

# **WELCOME !!**

## **CodeCraft : Unleashing Langchain & LLMs**

**– Week 1 : Day 0**

**– Nandan Hemanth**

# Prerequisites

- Basic knowledge of Python programming Language
- How legos building blocks works

# What is Langchain?



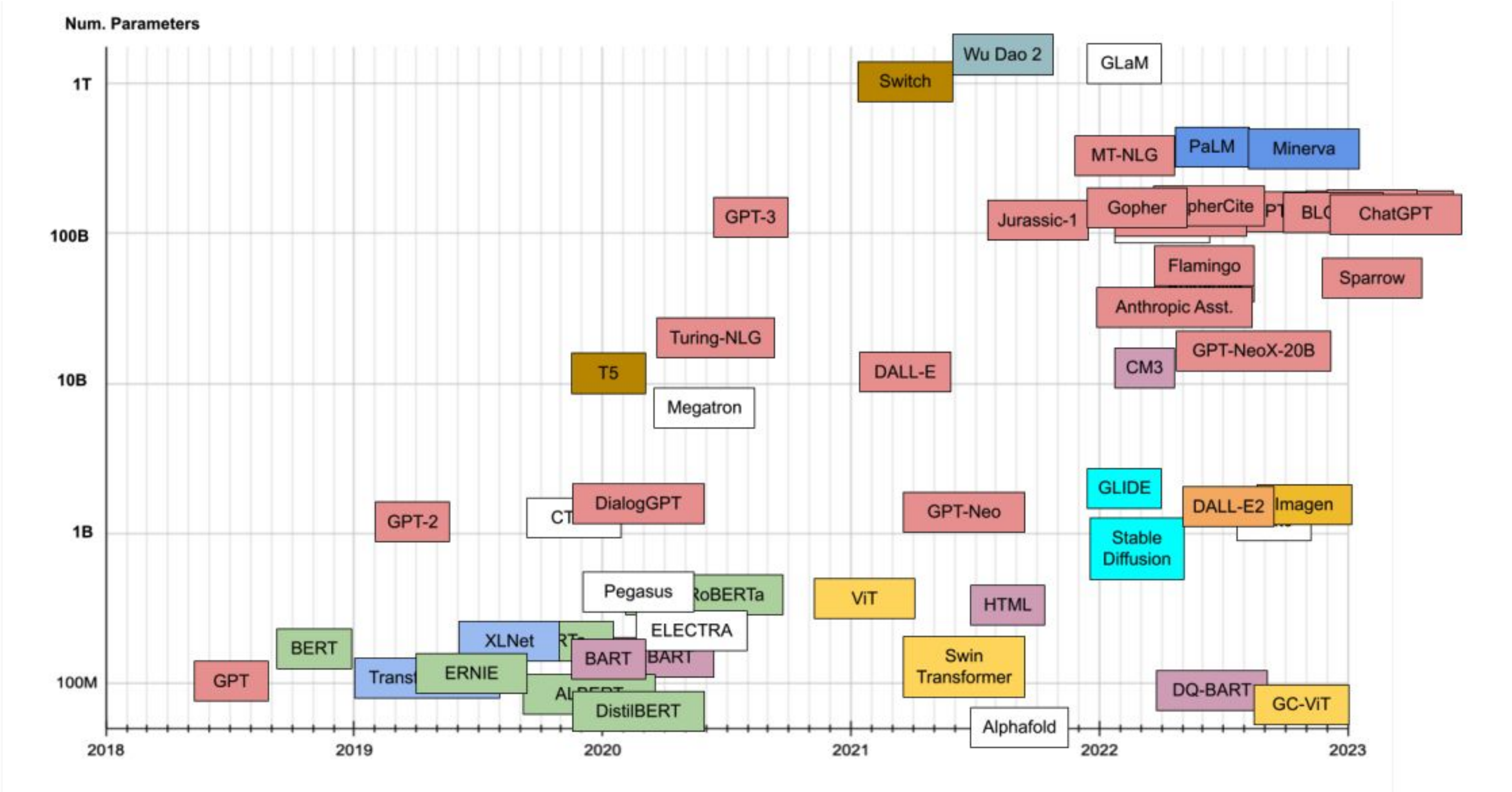
- LangChain is an open-source framework designed to simplify the creation of applications powered by Large Language Models (LLMs).
- Think of it as a set of tools (like Legos!) that allows you to connect different building blocks to achieve remarkable results in language processing tasks.

# What are Large Language Models (LLMs)?

- LLMs are like language geniuses trained on massive amounts of text data.
- They are powerful AI models capable of various language-related tasks, including:
  - Generating text: Writing creative content, translating languages, crafting different writing styles.
  - Understanding text: Answering questions, summarizing information, analyzing sentiment.



# Growth rate of LLMs:



# How is an LLM trained?

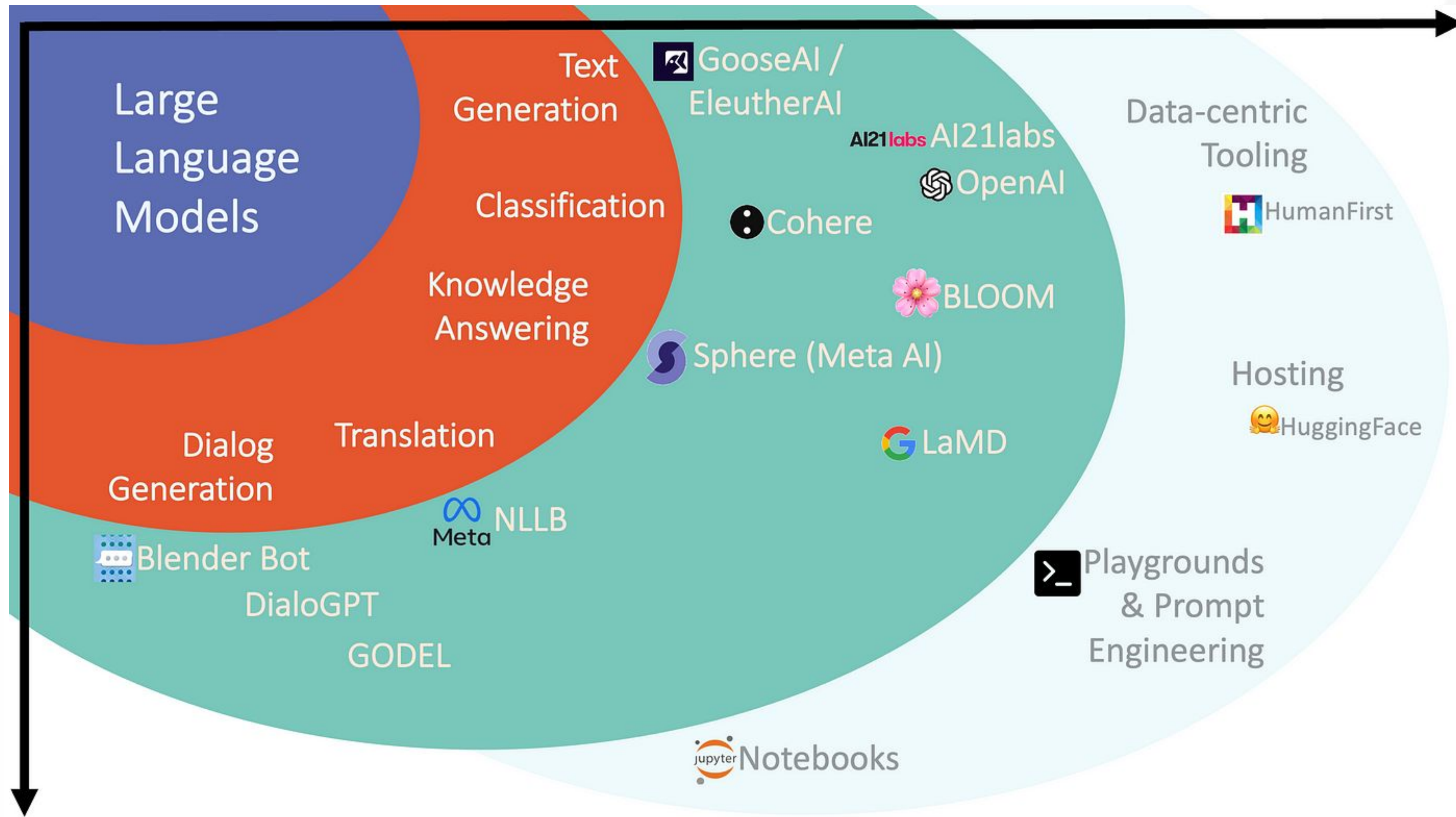
- LLMs are trained on massive datasets of text and code, similar to how a child learns language by being exposed to speech and text.
- This training allows them to:
  - Recognize patterns in language and identify relationships between words and phrases.
  - Predict what word or phrase is likely to come next, allowing them to generate text, translate languages, and answer questions.



