# 🧮 Module 0 — Math for AI Primer *(Optional)*

**Time budget:** 10 hours *(not included in core 60–80 hour total)*

---

## 💡 Who should take this?

Only go through this primer if you feel totally unfamiliar with terms like: **vector**, **matrix**, **dot product**, **probability distribution**, **standard deviation**, or **logarithm**.

This module is here to **refresh your comfort**, not turn you into a math expert. It helps you follow occasional math terms you'll see in later modules (e.g. in vector search, retrieval scoring, or model evaluation).

If you've worked with basic stats or linear algebra before, or you're comfortable learning these concepts on the fly — feel free to skip this and begin at Module 1.

### ✅ Quick Self-Assessment: Should You Do This Primer?

Answer **yes or no** to the following:

1. Do you feel unsure what a **vector** or **dot product** actually means in practice?
2. Have you never worked with **probability rules** like P(A ∩ B) or P(A ∪ B)?
3. Do you avoid terms like **standard deviation**, **mean**, or **distribution** in technical discussions?
4. Have you forgotten how **matrix multiplication** works (or why we do it)?
5. Would you struggle to explain what **cosine similarity** or **logarithmic scaling** refers to?

🟨 If you answered **"yes" to 2 or more**, consider doing the primer.

---

## 🎯 Learning goals:

- Refresh core math concepts relevant to LLMs and AI: vectors, matrices, probabilities, distributions.
- Build visual and intuitive understanding of how these ideas apply to search, similarity, and model evaluation.
- Practice basic math tasks that show up in embedding models and retrieval logic.

---

## 📚 Required resources (free unless noted):

1. **“Essence of Linear Algebra”**, video series, YouTube (3Blue1Brown), https://www.youtube.com/playlist?list=PLSQl0a2vh4HBtgXx6Z2H9pAGK9tY2Bu3k — *~3 hours*
   Visual intuitions for linear transformations, matrix multiplication, eigenvectors, and vector spaces.

2. **“Probability and Statistics”**, Khan Academy (selected lessons), https://www.khanacademy.org/math/statistics-probability — *~4 hours*
   Covers distributions, mean/median/variance, basic rules of probability, visualized through examples.

3. **“Precision vs Recall”**, video, StatQuest by Josh Starmer, https://www.youtube.com/watch?v=4z0lqjFnjsk — *~11 min*
   Applies core stats to model evaluation; shows how trade-offs work in classification.

---

## 🌱 Optional enrichment:

- **“Elements of AI”**, free online course, University of Helsinki, https://www.elementsofai.com/ — *~30 hours (use selectively)*
  Broad conceptual overview of AI history, ethics, and basics. Not math-heavy, but good for big-picture thinkers.

---

## 🛠️ Assignment:

Try one or more of the following:

- Manually compute **cosine similarity** between two small vectors (e.g. `[1, 0]` and `[0.5, 0.5]`)
- Calculate basic probabilities (e.g., P(A and B) vs P(A or B))
- Explain how a math concept like **logarithm**, **mean**, or **standard deviation** is used in AI (e.g., in loss functions or eval metrics)

Submit a short write-up or notebook showing your calculations and interpretation.

---

## 🤝 How to use the weekly sync: *(Optional)*

Join a call with peers who also completed this module. Share:

- Which concepts were new or tricky
- Any “aha!” moments from visual resources
- Math ideas you now see differently after applying them to AI

---

### 📝 Notes:

Don’t aim for perfection — this is a **confidence booster**, not a requirement.
If you're pressed for time, focus on the 3Blue1Brown videos and skip directly to Module 1 once you feel ready.

## Module 1 — LLM Fundamentals and Environment Setup

**Time budget:** 6 hours

---

### 🎯 Learning goals:

- Explain how Transformer-based LLMs work and why scale affects their capabilities
- Recognize key open-weight LLMs and how they differ from closed APIs
- Set up a Python environment, install libraries, and run an inference example
- Gain confidence running a basic model script as a foundation for future modules

---

### 📚 Required resources:

- “The Illustrated Transformer”, *article*, https://jalammar.github.io/illustrated-transformer/, [20 min]
  An intuitive, visual introduction to Transformer architecture. Explains self-attention and how Transformers enabled efficient parallel training and strong NLP capability.

- “Open-Source LLMs vs Closed: Unbiased Guide (2025)”, *article*, https://hatchworks.com/blog/gen-ai/open-source-vs-closed-llms-guide/ , [15 min]
  Practical comparison of open-weight models (e.g. TinyLLaMA, Phi-2) versus closed-API models, emphasizing control, customization, and deployment trade-offs.

- “Phi-2: The Surprising Power of Small Language Models”, *article*, https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/, [12 min]
  Introduces Phi-2 (2.7B parameters), demonstrating that small models can rival much larger ones when trained well.

