamaIndex (Beginner's Guide – 2025 Edition)

---

## □ What is LlamaIndex?

**LlamaIndex** is a Python framework that helps you **connect large language models (LLMs)** like GPT, LLaMA 3 (via Ollama), Claude, etc., to your **own data** (PDFs, files, APIs, databases, etc.).

It is designed for **RAG** — *Retrieval Augmented Generation* — where the LLM can "read" your documents before answering.

---

## □ Why Use LlamaIndex?

| Without LlamaIndex | With LlamaIndex |
|---|---|
| LLM doesn't know your files | LLM can answer from your PDFs, notes |
| Limited context window | Smart retrieval from vector index |
| No memory or long chats | Supports chat engines and memory |
| No file search or summarization | Built-in file indexing and QA |

---

## □ Core Concepts in LlamaIndex

Here are the **basic building blocks** of LlamaIndex:

### 1. Document

- Raw text data (e.g., from PDF, web, CSV, DB)
- Created via a **Loader**

```
from llama_index.core import SimpleDirectoryReader

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

---

### 2. Node

- A chunk of a document
- LlamaIndex splits documents into **small retrievable chunks** (nodes)
- This helps with accurate responses from LLMs

---

### 3. Index

- A searchable structure built from your documents/nodes
- Most common: `VectorStoreIndex` (stores embeddings for similarity search)

```
from llama_index.core import VectorStoreIndex

index = VectorStoreIndex.from_documents(documents)
```

---

### 4. LLM

- The language model used to generate responses
- LlamaIndex can work with OpenAI, Ollama, HuggingFace, etc.

```
from llama_index.llms.ollama import Ollama

llm = Ollama(model="llama3")
```

---

### 5. Query Engine

- Used to ask questions over your indexed data

```
query_engine = index.as_query_engine(llm=llm)
response = query_engine.query("What is this PDF about?")
```

---

### 6. Chat Engine (Optional)

- Adds **memory** and **conversation flow**
- Supports follow-up questions

```
chat_engine = index.as_chat_engine(chat_mode="condense_question")
response = chat_engine.chat("Tell me about section 2")
```

---

### 7. Memory (Optional)

- Stores chat history to maintain context

- E.g., `ChatMemoryBuffer`

---

## □ Basic Workflow

Here's the typical flow of building an LLM app with LlamaIndex:

```
[ PDF / text / CSV ]
        ↓
[ Document Loader ]
        ↓
[ Nodes + Indexing ]
        ↓
[ VectorStoreIndex ]
        ↓
[ Query Engine or Chat Engine ]
        ↓
[ Answers from LLM (via Ollama, OpenAI, etc.) ]
```

---

## □ Basic Code Example (Chat over PDF)

```
from llama_index.core import SimpleDirectoryReader, VectorStoreIndex
from llama_index.llms.ollama import Ollama

