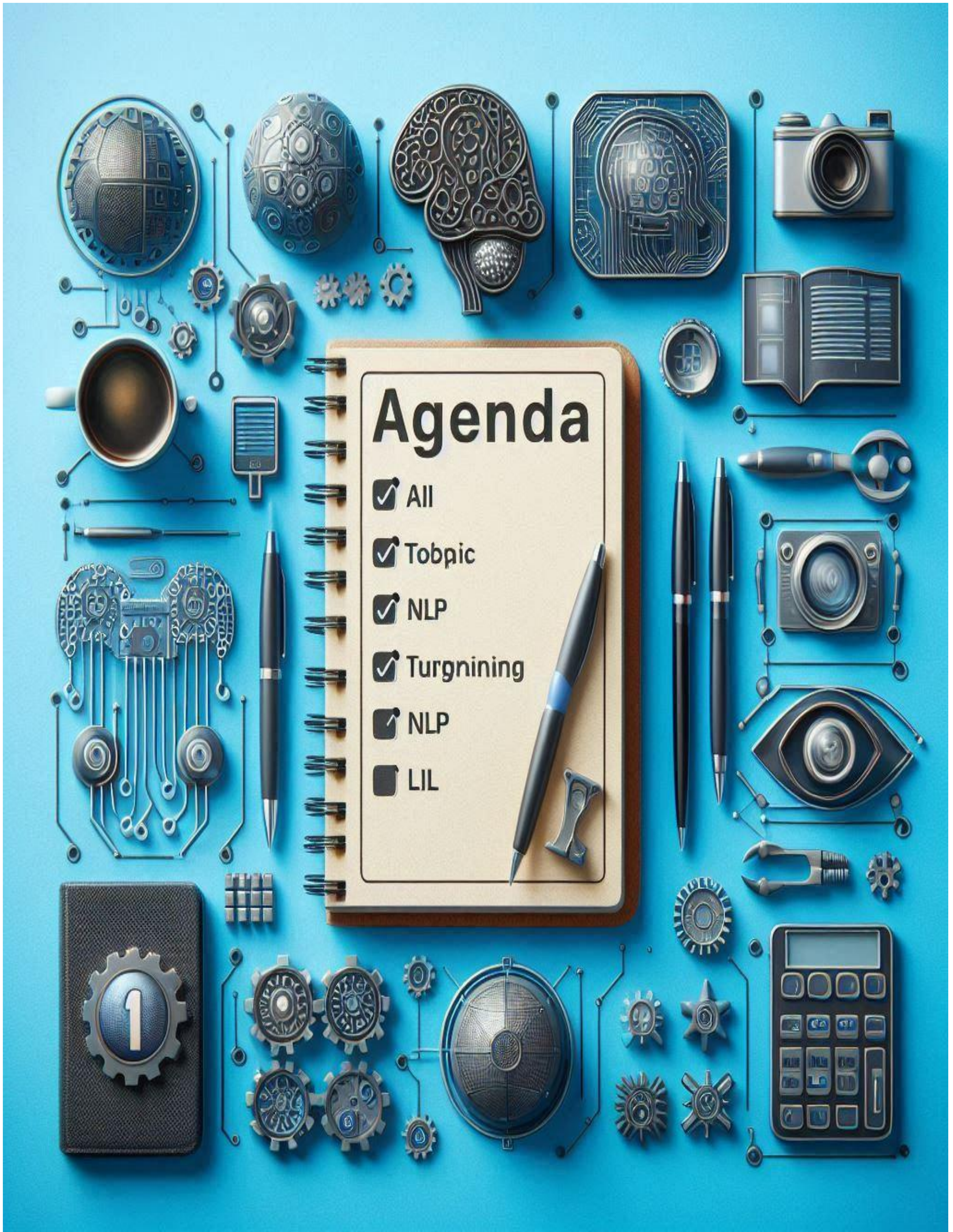
NTI
National Telecommunication Institute

What are your expectations for the course?



Agenda

- LLM foundations
- LangChain architecture
- practical use cases
- future trends

Introduction to Large Language Models (LLMs)

What are Large Language Models (LLMs)?

- LLMs are advanced AI systems trained on massive text datasets.
- They understand, generate, translate, and summarize human language.
- Examples: GPT (by OpenAI), PaLM (by Google), LLaMA (by Meta).