- “6 Ways for Running a Local LLM (how to use Hugging Face)”, *article*, https://semaphore.io/blog/local-llm, [6 min]
  Guide to installing Transformers and running models locally via Hugging Face. Covers CPU vs GPU, tools like Llama.cpp, Ollama, LangChain.

- “TinyLLaMA 1.1B: Step-by-Step Guide on Google Colab”, *tutorial*, https://dev.to/_ken0x/tinyllama-llm-a-step-by-step-guide-to-implementing-the-11b-model-on-google-colab-1pjh, [15 min]
  Hands-on Colab tutorial walking through setup and inference with TinyLLaMA (1.1B), ideal for personal or cloud GPU usage.

---

## 🌱 Optional enrichment:

- “A Gentle Introduction to 8-bit Quantization”, *article*, https://huggingface.co/blog/hf-bitsandbytes-integration, [15 min]
  Explains quantization (e.g. 8-bit inference) to reduce VRAM usage for larger models on consumer GPUs.

- “LoRA: Low-Rank Adaptation for LLMs”, *conceptual guide*, https://huggingface.co/docs/peft/main/en/conceptual_guides/lora, [10 min]
  Overview of LoRA fine-tuning: adding small trainable layers to frozen models, enabling efficient customization.

- “Running LLMs on Colab and Kaggle”, *notebook repo*, https://github.com/casualcomputer/llm_google_colab#readme, [self-paced]
  Curated Jupyter notebooks for deploying open LLMs on free cloud environments like Colab and Kaggle.

---

## 🛠️ Assignment:

Install Python and `transformers` locally (with optional CUDA) or use Colab. Load and run a small open-weight model (e.g. TinyLLaMA-1.1B or Falcon 1B). Generate text using at least two different prompts and varied output lengths. Save sample outputs and notes on installation/setup challenges and model behavior.

---

## 🤝 How to use the weekly sync:

Learners share their setup (local vs Colab), chosen model, generated outputs, and any encountered issues (e.g. CUDA installation errors). Troubleshoot together and compare model runtime performance and output. Discuss how prompt design impacted results and reflect on surprises (e.g. CPU speed, quality differences).

---

### 📝 Notes

If you don’t have a GPU or run into installation errors locally, you can use **Google Colab** to run Hugging Face models with CPU or free GPU. See the “TinyLLaMA on Colab” tutorial for a full walkthrough.

## Module 2 — Prompt Engineering Techniques

**Time budget:** 6 hours

---

### 🎯 Learning goals:

- Define what prompts are and how they impact LLM output quality
- Apply various prompt strategies (zero-shot, few-shot, chain-of-thought) to improve clarity and accuracy
- Iterate on prompts based on output feedback and adjust tone/structure
- Evaluate prompt effectiveness and build intuition for prompt design

---

### 📚 Required resources:

**“Prompt engineering (best practices)”**, documentation
Link: https://huggingface.co/docs/transformers/en/tasks/prompting
Estimated time: 15 min
Official Hugging Face guide introducing zero-shot, few-shot, and chain-of-thought prompting using open-weight models. Demonstrates prompt design and input formatting.

**“Prompt Templates”**, documentation
Link: https://python.langchain.com/docs/concepts/prompt_templates/
Estimated time: 10 min
LangChain's core guide to building reusable prompt templates (string and chat-level) compatible with open-source Hugging Face LLMs.

**“Build a simple LLM app with prompt templates”**, tutorial (Jupyter/Colab)
Link: https://python.langchain.com/docs/tutorials/llm_chain/
Estimated time: 20 min
Hands-on notebook showing how to use `PromptTemplate` alongside chat models for translation tasks. Encourages prompt experimentation with open-weight LLMs.

**“Chain-of-Thought prompting explained”**, blog post
Link: https://magazine.sebastianraschka.com/p/understanding-reasoning-llms
Estimated time: 7 min

Demonstrates how adding reasoning-style cues like “think step by step” improves results on logic tasks. Examples use open-weight model settings.

**“How to prompt LLaMA 2 correctly”**, Hugging Face blog
Link: https://huggingface.co/blog/llama2#how-to-prompt-llama-2
Estimated time: 5 min
Details the `<s>[INST]…<<SYS>>…</SYS>…[/INST]` prompt format for LLaMA-2 models, showing how system instructions control behavior.

---

## 🌱 Optional enrichment:

**“Prompt-based methods (prompt tuning & soft prompts)”**, documentation
Link: https://huggingface.co/docs/peft/en/task_guides/prompt_based_methods
Estimated time: 12 min
Official Hugging Face PEFT guide explaining prompt tuning, prefix tuning, and soft prompts—advanced methods to adapt open models with minimal training.

**“Self-Consistency with Chain-of-Thought sampling”**, blog post
Link: https://medium.com/@johannes.koeppern/self-consistency-with-chain-of-thought-cot-sc-2f7a1ea9f941
Estimated time: 10 min
Discusses sampling multiple CoT outputs and voting—improving reliability of open LLMs' chain-of-thought reasoning.