# What is the LLM architecture?

- While various architectures exist, transformers are a popular choice for LLMs due to their ability to:
  - Process information in parallel, making them efficient for handling large amounts of data.
  - Model relationships between different parts of the text, leading to better understanding and generation capabilities.

# Evaluating LLMs:





# Bridging the Gap of Langchain & LLMs:

- LLMs are incredibly powerful, but their raw capabilities can be difficult to access and control directly. Think of them as complex engines with immense potential, but not readily accessible to users.
- LangChain acts as a user-friendly interface that simplifies this interaction. It provides the tools and structure to effectively connect users with the capabilities of LLMs.

# Bridging the Gap of Langchain & LLMs:

- LangChain offers building blocks called "chains", which are essentially instructions and data that you use to tell the LLM what to do. You can combine these chains in various ways to achieve specific tasks.
- This allows users to:
  - Provide clear instructions to the LLM about the desired outcome.
  - Control the flow of information, specifying what data the LLM should access and how it should be processed.
  - Fine-tune the LLM's behavior for optimal performance in specific tasks.

# Applications of the Synergy:

- Machine translation: Translate text from one language to another seamlessly.
- Chatbot development: Create engaging chatbots that can answer questions and hold conversations.
- Text analysis and summarization: Analyze large amounts of text and extract key information or create summaries.
- Content creation: Generate different creative text formats like poems, scripts, or marketing materials.



## Real-world Scenario example:

- *Imagine you want to build a chatbot that can answer customer questions about your products.*

Using LangChain, you could:

- Create a chain that fetches relevant product information from your database.
- Design another chain that allows the LLM to understand and respond to customer queries in a natural and informative way.
- Combine these chains within LangChain to create your fully functional chatbot.

# References:

- Generative AI with Langchain:

<https://drive.google.com/file/d/1remcULuWB1loxx8Ud9KsQ1dPPm0bI1ks/view?usp=sharing>

- Langchain Documentation:

[https://python.langchain.com/docs/get\\_started/introduction](https://python.langchain.com/docs/get_started/introduction)

**Join the Whatsapp Group:**





# Feedback and Improvements:

- Pace of teaching? (slow / fast/ more explanation)
- Energizers and Refreshers (games/ quizzes/ activity)
- Any personal issues?

# Thank You!

PESU

# **WELCOME !!**

## **CodeCraft : Unleashing Langchain & LLMs**

**– Week 1 : Day 1**

**– Nandan Hemanth**



# Installing Software



Python: <https://www.python.org/downloads/>

Anaconda : <https://www.anaconda.com/download>

VSCode: <https://code.visualstudio.com/download>

# Installing Software – Python

Remember to check “Add python.exe to PATH” box!





# Installing Software - Anaconda

You can keep all the default settings for Anaconda





# Installing Code Editor

VSCode (Recomended): <https://code.visualstudio.com/download>

1st Alt (Zed Editor): <https://zed.dev/>

Alt Code Editors - Atom, Vim, Pycharm

# Installing Software – Code Editor

You can keep all the default settings!





# Command Prompt commands – Windows

– Terminal commands (Windows):

`'cd <dir name>'` – change directory (forward)

`'cd ..'` – change back to previous directory

`'dir'` – list all directory content

`'mkdir'` – make new directory

`'cd <X>:'` – change volume

`'code .'` – open current directory in code editor



# Terminal Commands - Ubuntu

- Terminal commands (Ubuntu):

`'cd <dir name>'` - change directory (forward)