Why Are LLMs Important in AI Today?

- Enable chatbots, virtual assistants, code generation, legal and medical document understanding, and more.
- Democratize access to complex AI capabilities with natural language interfaces.
- Core enabler of **Generative AI**.

The Evolution of Language Models

Era	Approach	Description
Pre-2010	Rule-based & Statistical	Hand-crafted rules, N-grams, Hidden Markov Models (HMMs)
2010–2017	Classical ML	SVMs, Logistic Regression, TF-IDF, Word2Vec (2013), GloVe
2017	**Transformers (Vaswani et al.)**	“Attention is all you need” paper – a turning point
2018–2020	Contextualized Embeddings	BERT, GPT-2, XLNet, RoBERTa
2020–2022	Large-scale Pretraining	T5, GPT-3, Megatron, PaLM
2023+	Foundation & Multimodal Models	GPT-4, Llama, Claude, Gemini, Mistral

Key Milestones:

- **Word2Vec (2013)** – Captured word relationships using vectors.
- **Transformer (2017)** – Introduced self-attention mechanism.
- **BERT (2018)** – Bi-directional understanding of context.
- **GPT Series (2018–2023)** – Generative, autoregressive models.
- **T5 (2019)** – Unified text-to-text framework.
- **GPT-4 (2023)** – Multimodal, instruction-following, RAG-ready.

Impact:

- Enabled state-of-the-art performance across NLP benchmarks.
- Foundation for tools like ChatGPT, Google Bard, and Claude.

Key Concepts in LLMs

1. Deep Learning & Neural Networks

- LLMs are built on **deep neural networks**, especially **transformers**.
- They use multiple layers of attention to learn complex language patterns.
- **Self-attention** enables models to weigh the importance of words in context.

2. Understanding vs. Generation

- **Natural Language Understanding (NLU)**:

  Goal: Extract meaning and intent.
- **Natural Language Generation (NLG)**:

  Goal: Produce coherent and human-like language.

3. Scale & Training Data

- Modern LLMs are trained on **hundreds of billions of tokens** (text units).

Scaling in:

- **Model size**: Parameters (e.g., GPT-3 = 175B).
- **Data diversity**: Books, code, web pages, social media.

Trade-offs:

- More data = better generalization.
- But also higher **compute cost**, risk of **bias** and **hallucination**.

Benefits and Challenges of LLMs

Advantages of LLMs:

- **Flexibility**
  Can perform multiple tasks with minimal fine-tuning (e.g., summarization, Q&A, translation).
- **Generative Capabilities**
  Create coherent, human-like text, images (via prompts), and even code.
- **Scalability**
  Once trained, can serve millions of users via APIs or integrations.
- **Few-shot/Zero-shot Learning**
  Can generalize with little or no task-specific data

Challenges of LLMs:

- **Bias and Fairness**
  May inherit stereotypes, harmful assumptions from training data.
- **High Computational Cost**
  Training requires vast hardware (TPUs/GPUs) and energy.
- **Ethical and Safety Concerns**
  Risk of misinformation, malicious use, data privacy breaches.
- **Hallucination**
  Models may confidently generate **false or unverifiable** information.

Introduction to LangChain

What is LangChain?

- **LangChain** is an open-source framework for building applications powered by **LLMs**.
- It helps developers **connect language models** with external tools, data sources, and **user interfaces**.
- Built primarily for **modular, composable LLM workflows**.