**“Prompt injection attacks & defenses”**, LangChain blog
Link: https://blog.langchain.dev/prompt-injection-attacks-and-defenses/
Estimated time: 7 min
Covers security practices to prevent malicious or malformed prompts from compromising open-source LLM behavior.

---

## 🛠️ Assignment:

Use an open-weight LLM (e.g., TinyLLaMA, Phi-2 via Hugging Face) to design and iterate **two tasks** (e.g., summarization, sentiment classification). Submit:

1. **Side-by-side examples** of your prompt iterations (initial → refined) for each task.
2. A brief commentary (~300–500 words) explaining what changed, why you revised it, what improved (or didn’t), and your insights on prompt behavior.

---

## 🤝 How to use the weekly sync:

Pair with a peer and work on the **same task** (e.g., summarizing a short article). Compare:

- Differences in prompt structure and format
- Unexpected or surprising model outputs
- Which prompt performed best and why
  Optionally, demo your initial vs refined prompt iterations to the group and highlight lessons learned.

# Module 3 — Open-Source LLMs and Hugging Face Transformers

Time budget: 8 hours

---

## 🎯 Learning goals:

- Use Hugging Face Transformers to load and run different open-source LLMs (e.g. LLaMA 2, Falcon, GPT-J)
- Compare models in terms of size, latency, and output quality
- Use the Hugging Face Model Hub, tokenizers, and pipeline APIs for text generation
- Troubleshoot performance and compatibility issues during inference, including CPU vs GPU and quantization trade-offs

---

## 📚 Required resources:

1. "Llama 2 is here – get it on Hugging Face" , blog article, https://huggingface.co/blog/llama2 , estimated time: ~15 min
   Explains how to deploy LLaMA 2 using Transformers, pipeline API, and integration with bitsandbytes for quantized inference.

2. "Llama 2 Using Huggingface" by Luc Nguyen, article, https://medium.com/@lucnguyen_61589/llama-2-using-huggingface-part-1-3a29fdbaa9ed , ~10 min
   Step-by-step guide to loading the LLaMA 2 model in Python, measuring CPU/GPU inference latency, and showing practical quantization speed-ups.

3. "Comparing the Best Open-Source Large Language Models" (Shakudo blog), article, search online, ~10 min
   Gives side-by-side comparison of models like Falcon-7B vs Falcon-40B in terms of GPU requirements, speed, and quality for deployment.

4. "Model Quantization with Hugging Face Transformers and Bitsandbytes Integration" by Rakesh Rajpurohit, article, https://medium.com/@rakeshrajpurohit/model-quantization-with-hugging-face-transformers-and-bitsandbytes-integration-b4c9983e8996 , ~5 min
   Shows how to use `load_in_4bit=True` or `load_in_8bit=True` with

Transformers to reduce memory footprint and inference time.

5. “How to Quantize LLMs Using BitsandBytes” by Jack N., blog article, https://apxml.com/posts/efficient-llm-quantization-bitsandbytes , ~7 min
   Walks through 4-bit and 8-bit quantization via bitsandbytes, with code examples and hardware impact (VRAM/RAM savings).

---

## 🌱 Optional enrichment:

1. “vLLM: Easy, Fast, and Cheap LLM Serving with PagedAttention” , blog post, search online, ~10 min
   Describes a high-throughput inference engine that can achieve up to 24× faster batching than standard Transformers pipelines.

2. “Running LLMs on CPU with llama.cpp” , tutorial, search online, ~7 min
   Shows how to convert models to GGML/quantized format to run on CPU only (e.g. quantized 7B model loading under 4 GB RAM).

3. “Quantize Model with GPTQ, AWQ and Bitsandbytes” by Luv Bansal, article, https://towardsai.net/ , ~9 min
   Compares different quantization techniques (GPTQ, AWQ, bitsandbytes) and explains pros/cons when pushing quantized models to the Hugging Face Hub.

---

## 🛠️ Assignment:

Loading and Comparing Model Variants

- Choose an open-source model (~6–7B parameters) available on Hugging Face (e.g. LLaMA-2-7B, GPT-J-6B, Falcon-7B) that fits your hardware (CPU or Colab GPU).

- Load the model *without quantization* (default precision) using `AutoTokenizer`, `AutoModelForCausalLM`, or `pipeline`; run a text-generation prompt, measure latency, note memory usage and output quality.

- Load the *same model in quantized mode* (4-bit or 8-bit) using `load_in_4bit=True` or `load_in_8bit=True` (via transformers + bitsandbytes). Run the same prompt, measure latency & memory, compare output.

- Write a short report: compare speed improvement, memory reduction, and output quality differences. Note CPU vs GPU performance if applicable.

Estimated time: 3 h experimentation + 1 h documentation.