`'cd ..'` - change back to previous directory

`'ls'` - list all directory content

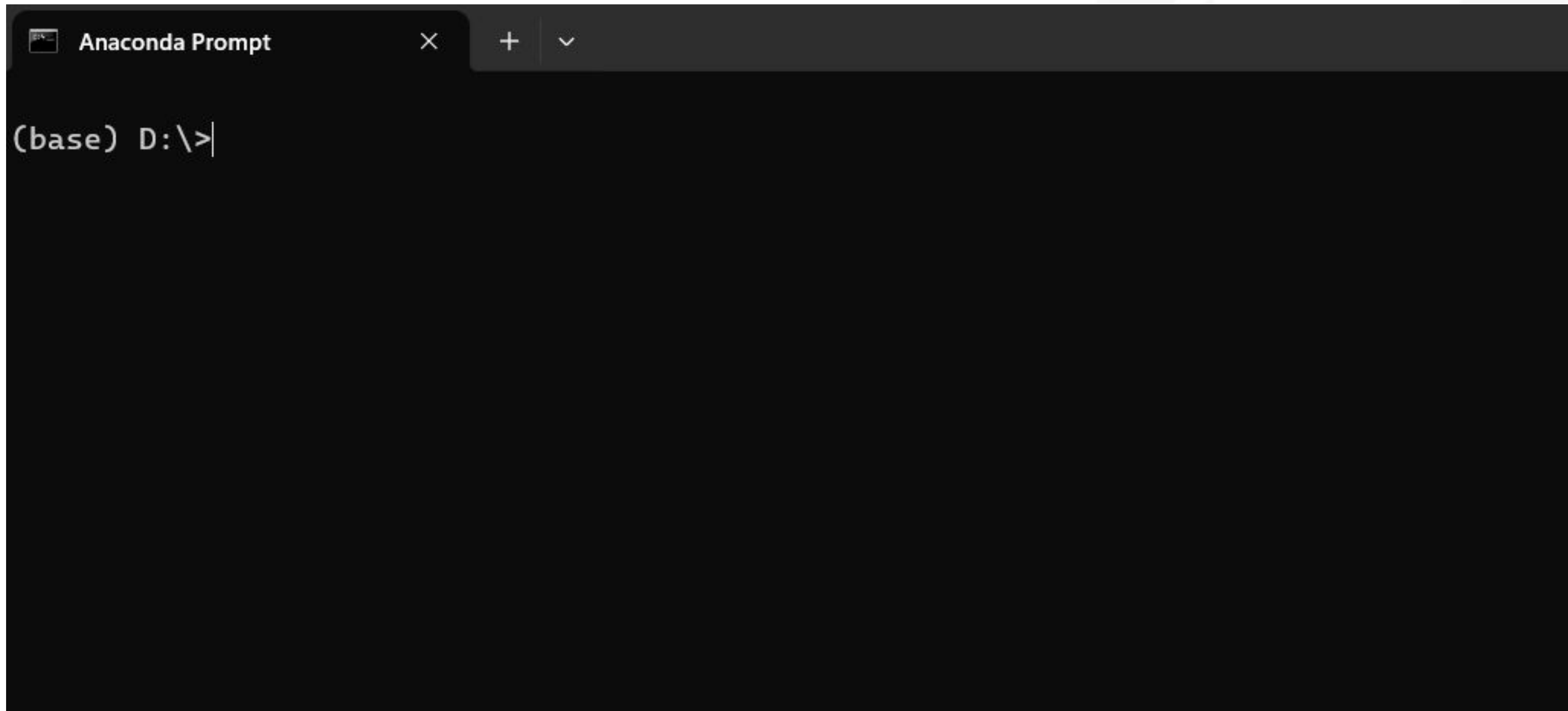
`'mkdir'` - make new directory

`'cd <X>:'` - change volume

`'code .'` - open current directory in code editor

# Anaconda Setup – “conda”

## 1. Open Anaconda Prompt



```
(base) D:\>
```

# Anaconda Setup – “conda”

2. Navigate to the working directory using “Terminal Commands”
3. Activate Virtual Environment



# Anaconda Setup – “conda”

```
Anaconda Prompt
(base) D:\>cd Guild
(base) D:\Guild>cd ..
(base) D:\>cd PES
(base) D:\PES>dir
Volume in drive D is New Volume
Volume Serial Number is D64D-6029

Directory of D:\PES

21-02-2024  12:12    <DIR>          .
13-02-2024  14:44    <DIR>          Blabber
04-03-2024  15:24    <DIR>          PESUIO
05-12-2023  22:36    <DIR>          ST
               0 File(s)                0 bytes
               4 Dir(s)  9,825,681,408 bytes free

(base) D:\PES>cd PESUIO
(base) D:\PES\PESUIO>mkdir Week1
(base) D:\PES\PESUIO>cd Week1
(base) D:\PES\PESUIO\Week1>
```

# Anaconda Setup – Activating Virtual environment

- `'conda env list'` – list all virtual envs
- `'conda create -n <env_name> <python_version>'` – create a new virtual environment with the specified python version
- `'conda activate <env_name>'` – activates the virtual environment

# Anaconda Setup – Activating Virtual environment

```
Anaconda Prompt - conda cr  ×  +  v

==> WARNING: A newer version of conda exists. <==
current version: 23.7.4
latest version: 24.1.2

Please update conda by running

    $ conda update -n base -c defaults conda

Or to minimize the number of packages updated during conda update use

    conda install conda=24.1.2

## Package Plan ##

environment location: E:\Ananconda_main\envs\test

added / updated specs:
- python=3.10

The following packages will be downloaded:
```



# Anaconda Setup - Activating Virtual environment

```
#  
# To activate this environment, use  
#  
#     $ conda activate test  
#  
# To deactivate an active environment, use  
#  
#     $ conda deactivate  
  
(base) D:\PES\PESUIO\Week1>conda activate test  
(test) D:\PES\PESUIO\Week1>|
```

# Installing dependencies using 'pip'

- "pip" - preferred installer program
- All libraries and frameworks can be installed using 'pip'
- Eg: (*run these commands in the cmd or terminal*)
  - "pip install pandas"
  - "pip install numpy"

# Installing langchain

- Command Prompt or Terminal:
  - a. `"pip install langchain"`
  - b. `"pip install langchain-community"`
  
- Anaconda Prompt:
  - c. `"conda install langchain"`
  - d. `"conda install langchain-community"`



# Installing langchain

```
Command Prompt
D:\PES\PESUIO\Week1>pip install langchain langchain-community
Requirement already satisfied: langchain in e:\python\lib\site-packages (0.0.344)
Requirement already satisfied: langchain-community in e:\python\lib\site-packages (0.0.25)
Requirement already satisfied: PyYAML>=5.3 in e:\python\lib\site-packages (from langchain) (6.0.1)
Requirement already satisfied: SQLAlchemy<3,>=1.4 in e:\python\lib\site-packages (from langchain) (2.0.23)
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in e:\python\lib\site-packages (from langchain) (3.9.1)
Requirement already satisfied: anyio<4.0 in e:\python\lib\site-packages (from langchain) (3.7.1)
Requirement already satisfied: dataclasses-json<0.7,>=0.5.7 in e:\python\lib\site-packages (from langchain) (0.6.3)
Requirement already satisfied: jsonpatch<2.0,>=1.33 in e:\python\lib\site-packages (from langchain) (1.33)
Collecting langchain-core<0.1,>=0.0.8 (from langchain)
  Downloading langchain_core-0.0.13-py3-none-any.whl.metadata (978 bytes)
Collecting langsmith<0.1.0,>=0.0.63 (from langchain)
  Downloading langsmith-0.0.92-py3-none-any.whl.metadata (9.9 kB)
Requirement already satisfied: numpy<2,>=1 in e:\python\lib\site-packages (from langchain) (1.26.2)
Requirement already satisfied: pydantic<3,>=1 in e:\python\lib\site-packages (from langchain) (2.5.2)
Requirement already satisfied: requests<3,>=2 in e:\python\lib\site-packages (from langchain) (2.31.0)
Requirement already satisfied: tenacity<9.0.0,>=8.1.0 in e:\python\lib\site-packages (from langchain) (8.2.3)
```



# Thank You!

PESU

# **WELCOME !!**

## **CodeCraft : Unleashing Langchain & LLMs**

**– Week 1 : Day 2**

**– Nandan Hemanth**



# Topics to be covered:

- Installation of the environment
- Available LLMs (open and closed source)
- LLM prompting and chatting
- Prompt Templates
- What are chains
- How to build sequential chains
- Memory

## Available LLMs:

- Open-source LLMs: In the world of Large Language Models (LLMs), open-source LLMs stand out as those whose underlying code and architecture are accessible to the public.  
Eg: GPT2, gpt4all, llama-1, huggingface, etc
- Closed-source LLMs: In contrast to open-source LLMs, closed-source LLMs are those where the underlying code and architecture are not publicly accessible. They are developed and maintained by private organizations or companies and often treated as proprietary technology.  
Eg: GPT4, Jurassic-1 Jumbo, etc

# What are chains?

- In the world of LangChain, chains are the fundamental building blocks that enable you to connect various elements and functionalities to achieve specific tasks.
- They act like instructions or modules that work together to process information and data using Large Language Models (LLMs).