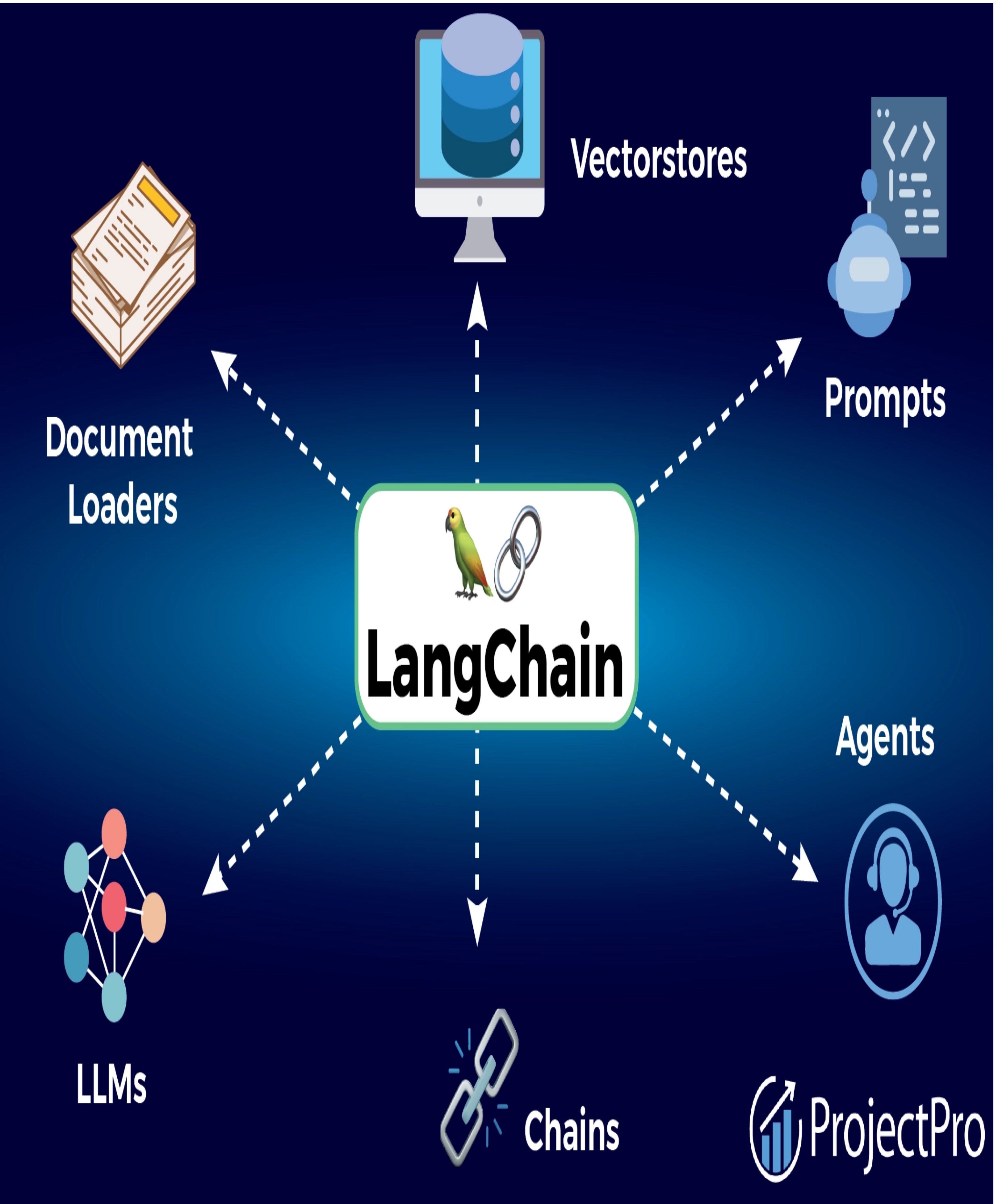
Purpose of LangChain:

- Make LLMs **actionable and context-aware**.
- Provide an ecosystem for:
  - **Prompt management**
  - **Memory** (conversation context)
  - **Agents** (reasoning + action-taking)
  - **Tool/Document chains**

How LangChain Bridges LLMs with Applications

- **Abstraction Layer**:
  Simplifies model invocation and pipeline creation.
- **Tool Integration**:
  Connects LLMs to APIs, databases, search engines, and more.
- **Contextual Interfaces**:
  Manages dynamic conversation flow with **memory and state**.
- **Custom Agents**:
  Enables creation of AI agents that reason, decide, and act in real-time.

Why Use LangChain?



1. Enhanced Model Integration & Orchestration

LangChain helps **coordinate multiple LLM components** into a single workflow.

Makes it easy to integrate:

- **Different models** (e.g., OpenAI, Cohere, Hugging Face)
- **External tools** (APIs, file systems, databases)

Supports **multi-step reasoning** and **agent-based execution**.

2. Streamlined Prompt Engineering

Centralized management of:

Prompt templates **Input variables** **Dynamic formatting**

Promotes **modularity** and **reusability** of prompts.