---

### 🤝 How to use the weekly sync:

**Model & Setup Show-and-Tell** (1 hour)

- Each learner shares the model they ran, hardware setup (CPU/RAM or GPU/VRAM), and whether they used quantization.
- Compare latency, memory usage, and output quality across setups.
- Discuss any troubleshooting experiences (e.g. out-of-memory errors, slow CPU runs), and surface solutions (e.g. switching model, using quantization, llama.cpp).
- Wrap up by reflecting on trade-offs: Is quantized output "good enough"? How much faster or more efficient was it? Which setup felt most practical for deployment?

---

### 📝 Notes

If you encounter out-of-memory (OOM) errors when loading larger models:

- Try running a **quantized version** (4-bit or 8-bit) using `bitsandbytes` as shown in the assignment.
- Use **Colab with GPU runtime** if your local machine lacks sufficient VRAM.
- For CPU-only environments, consider **llama.cpp** or models < 7B.

## Module 4 — Working with Embeddings and Vector Databases

**Time budget:** 8 hours

---

### 🎯 Learning goals:

- Explain what text embeddings are and how they capture semantic meaning.
- Generate embeddings using open-weight models like `all-MiniLM-L6-v2` and `bge-small-en`.
- Store and query embeddings in Chroma and Qdrant vector databases.
- Compare semantic similarity search to traditional keyword-based retrieval.

---

### 📚 Required resources:

**"What are Embeddings and Vector Databases?", article**
https://huggingface.co/blog/qdrddr/what-are-embeddings-and-vector-databases
(~25 min)

Conceptual overview of embeddings and how vector DBs enable semantic similarity search, with intuitive visuals and examples.

**“Chroma DB Tutorial: A Step-By-Step Guide”, tutorial**
https://www.datacamp.com/tutorial/chromadb-tutorial-step-by-step-guide
(~45 min)
Hands-on guide to installing ChromaDB, using an open embedding model (e.g. MiniLM), creating collections, adding documents with metadata, and running similarity queries.

**“A Developer’s Friendly Guide to Qdrant Vector Database”, tutorial**
https://www.cohorte.co/blog/a-developers-friendly-guide-to-qdrant-vector-database/
(~45 min)
Code-centric walkthrough: generate embeddings with Sentence Transformers, store them in Qdrant, run vector similarity search, and apply metadata filters.

**“Embedding Models You Can Use in Your Next Project for Free”, article**

https://python.plainenglish.io/embedding-models-you-can-use-in-your-next-project-for-free-7b9e244ac960
(~20 min)
Overview of performant, open-weight embedding models (MiniLM, BGE, E5, etc.) and guidance for selecting one without proprietary APIs.

**“Embeddings and Vector Databases With ChromaDB”, article**
https://realpython.com/chromadb-vector-database/
(~30 min)
RealPython explains semantic embedding workflows and how ChromaDB simplifies storage and retrieval, including metadata filtering and integration with Hugging Face models.

---

## 🌱 Optional enrichment:

**“INSTRUCTOR-XL: Instruction-Tuned Embeddings for Better Search”, model card**
https://huggingface.co/hkunlp/instructor-xl
(~20 min)
Advanced model that accepts an “instruction” with input to generate task-tuned embeddings; useful for domain-specific semantic retrieval.

**“What Is a Vector Database?”, article**
https://weaviate.io/blog/what-is-a-vector-database
(~30 min)
Introduction to Weaviate, another open-source vector DB with GraphQL API and hybrid search support.

**“Hybrid Search Explained”, article**
https://weaviate.io/blog/hybrid-search
(~20 min)

Discussion of combining dense (semantic) and sparse (keyword) search to improve retrieval accuracy — supported by Qdrant and Weaviate.

---

## 🛠️ Assignment:

**Build a local semantic search engine using embeddings and a vector database**

- Choose a small dataset (e.g. Markdown, plain text, or PDFs).
- Chunk documents and compute embeddings using either `all-MiniLM-L6-v2` or `bge-small-en`.
- Store the vectors and metadata in a local ChromaDB or Qdrant instance.
- Implement semantic search: embed a user query, retrieve the top matches, and display results.
- Optionally apply metadata filters or compare against simple keyword search results.
- (Optional) Use LangChain or LlamaIndex wrappers to streamline document loading, embedding, indexing, and querying.

---

## 🤝 How to use the weekly sync:

**Vector Store Show-and-Tell Session**
Each learner presents their implementation: model choice, vector store used (Chroma or Qdrant), indexing and metadata strategy, and sample semantic search query outputs. Compare setups and discuss query performance and filter handling. Highlight cases where semantic search succeeded but keyword search failed. Peer feedback helps with debugging and refining approach.

---

## 📝 Notes

To run **Chroma** or **Qdrant** locally:

- ChromaDB installs directly via pip (`pip install chromadb`), no server required.
- For Qdrant, you can run it via Docker (`docker run -p 6333:6333 qdrant/qdrant`) or use Qdrant Cloud.
  If setup proves difficult, start with Chroma — it's easier to prototype with.