# Understanding Chains

- Think of them as reusable components that perform specific actions within your LangChain application.
- Each chain encapsulates a particular functionality, like data processing, LLM interaction, or manipulation of outputs.
- You can combine and sequence multiple chains to create complex workflows and achieve more intricate results.

# Types of Chains:

- **LLM Chains:** These are the core chains that interact directly with LLMs, providing instructions, data, and processing the LLM's outputs.
- **Data Processing Chains:** These chains handle tasks like data pre-processing, cleaning, and formatting before feeding it to the LLM.
- **Utility Chains:** These are pre-built chains offering functionalities like summarization, translation, or sentiment analysis, simplifying your workflow.

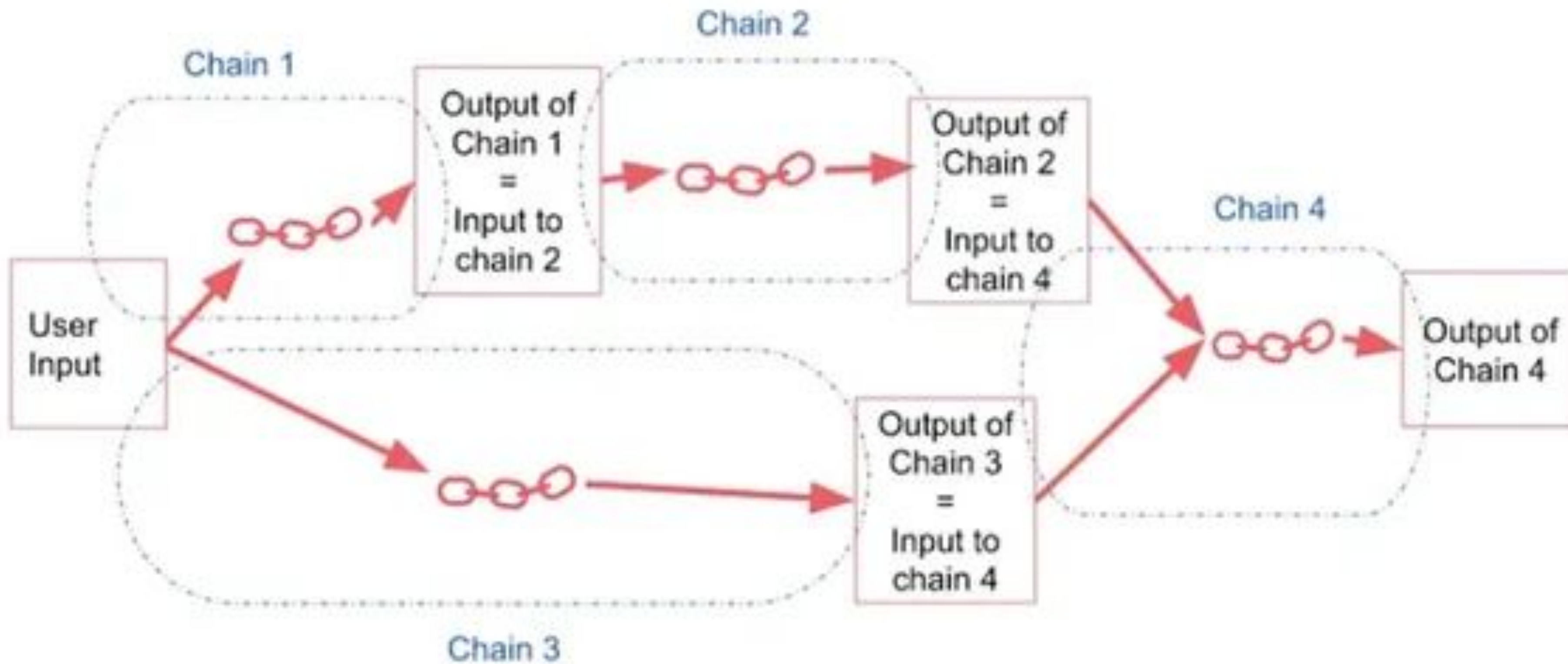
## Real-world Example:

Imagine you want to translate a document using LangChain and an LLM:

- Data Processing Chain: This chain would read the document and format the text for the LLM.
- LLM Chain: This chain would send the formatted text to the LLM and receive the translated version.
- Output Chain: This chain might handle formatting the translated text for display.
- You would then connect these chains in the desired order to form a complete workflow for document translation.



# Real-world Example:



# Benefits of Using Chains:

- **Modularity:** Break down complex tasks into smaller, reusable chains, promoting code organization and maintainability.
- **Flexibility:** Combine different chains in various ways to create diverse applications, catering to specific needs.
- **Readability:** Chains improve code readability by clearly separating different functionalities within your LangChain application.

# What are Prompt Templates?

- In LangChain, prompt templates serve as powerful tools to standardize and streamline the process of creating prompts for Large Language Models (LLMs).
- They act as reusable blueprints that define the structure and essential elements of your prompts, reducing redundancy and ensuring consistency.



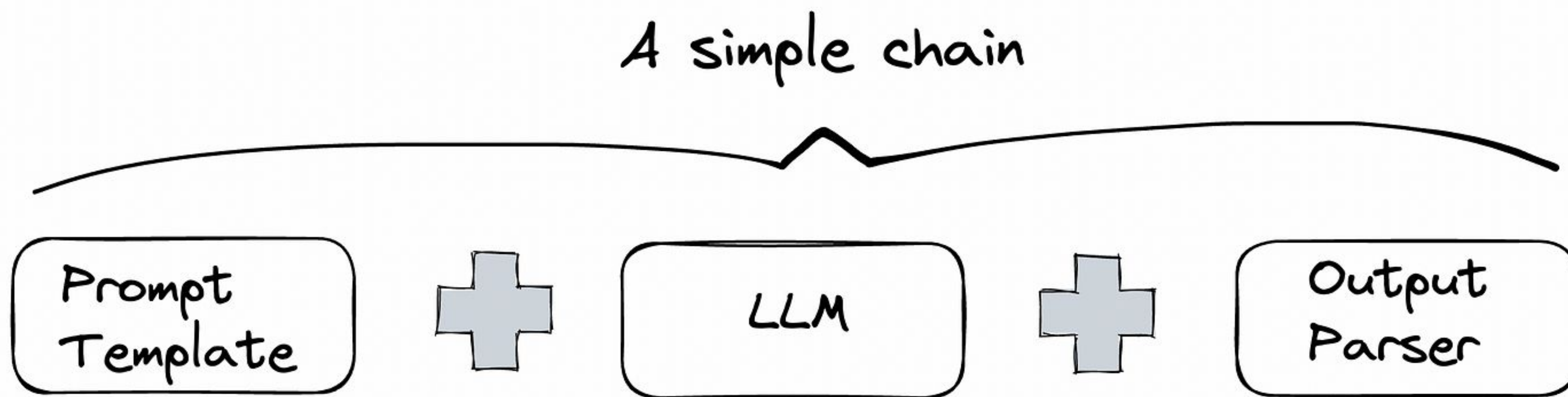
# Understanding Prompt Templates:

- Imagine building prompts for a specific task, like summarizing text snippets. Instead of writing each prompt individually, you can create a template that captures the core elements.
- This template would define placeholders for specific information, such as the text snippet to be summarized.
- When needed, you can fill these placeholders with the actual data to generate individual prompts for each text snippet, maintaining a consistent structure and format.

# Key Components of a Prompt Template:

- Instructions: These guide the LLM on how to approach the task, providing context and setting expectations.
- Input Variables: These represent placeholders within the template that will be filled with actual data when generating individual prompts. They can be named and typed for clarity.
- Template String: This combines the instructions and placeholders, forming the overall structure of the prompt.

# Prompt Template:





# Benefits of Using Prompt Templates:

- **Efficiency**: Saves time and effort by eliminating the need to write individual prompts from scratch.
- **Consistency**: Ensures all prompts adhere to the same structure and format, leading to consistent LLM outputs.
- **Flexibility**: Allows you to easily adapt prompts to different datasets or tasks by modifying the input variables and template string.
- **Readability**: Improves the readability and maintainability of your LangChain code by separating prompts from the core logic.

# What are sequential Chains:

- Within the realm of LangChain and its modular building blocks called "chains," sequential chains play a crucial role in achieving specific tasks.
- They are a specific type of chain characterized by the ordered execution of multiple chains one after the other.

# Types of Sequential Chains:

- Simple Sequential Chain: This is the most basic form, consisting of a linear sequence of chains where the output from one directly feeds into the next.
- Sequential Chain: This offers more flexibility, allowing for multiple inputs and outputs at various stages of the sequence. This enables more complex workflows with branching or merging of data streams.



# Benefits of Sequential Chains:

- Clear Workflow: The linear structure provides a well-defined execution order, making it easier to understand and manage complex workflows.
- Modularity and Reusability: Individual chains within the sequence remain reusable in different contexts, promoting code efficiency.
- Efficient Data Flow: The chain-to-chain data transfer optimizes the flow of information throughout the process, ensuring seamless progression towards the desired outcome.

# What is memory in langchain?

In LangChain, memory plays a crucial role in enabling contextual awareness and persistence within your language-processing applications.

# Properties of memory in langchain?

## Maintain State Between Calls:

- Unlike traditional applications that treat each interaction as isolated, LangChain allows you to preserve information from previous interactions using memory.
- This is particularly valuable for tasks requiring contextual understanding, such as building chatbots that remember previous conversations or creating personalized experiences based on user interactions.



# Properties of memory in langchain?

## Information Storage:

- User inputs and responses during interactions.
- Outputs generated by LLMs in previous stages of your LangChain application.
- Intermediate data created during the processing pipeline.

# Properties of memory in langchain?

## Different Memory Integrations:

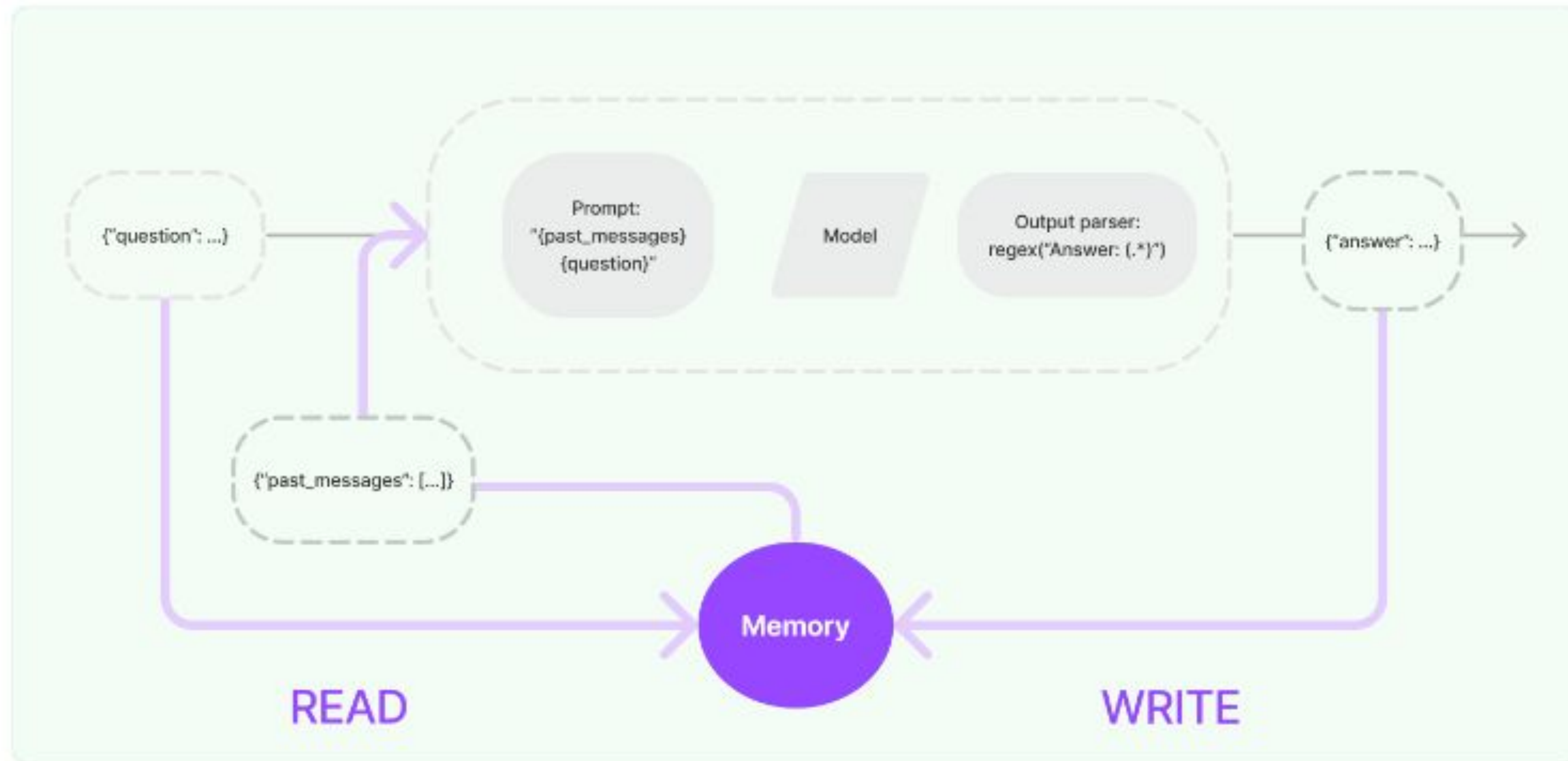
- In-memory storage: Suitable for temporary data within a single execution.
- Persistent storage: Enables data retention across different executions, allowing for longer-term context building.
- Database integrations: Connect LangChain to external databases for more robust and scalable storage solutions.