3. Chaining Calls & Managing State

Chain multiple tasks (e.g., search → summarize → answer) using:

- **LLMChains**
- **Sequential or parallel execution**

Maintains memory/state across user interactions using:

- **ConversationBufferMemory**
- **Vector stores + retrievers**

Enables **context-aware** chatbots and assistants.

Core Components of LangChain

1. **Chains:** A sequence of calls or steps involving LLMs and functions.

Types:

- **LLMChain**: A single prompt → LLM → output.
- **SequentialChain**: Series of steps, where output from one feeds into the next.
- **RouterChain**: Dynamically routes inputs to appropriate subchains.

2. Agents: Intelligent decision-makers that choose which tools or steps to use.

- They ask themselves:
  "What should I do next?"
- Support **tool use**, **multi-step planning**, and **dynamic behaviour**.

3. Memory Modules: Track and store past interactions, allowing **contextual conversations**.

Types:

- **ConversationBufferMemory**: Stores entire chat history.
- **SummaryMemory**: Keeps summarized context.
- **VectorStoreRetrieverMemory**: Stores knowledge in embeddings.

4. Tools & Integrations Connect LLMs to real-world capabilities:

- **APIs** (e.g., weather, search)
- **Databases** (SQL, NoSQL)
- **Document loaders** (PDFs, web pages)
- **Vector databases** (FAISS, Pinecone, Chroma)
- Enables **retrieval-augmented generation (RAG)**.

LangChain Architecture Overview

Flow of Data:

User Input

→ Passed to a prompt template.

Prompt Template

→ Fills placeholders and sends formatted prompt to a chain or agent.

Chains / Agents

→ Determine how to handle the task (simple LLM call vs multi-step reasoning).

LLM Interface

→ Communicates with the selected large language model (e.g., OpenAI GPT-4, Anthropic Claude).

Memory & Tools

Memory: Stores conversation history/context.

Tools: Perform external actions like searching the web, calling APIs, or accessing documents.