# Module 5 — Building LLM Pipelines with LangChain and LlamaIndex

**Time budget:** 8 hours

## 🎯 Learning goals:

- Understand the roles of LangChain and LlamaIndex in pipeline design
- Use prompt templates, memory, and chaining features
- Ingest and index external text data (e.g. PDFs or Markdown) and query it with an LLM
- Combine components into working pipelines using real data

---

## 📚 Required resources:

1. **“Build a RAG Pipeline With the LLama Index”**, article, Analytics Vidhya, 10 min
   Link: https://www.analyticsvidhya.com/blog/2023/10/rag-pipeline-with-the-llama-index/
   A concise walkthrough using LlamaIndex to load documents, chunk text, generate embeddings, use a local vector DB, and perform Q&A via index querying.

2. **“RAG using Llama 2, Langchain and ChromaDB”**, notebook, Kaggle, ~15 min
   Link: https://www.kaggle.com/code/gpreda/rag-using-llama-2-langchain-and-chromadb
   Interactive notebook combining LangChain, Llama 2, and ChromaDB to build a local RAG Q&A pipeline over ingested documents.

3. **“Memory and Hybrid Search in RAG using LlamaIndex”**, article, Analytics Vidhya, ~6 min
   Link: https://www.analyticsvidhya.com/blog/2024/09/memory-and-hybrid-search-in-rag-using-llamaindex/
   Demonstrates how to enhance retrieval pipelines using LlamaIndex with Qdrant for hybrid search and persistent memory/chat context.

4. **“Build a Retrieval Augmented Generation (RAG) App”**, tutorial, LangChain documentation, ~20 min
   Link: https://python.langchain.com/en/latest/modules/chains/getting_started.html
   Official LangChain tutorial showing how to create a pipeline: document loader, text splitting, embeddings into a vector store, and LLM-based question answering.

5. **“RAG With Llama 3.1 8B, Ollama, and LangChain”**, article, DataCamp blog, ~12 min

Link: https://www.datacamp.com/blog/rag-llama3-1-8b-ollama-langchain
Practical example using Llama 3.1 locally via Ollama with LangChain for document ingestion, embedding, indexing, and querying.

6. **“Building RAG Pipelines using LlamaIndex | RAG and Agents Bootcamp”**, video (~60 min)
Link: https://www.youtube.com/watch?v=1g_zYi1GTI4
Recorded webinar showing how to build RAG pipelines using both LangChain and LlamaIndex with live coding and side-by-side comparison.

---

## 🌱 Optional enrichment:

- **”How To Add Conversational Memory To LLMs Using LangChain”**, article, https://supermemory.ai/blog/how-to-add-conversational-memory-to-llms-using-langchain/ , ~21 min - A practical 2025 guide demonstrating how to implement short-term conversational memory in LangChain using LangGraph. Covers techniques such as message trimming and summarization, with real-world use cases like a therapy chatbot
- **“Introducing LangChain Agents: 2024 Tutorial with Example”**, article, Bright Inventions, ~10 min
Link: https://brightinventions.pl/blog/langchain-agents-python-tutorial-example/
Explores LangChain agents that can reason and use tools dynamically, contrasting them with static chains.

- **“Fine-tune Llama 2 with LoRA: Customizing a large language model for Q&A”**, blog, AMD ROCm, ~4 min
Link: https://rocm.docs.amd.com/en/latest/tutorials/llama2_finetuning_lora.html
Shows how to fine-tune Llama 2 for Q&A using LoRA, including efficient adaptation of large models.

- **“Cognee + LlamaIndex: Building Powerful GraphRAG Pipelines”**, article, Analytics Vidhya, ~8 min
Link: https://www.analyticsvidhya.com/blog/2025/02/cognee-llamaindex/
Introduces graph-based RAG: ingesting documents into a semantic graph with LlamaIndex and Cognee for enhanced retrieval.

---

## 🛠️ Assignment:

Build a PDF or Markdown Q&A pipeline that:

- Uses a local open-weight LLM (e.g. Llama 2 or 3) and a local vector database (e.g. Chroma or Qdrant)
- Parses and chunks document content (PDF or Markdown)
- Generates embeddings and indexes them in the local vector store

- Accepts a user question, retrieves relevant chunks, and produces a context-aware answer through LangChain or LlamaIndex
- *Optional:* include fallback inference via Hugging Face Inference Endpoint or similar cloud API

---

### 🤝 How to use the weekly sync:

In small peer groups, each participant demos their pipeline (one using LangChain, one using LlamaIndex if possible). Compare design decisions, document loaders, chunking strategies, and vector DB setup. Discuss retrieval behavior, evaluate generated outputs, and troubleshoot any memory, retrieval, or formatting issues collaboratively.