# Load your data (PDF, txt, etc.)
documents = SimpleDirectoryReader(input_dir="./data").load_data()

# Load local LLM (LLaMA 3)
llm = Ollama(model="llama3")

# Build index
index = VectorStoreIndex.from_documents(documents)

# Ask questions
query_engine = index.as_query_engine(llm=llm)
response = query_engine.query("What is this document about?")
print(response)
```

---

## □ Common Loaders (Input Types)

LlamaIndex supports many data sources:

| Source | Loader |
|---|---|
| PDFs | `SimpleDirectoryReader` (via PyMuPDF) |
| Markdown | `SimpleDirectoryReader` |
| Notion | `NotionPageReader` |
| Web pages | `BeautifulSoupWebReader` |
| CSV | `PandasCSVReader` |
| SQL/DB | `SQLDatabaseReader` |
| APIs | Custom via `Document` object |

---

## □ Advanced Features (For Later)

| Feature | Description |
|---|---|
| RAG Pipelines | Retrieval + custom prompts |
| Memory | Keeps multi-turn context |
| Agents | Tool-using agents (math, API calls) |
| Streaming | Token-by-token output from LLM |
| Knowledge Graphs | Connect concepts from your docs |
| Custom Tools | Plug in functions for the agent |

---

## □ Best LLMs to Use with LlamaIndex

| LLM | Via | Best For |
|---|---|---|
| `llama3` | Ollama | Local + fast |
| `mistral` | Ollama | Lightweight |
| `gpt-4` | OpenAI | Best quality |
| `claude` | Anthropic | Long context |

---

## □ Learning Resources

- □ LlamaIndex Docs
- □ Ollama
- □ LangChain vs LlamaIndex Comparison

---

## □ Summary

| Topic | Key Points |
|---|---|
| **What** | LlamaIndex connects LLMs to your data |
| **Why** | For private, powerful Q&A, search, chat |
| **How** | Load → Index → Ask Questions |
| **Tools** | Works with Ollama (LLaMA 3), PDFs, APIs |
| **Good For** | PDF bots, internal search, custom chatbots |

Creating a **chat over your PDF using LlamaIndex** with **LLaMA 3 via Ollama** is a great beginner-to-intermediate project. Here's a complete step-by-step guide to help you build it from scratch using **LlamaIndex**, **PyMuPDF**, and **Ollama (LLaMA 3)**.

---

# □ What You'll Build

An offline, private AI chatbot that can **answer questions based on the contents of a PDF** (e.g. research paper, eBook, invoice) using:

- □ **LlamaIndex** — for indexing + querying
- □ **PyMuPDF** — for reading PDFs
- □ **LLaMA 3 via Ollama** — as the language model backend

---

# □ Step 1: Install Required Packages

Install the necessary libraries:

```
pip install llama-index llama-index-readers-file llama-index-llms-ollama
pymupdf
```

□ What these do:

- `llama-index` – the main framework
- `llama-index-readers-file` – loads and parses local files (like PDFs)
- `llama-index-llms-ollama` – connects LlamaIndex to Ollama
- `pymupdf` – helps extract text from PDFs

---

# □ Step 2: Prepare Your PDF

Make sure your PDF is ready and located in your project folder. For example:

```
my_project/
├── chat_pdf.py
└── example.pdf
```

---

# □ Step 3: Full Python Script — `chat_pdf.py`

```
# chat_pdf.py
```

```
from llama_index.core import VectorStoreIndex, SimpleDirectoryReader
from llama_index.llms.ollama import Ollama
import os

# Step 1: Load your PDF file
documents = SimpleDirectoryReader(input_dir="./",
required_exts=[".pdf"]).load_data()

# Step 2: Load LLaMA 3 model from Ollama
llm = Ollama(model="llama3")  # Make sure llama3 is pulled via `ollama pull
llama3`

# Step 3: Build the vector index from the PDF
index = VectorStoreIndex.from_documents(documents)

# Step 4: Create a query engine
query_engine = index.as_query_engine(llm=llm, verbose=True)

# Step 5: Chat loop
print("□ Chat over your PDF (type 'exit' to quit)\n")

while True:
    query = input("You: ")
    if query.lower() in ["exit", "quit"]:
        break
    response = query_engine.query(query)
    print(f"□ AI: {response}\n")
```

---

# ▶□ Step 4: Run Your App

Make sure you've pulled the model:

```
ollama pull llama3
```

Then run the script:

```
python chat_pdf.py
```

Ask questions like:

- "What is the main topic of the document?"
- "Summarize the second chapter."
- "What conclusions does it make?"

---

# □ Notes

| Feature | How it Works |
|---|---|
| PDF Parsing | Uses `SimpleDirectoryReader` with `pymupdf` |
| Indexing | `VectorStoreIndex` creates a searchable index |
| LLM | Uses `Ollama` to run `llama3` locally |
| Querying | You ask a question → LLM retrieves relevant context and responds |

Great! Let's build a **Gradio-based PDF chatbot** using:

- □ **LlamaIndex** for indexing and querying PDF content
- □ **LLaMA 3 via Ollama** as the local LLM
- □ PDF **upload feature**
- □ **Chat interface** powered by **Gradio**

---

# □ Features of ThisApp

- Upload **any PDF**
- Ask **questions about its content**
- Chat via **Gradio UI**
- Uses **Ollama + llama3** (runs offline)
- Fully local, **no API key needed**

---

# □ Step 1: Install Required Packages

Install everything needed:

```
pip install llama-index llama-index-readers-file llama-index-llms-ollama
pymupdf gradio
```

And make sure the model is pulled:

```
ollama pull llama3
```

---

# □ Step 2: Full Code — `pdf_chat_gradio.py`

```
# pdf_chat_gradio.py

import gradio as gr
from llama_index.core import VectorStoreIndex, SimpleDirectoryReader,
ServiceContext
from llama_index.llms.ollama import Ollama
from llama_index.core.memory import ChatMemoryBuffer
from llama_index.core.chat_engine import CondenseQuestionChatEngine
import os
import shutil

# Temporary folder for uploaded PDFs
UPLOAD_DIR = "uploaded_pdfs"
os.makedirs(UPLOAD_DIR, exist_ok=True)

# Global variables for chat engine and state
chat_engine = None
chat_history = []

# Function to load PDF, create index and chat engine
def setup_chat_engine(pdf_file):
    global chat_engine, chat_history
    chat_history.clear()

    # Save uploaded PDF
    pdf_path = os.path.join(UPLOAD_DIR, pdf_file.name)
    shutil.copy(pdf_file.name, pdf_path)

    # Load PDF content
    documents = SimpleDirectoryReader(input_dir=UPLOAD_DIR,
required_exts=[".pdf"]).load_data()

    # Set up LLM via Ollama
    llm = Ollama(model="llama3")

    # Create a vector index
    index = VectorStoreIndex.from_documents(documents,
service_context=ServiceContext.from_defaults(llm=llm))

    # Create a memory buffer and chat engine
    memory = ChatMemoryBuffer.from_defaults(token_limit=2000)
    chat_engine = index.as_chat_engine(chat_mode="condense_question",
memory=memory)

    return "□  PDF uploaded and indexed. Ask your questions!"

# Chat function
def chat_with_pdf(user_input):
    global chat_engine, chat_history
    if chat_engine is None:
        return "□  Please upload a PDF first.", chat_history

    response = chat_engine.chat(user_input)
    chat_history.append((user_input, str(response)))
    return str(response), chat_history

# Gradio UI setup
```

```
with gr.Blocks() as demo:
    gr.Markdown("# □ Chat with Your PDF (LLaMA 3 via Ollama)")

    with gr.Row():
        pdf_input = gr.File(label="Upload a PDF", file_types=[".pdf"])
        upload_button = gr.Button("Process PDF")

    status_output = gr.Textbox(label="Status")

    chatbot = gr.Chatbot(label="Chat with PDF")
    question = gr.Textbox(label="Your Question")
    send_button = gr.Button("Send")

    # Handle upload
    upload_button.click(fn=setup_chat_engine, inputs=pdf_input,
outputs=status_output)

    # Handle chat
    send_button.click(fn=chat_with_pdf, inputs=question, outputs=[chatbot,
chatbot])

# Run the app
if __name__ == "__main__":
    demo.launch()
```

---

# ▶□ How to Run

```
python pdf_chat_gradio.py
```

Then open the link Gradio gives you, e.g.:

```
Running on local URL: http://127.0.0.1:7860
```

---

# □ UI Preview

**Upload PDF Ask Questions Chat UI**
✔□ Button ✔□ Text box ✔□ Chatbot

---

# □ What's Happening

| Step | Action |
|---|---|
| 1. Upload | PDF gets saved locally |
| 2. Index | LlamaIndex creates a vector store |
| 3. Chat | You ask questions, engine retrieves & answers |
| 4. Memory | Keeps chat context across questions |

---

## □ Bonus Tips

- Use small/medium PDFs (under 100 pages) for faster indexing.
- LLaMA 3 via Ollama runs locally, so this is **100% private**.
- Extend it to support **multiple files**, **PDF summarization**, or even **file Q&A over folders**.

Here's a clean and minimal example of a **question-answering AI agent** using:

- □ **Ollama** to run **LLaMA 3** locally
- □ No external APIs (runs fully offline)
- □ Handles general Q&A using natural language input

---

## □ What This Does

You'll be able to ask **any question**, and the agent will respond using **LLaMA 3** locally.

---

## □ Requirements

Install the necessary package:

```
pip install llama-index llama-index-llms-ollama
```

Pull the model if you haven't already:

```
ollama pull llama3
```

---

## □ Full Code: `qa_agent_ollama.py`

```
# qa_agent_ollama.py

from llama_index.llms.ollama import Ollama
from llama_index.core.query_engine import CustomQueryEngine
from llama_index.core.schema import QueryBundle
from llama_index.core.base.llms.base import ChatMessage

# Step 1: Load the local LLaMA 3 model
llm = Ollama(model="llama3", temperature=0.7)

# Step 2: System prompt to guide the agent's behavior
SYSTEM_PROMPT = (
    "You are a helpful and knowledgeable assistant. "
    "Answer user questions accurately and clearly."
)

# Step 3: Define a basic Query Engine to handle the Q&A loop
class QAQueryEngine(CustomQueryEngine):
    def __init__(self, llm):
        self._llm = llm

    def query(self, query_bundle: QueryBundle):
        messages = [
            ChatMessage(role="system", content=SYSTEM_PROMPT),
            ChatMessage(role="user", content=query_bundle.query_str)
        ]
        response = self._llm.chat(messages)
        return response

# Step 4: Initialize the agent
query_engine = QAQueryEngine(llm)

# Step 5: Interactive chat loop
print("□ Question Answering Agent (LLaMA 3 via Ollama)")
print("Type 'exit' to quit.\n")

while True:
    question = input("You: ")
    if question.lower() in ["exit", "quit"]:
        break
    response = query_engine.query(QueryBundle(question))
    print(f"□ Answer: {response}\n")
```

---

## ▶□ Example Interaction

```
You: What is the capital of Japan?
□ Answer: The capital of Japan is Tokyo.

You: Who wrote Hamlet?
□ Answer: Hamlet was written by William Shakespeare.
```

---

## □ Summary

| Feature | Included |
|---|---|
| LLaMA 3 (Ollama) | □ |
| Local-only | □ |
| General Q&A | □ |
| No RAG / PDFs | □ (can beadded) |
| No memory | □ (can be added) |

Creating a **simple AI math agent using LlamaIndex** and **LLaMA 3 (via Ollama)** is totally possible — although LlamaIndex is primarily built for document retrieval (RAG), not mathematical reasoning. But we can combine them smartly to make a minimal math agent.

---

## □ What You'll Build

A simple math assistant that:

- Accepts natural language math questions
- Uses **LLaMA 3 via Ollama** to solve them
- Uses **LlamaIndex** to structure a minimal interface (optional, mostly for standardization)

---

## □ Note on LlamaIndex and Math

LlamaIndex doesn't offer out-of-the-box math tools like LangChain does (e.g. `llm-math`), but if your model (like **LLaMA 3**) is good at reasoning, you can build a simple system that **parses the question and lets the model solve it** using natural language and in-context examples.

So, here's how to do it!

---

## □ Requirements

Install:

```
pip install llama-index llama-index-llms-ollama
```

Ensure LLaMA 3 is installed in Ollama:

```
ollama pull llama3
```

---

## □ Step-by-Step: Simple Math Agent Using LlamaIndex

### □ 1. Full Code: `math_agent_llamaindex.py`

```
# math_agent_llamaindex.py

from llama_index.llms.ollama import Ollama
from llama_index.core.query_engine import CustomQueryEngine
from llama_index.core.schema import QueryBundle
from llama_index.core.base.llms.base import ChatMessage


# Step 1: Load LLaMA 3 model
llm = Ollama(model="llama3", temperature=0.0)

# Step 2: Define simple prompt template for math
SYSTEM_PROMPT = (
    "You are a helpful math assistant. "
    "Your job is to solve math problems step-by-step and provide the final
answer clearly."
)

# Step 3: Custom QueryEngine that sends math question to LLaMA 3
class MathQueryEngine(CustomQueryEngine):
    def __init__(self, llm):
        self._llm = llm

    def query(self, query_bundle: QueryBundle):
        messages = [
            ChatMessage(role="system", content=SYSTEM_PROMPT),
            ChatMessage(role="user", content=query_bundle.query_str)
        ]
        response = self._llm.chat(messages)
        return response


# Step 4: Create query engine instance
query_engine = MathQueryEngine(llm)

# Step 5: Interactive prompt
```

```
print("□ Simple Math Agent (LLaMA 3 via LlamaIndex)\n")
while True:
    user_input = input("You: ")
    if user_input.lower() in ["exit", "quit"]:
        break
    response = query_engine.query(QueryBundle(user_input))
    print(f"□ Math Agent: {response}\n")
```

---

### □ Example Usage

```
You: What is 23 * 19?
□ Math Agent: 23 * 19 = 437

You: What is the square root of 625?
□ Math Agent: The square root of 625 is 25.