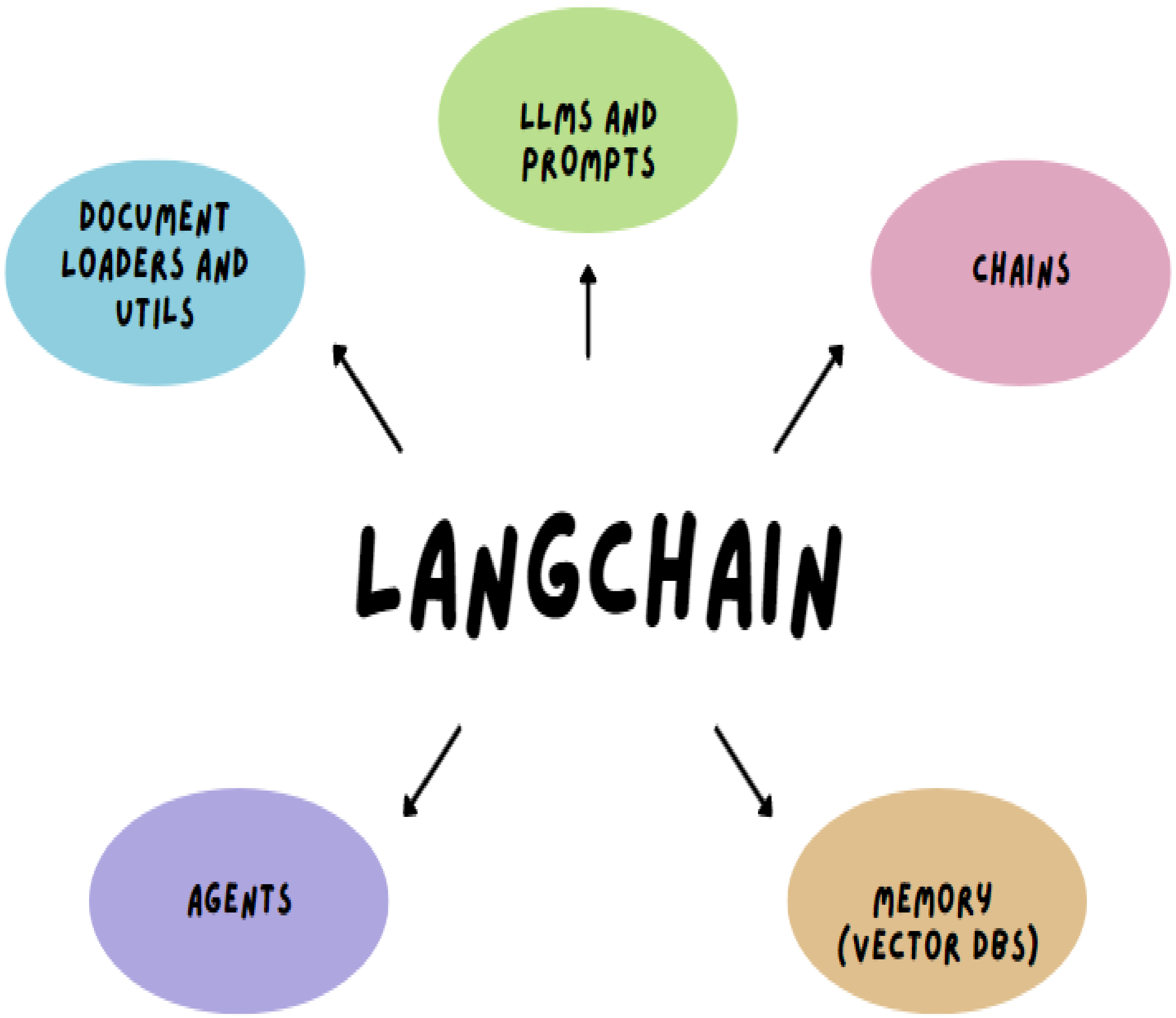
Output Parser

→ Processes and formats the LLM's raw response.

Final Output

→ Delivered back to the user or to an external app/system.

Types of Chains



1. Sequential Chains

Each step runs **after** the previous one.

Output of step A → Input of step B.

Example:

Extract topic from text.

Generate summary of that topic.

Ask follow-up questions.

2. Parallel Chains

Multiple prompts run **at the same time** using the same input.

Example:

One chain summarizes a text.

Another translates it.

A third extracts keywords.

Final results are **combined** or used differently.

Agents

What Are Agents?

Agents are **LLM-powered decision-makers**.

Unlike fixed chains, agents can:

Decide what action to take.

Choose the right tool or prompt.

Change behavior based on **real-time input**.

Role in Decision-Making:

Given a user input, an agent will:

Analyze the task

Select the appropriate tool or step

Execute, observe results

Repeat if needed (looping)

Dynamic Tool & Prompt Selection:

Agents can:

Dynamically pick tools (e.g., calculator, web search, database query).

Adapt prompts based on task type (e.g., translate vs summarize).

Enable **flexible**, **context-aware** applications.

Memory Management in LangChain

Why Memory Matters?

LLMs are **stateless** by default — they don’t remember previous interactions unless explicitly given.

LangChain provides **memory modules** to **store, retrieve, and reuse context** over multiple turns.

Memory Modules in LangChain

ConversationBufferMemory

Stores exact messages from recent turns.

ConversationSummaryMemory

Uses LLM to summarize previous interactions, saving space.