## Module 6 — Designing a RAG Pipeline and Advanced Orchestration

**Time budget:** 10 hours

---

### 🎯 Learning goals:

- Design a working RAG loop that grounds LLM responses in source documents
- Orchestrate multi-step workflows using LangChain
- Tune and evaluate performance of a knowledge-base Q&A system
- (Optional) Explore agentic patterns or multi-agent orchestration

---

### 📚Required resources:

1. **“Build a Retrieval Augmented Generation (RAG) App: Part 1”**, tutorial, LangChain Docs, https://python.langchain.com/docs/tutorials/rag/ — *~30 min*
   Step-by-step guide to building a RAG pipeline using LangChain: document loading, text splitting, embeddings, vector search, and RetrievalQA orchestration. A complete example in ~50 lines of code.

2. **“Building a Simple RAG Pipeline with LlamaIndex”**, article, Medium, https://medium.com/@tuhinsharma121/building-a-simple-yet-powerful-rag-pipeline-with-llamaindex-a-christmas-carol-example-077f01062de1 — *~10 min*
   Ingests a PDF corpus, chunks and indexes it in ChromaDB using LlamaIndex, and queries it with a custom prompt template. Includes evaluation strategies for retrieval quality.

3. **“Building a RAG Application Using LlamaIndex”**, article, https://www.kdnuggets.com/building-a-rag-application-using-llamaindex , ~15 min - A 2025 tutorial showing how to create a question-answering app using LlamaIndex with modular retriever and query engines. Includes chunking strategy tips and thoughtful architectural choices.
   Video: https://www.youtube.com/watch?v=vNpxWaVzky8

4. **“How to Evaluate Retrieval Augmented Generation (RAG) Systems”**, article, RidgeRun, https://www.ridgerun.ai/post/how-to-evaluate-retrieval-augmented-generation-rag-systems — *~10 min*
   Introduces precision@k, recall@k, MRR, and other retrieval metrics. Explains how to manually and programmatically detect hallucinations and evaluate response quality.

5. **“Best Practices in RAG Evaluation: A Comprehensive Guide”**, article, Qdrant Blog, https://qdrant.tech/blog/rag-evaluation-guide/ — *~12 min*
   Practical guide for evaluating both retrieval and generation in RAG. Offers tuning advice on chunking, embedding model selection, and top-k parameterization.

6. **“Building RAG from Scratch (Open-source only!)”**, tutorial, LlamaIndex Docs, https://docs.llamaindex.ai/en/v0.10.19/examples/low_level/oss_ingestion_retrieval.html — *~20 min*
   Walks through building a RAG system entirely with open-source tools: ingestion, embedding (SentenceTransformers), local vector DB (Chroma), and generation using llama.cpp.

---

## 🌱 Optional enrichment:

1. **“Using with LangChain – LlamaIndex Integration”**, docs, LlamaIndex, https://docs.llamaindex.ai/en/v0.10.18/community/integrations/using_with_langchain.html — *~15 min*
   Explains how to use LlamaIndex’s retriever and query engine components inside LangChain chains, allowing orchestration of indexing + tool calls.

2. **“Use LlamaIndex & LangChain Together | Tutorial”**, video, YouTube, https://www.youtube.com/watch?v=MDxlVnAsvUk — *~15 min*
   Live-coded example of combining LlamaIndex and LangChain to construct a retrieval pipeline. Useful for learners balancing both frameworks.

3. **“LangChain vs LlamaIndex: A Detailed Comparison”**, article, DataCamp Blog, https://www.datacamp.com/blog/langchain-vs-llamaindex — *~10 min*
   A side-by-side comparison of strengths: LangChain excels at chaining/tooling; LlamaIndex is ideal for ingestion and index abstraction. Good for architecture decisions.

---

## 🛠️ Assignment:

Build and test a complete Retrieval-Augmented Generation (RAG) pipeline using your own document corpus (e.g. Markdown files, project notes, local PDFs). Use LangChain, LlamaIndex, or both to:

- Ingest and chunk documents
- Generate embeddings with SentenceTransformers or Hugging Face models
- Store vectors in Chroma or Qdrant
- Retrieve top-k chunks and feed them into an LLM for generation
- Experiment with different prompt formats, top-k values, and chunk sizes
- Evaluate retrieval quality using metrics like precision@k or manual groundedness checks
- Identify and reduce hallucinations through prompt tuning and improved context formatting

Final deliverable: Python script or notebook + brief notes on what worked, what didn't, and evaluation results.

---

## 🤝 How to use the weekly sync:

In small peer groups (2–3 learners), schedule a 1–2 hour discussion to:

- Compare pipeline designs (e.g., LangChain vs LlamaIndex orchestration, chaining strategies, retriever types)
- Share findings from top-k tuning, chunk overlap experiments, or embedding choices
- Review one another's outputs to detect hallucinations or retrieval errors
- Optionally debug together if retrieval/generation breaks or underperforms

This session helps learners reflect on architectural trade-offs and build confidence in interpreting and improving retrieval quality.