# Benefits of Using Memory in LangChain:

- Improved Contextual Understanding: Preserving information from previous interactions allows LLMs to better understand context, leading to more relevant and coherent responses.
- Personalized User Experiences: By remembering past interactions, your applications can adapt to individual users and provide a more personalized experience.
- Streamlined Workflows: Memory can simplify processing by storing intermediate data, eliminating the need to recalculate it in subsequent stages of your workflow.



# Benefits of Using Memory in LangChain:

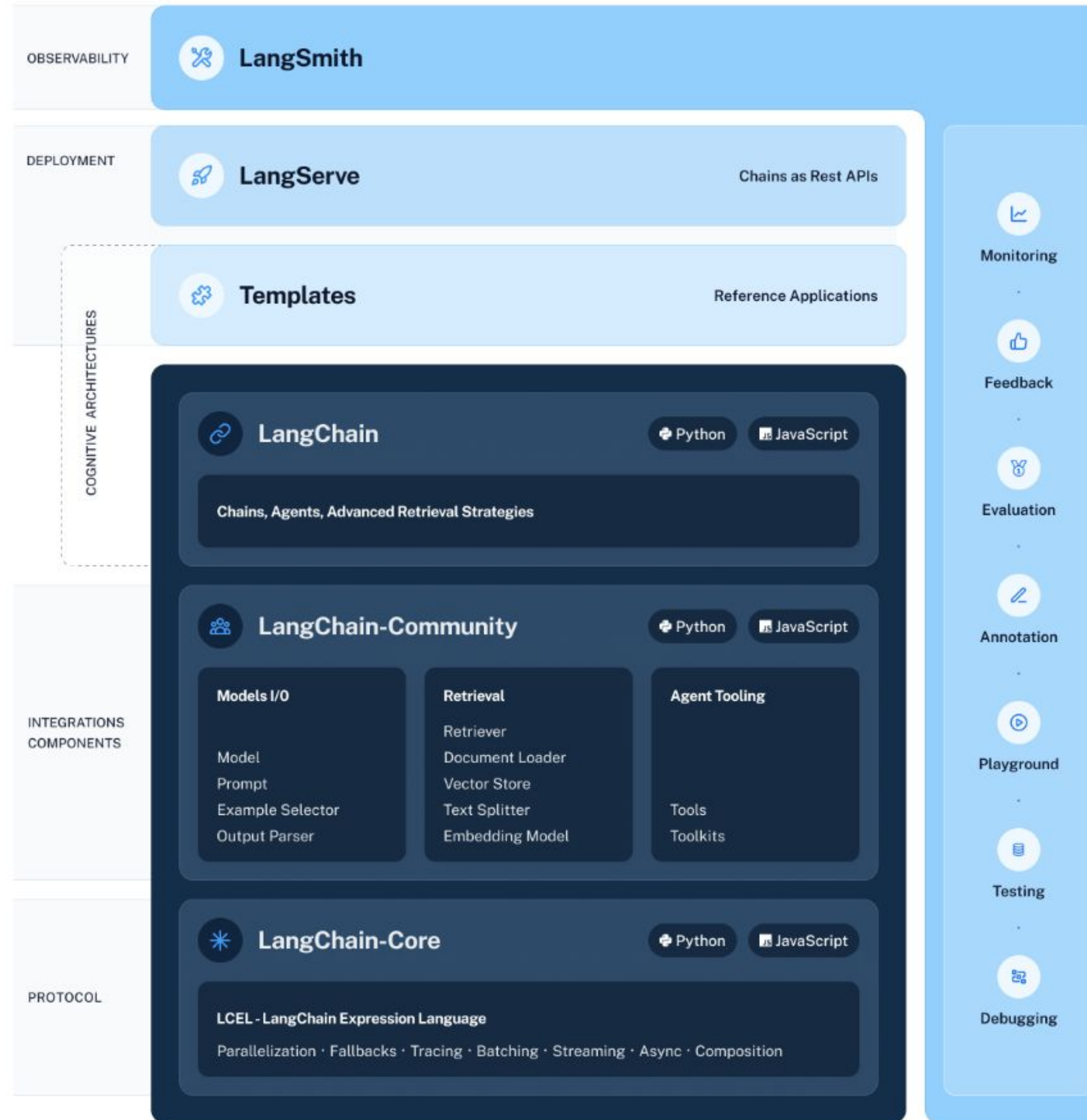


# Challenges and Considerations:

- Data Privacy: Carefully consider data privacy implications when utilizing memory for storing user information. Implement appropriate security measures and adhere to relevant regulations.
- Memory Management: Depending on the chosen integration and the amount of data stored, managing memory usage is crucial to ensure optimal performance and avoid resource exhaustion.



# Langchain-





# Thank You!

PESU

# **WELCOME !!**

## **CodeCraft : Unleashing Langchain & LLMs**

**– Week 1 : Day 3**

**– Nandan Hemanth**

# Topics to be covered:

- What are Agents in Langchain?
- What are indexes in langchain?
- What are document loaders?



# What are Agents?

- In LangChain, agents act as the central decision-makers and orchestrators within your language processing applications.
- They are the brains behind the operations, responsible for planning, executing, and managing the overall workflow.