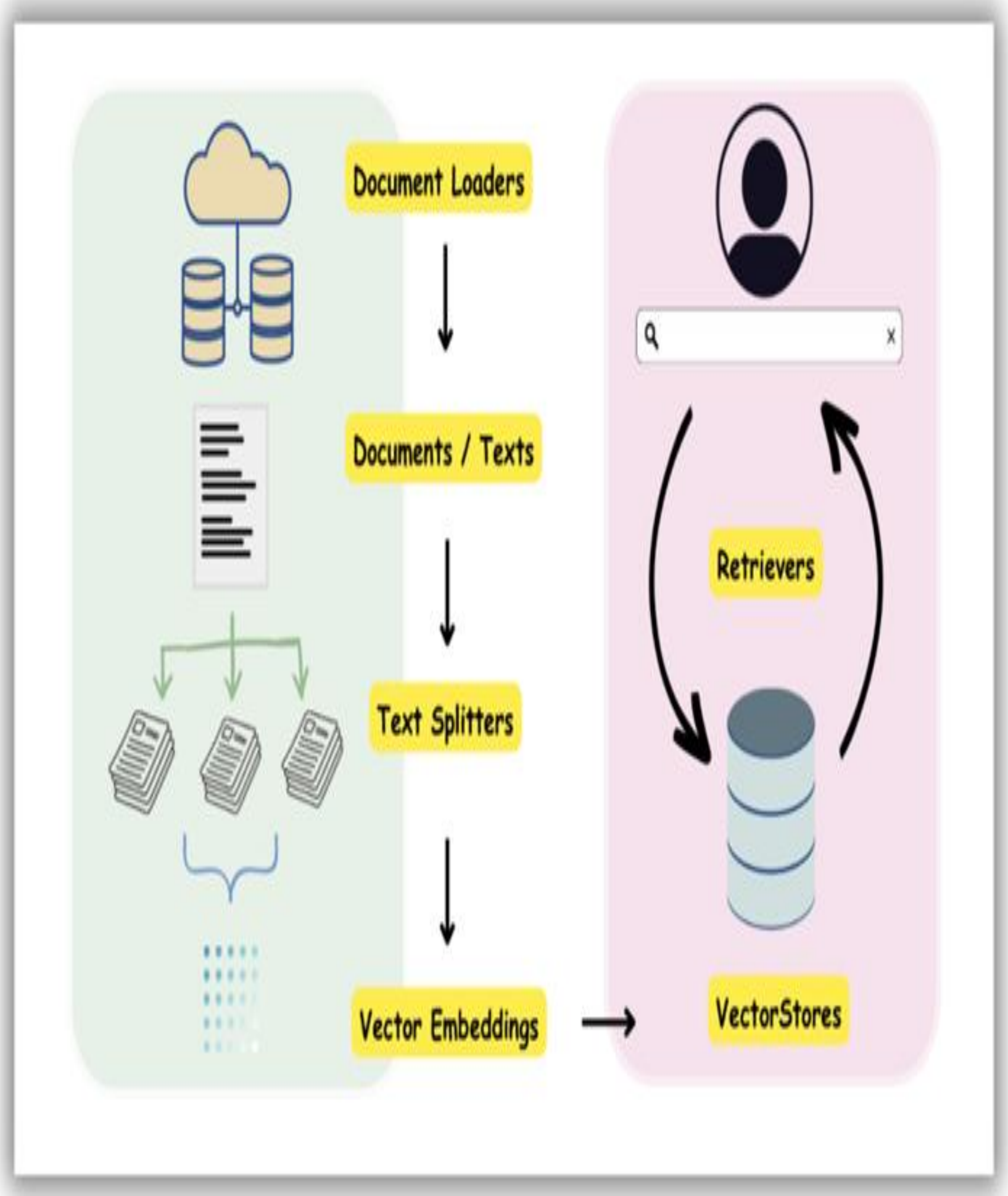
VectorStoreRetrieverMemory

Stores content (e.g., documents, answers) as **embeddings** in a vector database (FAISS, Pinecone).

CombinedMemory

Mix of short-term + long-term strategies.

Basic LangChain Workflow



A basic LangChain setup includes:

- **Prompt Template** – format the input.
- **LLM** – define the language model (e.g., OpenAI, Cohere).
- **Chain** – connect prompt → model → output.
- **Run** – pass user input to generate results.