---

## 📝 Notes

RAG pipelines can require a lot of moving parts. If you run into issues with local setup:

- Start with a smaller corpus (e.g., 1–2 documents).
- Use a **Jupyter notebook in Colab** with GPU runtime.
- Use LangChain's `RetrievalQA` wrapper to simplify pipeline orchestration

# Module 7 — Developing and Deploying LLM Applications

**Time budget:** 10 hours

---

## 🎯 Learning goals:

- Wrap LLM inference in an API or UI interface (FastAPI, Streamlit, or Gradio)
- Containerize the full app using Docker
- Explore hosting options for local or remote deployment
- Understand performance and scaling concerns (e.g., vLLM, batching)

---

## 📚 Required resources:

1. **Your Local LLM using FastAPI**, article, https://plainenglish.io/blog/your-local-llm-using-fastapi, ~8 min
   Shows how to build a simple FastAPI API around a Hugging Face Transformer model and expose a `/generate` endpoint for prompt-based inference.

2. **Build a basic LLM chat app – Streamlit Docs**, tutorial, https://docs.streamlit.io/develop/tutorials/chat-and-llm-apps/build-conversational-apps, ~30 min
   Official guide to creating a conversational LLM UI using `st.chat_input` and `st.chat_message`. Builds three levels of chatbots: echo, streaming, and memory-enabled.

3. **Build AI Chatbot in 5 Minutes with Hugging Face and Gradio**, tutorial, https://kdnuggets.com/2023/06/build-ai-chatbot-5-minutes-hugging-face-gradio.html, ~5 min
   Quick hands-on tutorial using Gradio and DialoGPT to build and deploy a chatbot UI locally or on Hugging Face Spaces.

4. **How to Dockerize your LLM FastAPI Project**, article, https://blog.devgenius.io/dockerize-your-llm-fastapi-project-%EF%B8%8F-7e4f872dff4f, ~5 min
   End-to-end example of containerizing an LLM FastAPI app using Docker. Includes Dockerfile, docker-compose setup, and local test instructions.

5. **Meet vLLM: For faster, more efficient LLM inference and serving**, blog post, https://www.redhat.com/en/blog/meet-vllm-faster-more-efficient-llm-inference-and-serving, ~8 min
   Introduces vLLM's PagedAttention and batching optimizations that deliver major throughput improvements over Hugging Face Transformers.

6. **Containerize and Deploy ML Models with FastAPI & Docker**, article, https://hemz.medium.com/containerize-and-deploy-ml-models-with-fastapi-docker-d8c19cc8ef94, ~7 min
   Offers a general Docker deployment pattern for ML APIs using FastAPI, including Dockerfile and best practices for container builds.

---

## 🌱 Optional enrichment:

- **Deploying an App to Hugging Face using Docker**, guide, https://brighteshun.medium.com/deploying-a-sentiment-analysis-app-to-hugging-ace-using-docker-541e32ed1003, ~5 min
  Shows how to package a Gradio app with Docker and deploy it to Hugging Face Spaces with Git.

- **vLLM: Easy, Fast, and Cheap LLM Serving with PagedAttention**, blog post, https://blog.vllm.ai/2023/06/20/vllm.html, ~10 min
  Technical deep dive into the vLLM architecture, continuous batching, and GPU memory optimizations.

- **Machine Learning Model Deployment with FastAPI, Streamlit and Docker Compose**, article, https://medium.com/latinxinai/fastapi-and-streamlit-app-with-docker-compose-e4d18d78d61d, ~7 min
  Walkthrough for chaining a FastAPI backend with a Streamlit UI using Docker Compose for local development.

---

## 🛠️ Assignment:

Build and containerize a simple LLM-powered application. Choose either FastAPI, Streamlit, or Gradio to create an interface that accepts user input and displays the model's response. Wrap a prompt-based or simple RAG chain and expose it through your selected interface. Write a Dockerfile and verify that you can run the app locally via `docker run`. Provide simple usage instructions (e.g., `curl` for APIs or URL for UIs). Conduct basic performance testing—measure latency for a few requests and reflect on any performance-related behaviors (e.g., CPU vs GPU, batch size).

Finally, evaluate at least one remote or cloud-based hosting option (e.g., Hugging Face Spaces, AWS, or Streamlit Community Cloud). Document:

- Which hosting platform you would choose and why (cost, ease of use, hardware support, etc.)

- What changes (if any) would be needed to deploy your app there

---

### 🤝 How to use the weekly sync:

Demo your containerized app to a peer. Walk through your app interface, Docker setup, and how you launch the container. Compare approaches across FastAPI, Streamlit, and Gradio. Discuss container size, latency, and performance observations. This session is a chance to share lessons learned and troubleshoot issues together.

### 📝 Notes

If Docker setup is new or causes errors:

- Follow the step-by-step deployment articles provided.
- Use `docker run` or `docker-compose` locally.
- If Docker is not supported on your machine, fall back to running the app with `uvicorn` or `streamlit run`.