# Functions of Agents:

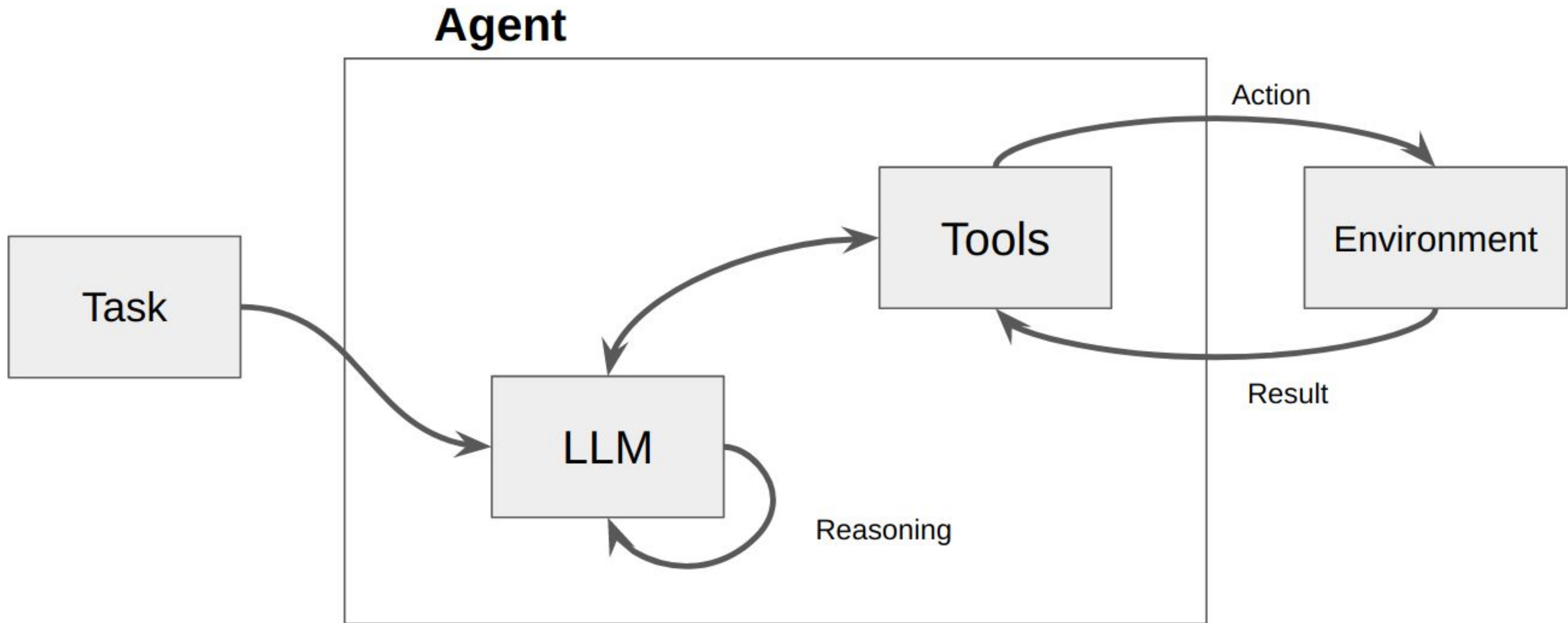
- Defining the Workflow: Agents determine the sequence of actions and interactions within your application. They specify which tools to use, in what order, and what data to pass between them.
- Interacting with Tools: Agents utilize various tools available in LangChain to achieve specific tasks. These tools can be for:
  - Data processing: Cleaning and manipulating data before feeding it to LLMs.

# Functions of Agents:

- LLM interaction: Sending prompts to LLMs and receiving their outputs.
- Utility functions: Performing common tasks like summarization or translation.
- Managing Memory: Agents can leverage memory to store information and maintain context across interactions. This preserved information helps LLMs better understand the current situation or personalize responses based on user history.



# Functions of Agents:



# Advantages of Agents:

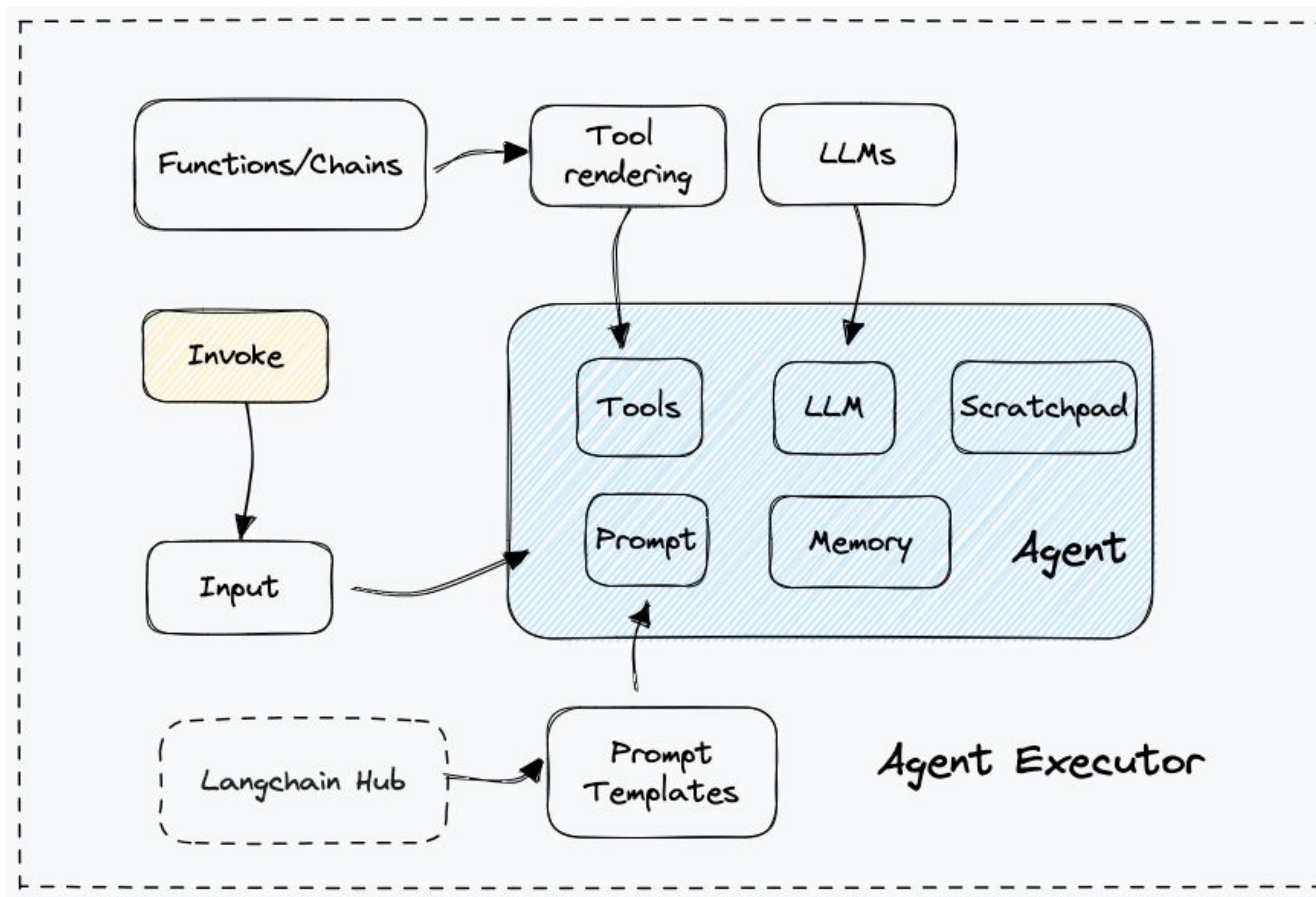
- **Decision Making:** Unlike traditional chains (sequences of calls) that follow a predefined path, agents can make decisions based on the outcome of tasks or user input. This allows for more dynamic and adaptable applications.
- **Flexibility:** Agents enable the creation of modular applications by combining different tools and workflows within a single agent. This promotes reusability and simplifies complex tasks.
- **Efficiency:** By using agents, you can create more streamlined and efficient applications, reducing the need for extensive and repetitive code for each step of the workflow.