How to craft effective prompts for LLMs

PROMPT PERFECT

1. Persona
Define **who** the LLM should act as.

2. Instruction
Clearly specify the **exact task** with no ambiguity.

3. Context
Provide **background info** to help the model understand **why** it's doing the task.

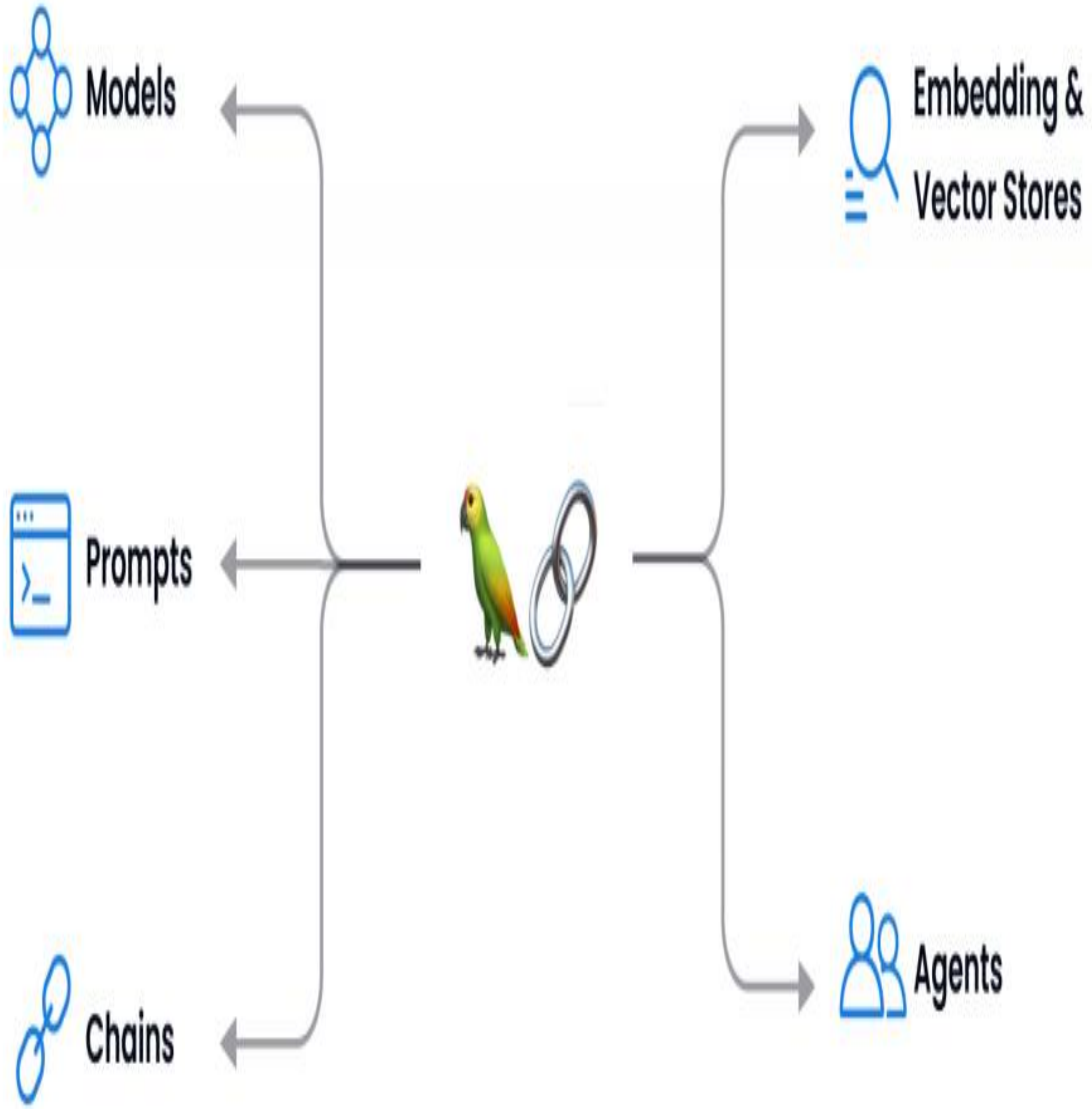
4. Format
Define the **desired structure** of the output.

5. Audience
Specify **who** the text is for (and their expertise level).

6. Tone
Set the **voice style** of the output.

7. Data
Supply the **core content or input** the model needs to process.

Creating Custom Chains



What is a Custom Chain?

A **custom chain** is a user-defined sequence of steps that:

Preprocesses input

Interacts with one or more LLMs or tools

Postprocesses and formats the final output

Useful when built-in chains are too generic for your use case

Key Customization Options

Add logic like:

- Text preprocessing
- API or database lookups
- Custom prompts or chaining with other components

Extend or combine with:

- LLMChain, SequentialChain, or RouterChain
- Memory, Tools, or Agents

LangChain Pros and cons

LangChain Pros:

- Easy-to-use **chain and agent abstraction**
- Rich ecosystem: tools, memory, retrievers
- Native **support for RAG** (Retrieval-Augmented Generation)
- Flexible integrations with LLMs, APIs, databases

LangChain Cons:

- **Can be complex** for simple use cases
- **Steeper learning curve** for full agent/tool workflows
- **Fast-changing APIs** may lead to breaking changes

When to Choose LangChain?

- You want **modular, extensible LLM applications**
- You need **multi-step workflows or agent logic**
- You plan to **integrate external tools/data sources**

Thanks