You: Add 12 to the previous result.
□ Math Agent: 25 + 12 = 37
```

If LLaMA 3 is reasoning well, it will solve these easily.

---

# □ How It Works

| Component | Role |
|---|---|
| `Ollama` | Connects to local LLaMA 3 model |
| `MathQueryEngine` | Sends messages directly to the model |
| `SYSTEM_PROMPT` | Guides the model to act like a math assistant |
| `LlamaIndex` | Used here for structure and chat interface |

# 10 Simple Project Ideas Using LlamaIndex

---

### 1. Chat with a PDF

 Build a chatbot that answers questions from a single PDF.

- Load a PDF using `SimpleDirectoryReader`
- Create a `VectorStoreIndex`
- Ask questions using `.query()`

**Bonus**: Add a Gradio UI

---

### 2. Multi-PDF Knowledge Bot

 Ask questions across multiple PDFs (e.g. multiple research papers or reports).

- Ingest a folder of PDFs
- Index them together
- Use `.as_chat_engine()` for conversation

---

### 3. Book Summarizer

 Upload a long book and get chapter-wise summaries.

- Use document loaders
- Chunk and identify chapters
- Summarize each section with `.query("summarize this")`

---

### 4. Private Google (Search Your Notes)

 Upload a folder of notes, articles, or blog posts and search them using natural language.

- Load `.txt` or `.md` files
- Use `VectorStoreIndex` for retrieval
- Add memory to refine or follow up

---

### 5. AI Resume Analyzer

 Upload a resume and ask questions like "Does this match a data science role?"

- Load `.pdf` resume
- Prompt the LLM to analyze strengths, gaps
- Output score or analysis

---

### 6. Syllabus or Policy Q&A Bot

 Upload a school syllabus or company policy PDF and answer questions like “What’s the deadline for project 2?”

- Great for onboarding bots
- Focus on QA from long policy documents

---

### 7. Q&A Over CSV

 Load a CSV file (e.g., sales data) and ask “What’s the top-selling product?”

- Use `PandasCSVReader`
- Index the rows
- Use natural language queries

---

### 8. Meeting Minutes Summarizer

 Upload a transcript of a meeting (text file), and summarize key points or action items.

- Load `.txt`
- Chunk the text
- Use `.query("Summarize action items")`

---

### 9. Personal Diary Analyzer

 Load diary entries and ask questions like “When did I feel most stressed?” or “Summarize last week emotionally.”

- Load `.txt` files

- Use time-based sorting + emotion prompts

---

### 10. AI Tutor for Science PDFs

 Upload a physics or chemistry chapter and ask follow-up questions like a tutor.

- Load a chapter PDF
- Use `.as_chat_engine()` for tutoring
- Add memory for multi-turn Q&A

---

# Bonus Ideas (Intermediate)

- Chatbot with long-term memory (`ChatMemoryBuffer`)
- RAG over Notion, Google Docs, or websites
- Webpage scraper + LLM Q&A bot
- Academic research assistant (loads multiple papers)

---

# Tools You Can Mix

| Add-on | Tool / Integration |
|---|---|
| UI | Gradio / Streamlit |
| Local LLM | Ollama (LLaMA 3, Mistral, etc.) |
| PDF Loader | PyMuPDF + `SimpleDirectoryReader` |
| Data | CSV, Markdown, Text, HTML |
| Memory | `ChatMemoryBuffer`, `SummaryMemory` |

---

# Tip to Get Started

Start with this:

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

```
from llama_index.llms.ollama import Ollama

docs = SimpleDirectoryReader("data").load_data()
llm = Ollama(model="llama3")
index = VectorStoreIndex.from_documents(docs)
response = index.as_query_engine(llm=llm).query("What is this about?")
print(response)
```