## Module 8 — Evaluation, Guardrails, and Capstone Project

**Time budget:** 12 hours

---

### 🎯 Learning goals:

- Evaluate model outputs using test cases or eval frameworks (e.g. string overlap metrics, semantic similarity, truthfulness testing)
- Implement basic guardrails (e.g. hallucination detection, content filtering, prompt-injection monitoring, user confirmation loops)
- Design a prototype LLM-powered application solving a domain-specific problem
- Build a runnable MVP capstone prototype

---

### 📚 Required resources:

1. **LLM Evaluation Metrics: The Ultimate LLM Evaluation Guide**, *Article*

   https://wandb.ai/ayush-thakur/llm-eval/reports/LLM-Evaluation-Metrics-The-Ultimate-LLM-Evaluation-Guide--Vmlldzo2ODg4MDc0
   A comprehensive and up-to-date guide that introduces core LLM evaluation metrics (BLEU, ROUGE, METEOR, embedding-based, truthfulness). Gives learners practical

context for when and how to apply string overlap vs semantic similarity scoring, and highlights tools to implement these evaluations on open-weight models.

2. **Guardrails AI project homepage**, *Documentation / quick-start*
   https://github.com/guardrails-ai/guardrails
   Learn how to define structured prompts and enforce safe, well-formed outputs using the Guardrails Python library. Covers RAIL specs, validation logic, and re-asking workflows for correcting LLM outputs.

3. **Rebuff: LLM Prompt Injection Detector**, *GitHub + docs*
   https://github.com/protectai/rebuff
   Rebuff is a multi-layer open-source framework that detects prompt injection using heuristics, LLM-based detectors, and a vector DB of known attacks. Simple to install and integrate into your own pipeline.

4. **Safeguarding LLMs Against Prompt Injection Attacks**, *Article*
   https://blog.securityinnovation.com/securing-llms-against-prompt-injection-attacks
   Explains real-world prompt injection risks in an OWASP-style format and explores Rebuff, LLM Guard, and PyRIT as open-source protection tools.

5. **LLM Guardrails: Types of Guards**, *Article*
   https://arize.com/blog-course/llm-guardrails-types-of-guards/
   Introduces a practical guardrail taxonomy (input, output, content filtering, post-processing). Useful for designing a layered defense strategy in your app.

6. **Test-Driven Development for Prompt Injection Testing**, *Blog Tutorial*

   https://medium.com/@ari_in_media_res/test-driven-development-for-prompt-injection-testing-fail-fast-learn-iterate-55e0c85cadc3
   Shows how to create test cases for injection exploits using Rebuff, Chroma, and local models like Ollama — a great hands-on reference for building your own eval suite.

---

## 🌱 Optional enrichment:

1. **GuardReasoner: Towards Reasoning-based LLM Safeguards**, *Preprint / code*
   https://arxiv.org/abs/2501.18492
   A cutting-edge technique using reasoning-trained classifiers to filter unsafe or off-topic prompts. Includes open weights and dataset for experimentation.

2. **LlamaFirewall: Secure Agent Guardrails for Autonomous LLM Agents**, *Preprint*
   https://arxiv.org/abs/2505.03574
   Guardrail framework for multi-agent systems, capable of runtime prompt injection and code execution detection.

3. **NVIDIA NeMo Guardrails**, *Toolkit / paper*
   https://arxiv.org/abs/2310.10501
   Rule-based dialogue manager and guardrail system from NVIDIA — useful if you're exploring alternatives to Guardrails AI.
4. **Guardrails for LLMs: a tooling comparison**, *Blog*
   https://www.fuzzylabs.ai/blog-post/guardrails-for-llms-a-tooling-comparison
   Compares Guardrails AI and NVIDIA NeMo Guardrails across features like latency, guard scope, and usability. Helps learners critically assess tool choices.

---

## 🛠️ Assignment:

Each team (2–3 learners) must **scope, design, and implement a runnable MVP** of their capstone LLM application. Choose a real-world domain (e.g. customer support chatbot, educational tutor, productivity assistant). Integrate components built in earlier modules (prompt design, RAG pipeline, vector DB, UI layer). Include at least one evaluation mechanism (e.g. test cases, BLEU/ROUGE/semantic similarity check, truthfulness tests) and implement **at least one form of guardrail** (e.g. output filtering, Guardrails AI, Rebuff, or a user confirmation loop). Prepare to demo the working MVP, explain how evaluation and guardrails are implemented, and reflect on limitations and next steps. **Reuse components** built in earlier modules where possible.

**Optional:**

**Also include** a brief **architecture diagram or tech stack summary** with your submission. This should illustrate how major components (e.g., vector DB, LLM interface, guardrails, UI/API) fit together, and which tools you selected. A simple hand-drawn sketch, flowchart, or bullet list is fine — the goal is to help reviewers understand your design decisions and trade-offs.

---

## 🤝 How to use the weekly sync:

Teams present their MVP-in-progress: walk peers through the app workflow, evaluation tests used, guardrail logic added, and any edge cases still unhandled. Peers and instructor provide feedback on scope, UX, risks, model behavior or testing gaps. Teams iteratively refine their prototype based on this feedback before final submission.