# Basic Analogy of Agents:

- Imagine building a robot that can answer your questions. The agent is the robot's control system. It decides how to access information (using data processing tools), interacts with the language model (using LLM interaction tools) to find the answer, and finally presents the answer to you (potentially using utility functions).
- By understanding the role of agents, you can unlock the full potential of LangChain and create powerful, dynamic, and adaptable language processing applications.



# Basic Analogy of Agents:



# What is Document loading?

In LangChain, document loading refers to the process of reading and preparing data for use within your application. It acts as the foundation for your application, as it provides the essential raw materials for the subsequent processing and interactions with Large Language Models (LLMs).



# The Role of Documents:

LangChain works with data in the form of "Documents". These documents can be various types of textual data, such as:

- Plain text files
- Web pages
- Emails
- Transcripts



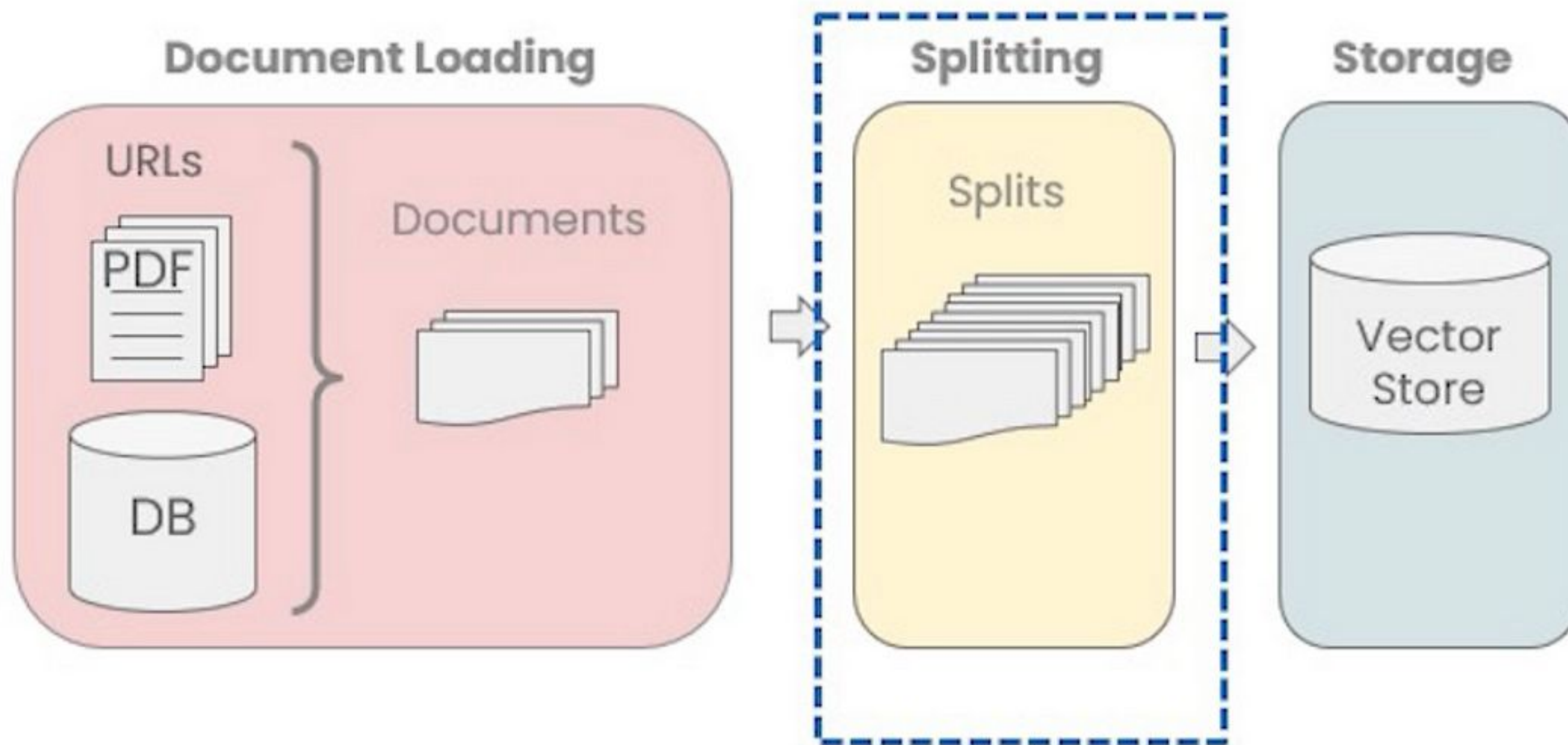
# Benefits of Document Loading:

- **Flexibility**: LangChain offers a wide variety of loaders to handle diverse data sources, allowing you to work with various types of textual information.
- **Efficiency**: Loaders streamline the process of accessing and preparing data, saving you time and effort compared to manual data manipulation.
- **Consistency**: By converting data into the LangChain "Document" format, you ensure compatibility and smooth interaction with other components within your application.

# Common Document Loaders:

- **TextLoader:** Reads text files.
- **CSVLoader:** Reads data from CSV (comma-separated values) files.
- **JSONLoader:** Reads data from JSON (JavaScript Object Notation) files.
- **WebPageLoader:** Fetches and processes content from web pages.
- **DirectoryLoader:** Loads multiple files from a directory, applying a chosen loader to each file.

# Common Document Loaders:





# Advanced Features in Document Loading:

- Lazy loading: Only load documents when explicitly needed, improving memory usage for large datasets.
- Splitting documents: Divide large documents into smaller chunks for efficient LLM processing.

# What is Indexing in langchain?

In LangChain, indexing plays a crucial role in optimizing the performance and efficiency of your language-processing applications. It's a behind-the-scenes process that involves storing and organizing information in a way that allows for faster retrieval and processing, especially when dealing with large datasets.

# Functions of Indexing:

- Extracting key information from documents. This could be words, phrases, or other relevant features.
- Storing this information along with document references in a special data structure called a "vector store".
- Building an efficient indexing system that allows for quick searching and retrieval of documents based on the extracted information.



# Benefits of Indexing:

- **Faster Retrieval:** By using the index, LangChain can locate relevant documents in a fraction of the time compared to searching through the entire dataset. This is particularly beneficial for large datasets or applications that require frequent document access.
- **Improved Efficiency:** The indexing process helps optimize resource utilization by efficiently accessing and processing relevant information instead of scanning entire documents each time.
- **Enhanced Scalability:** As your data volume grows, the indexing system efficiently scales to handle the increased demand, ensuring consistent performance even with large datasets.

# Key Components of Indexing:

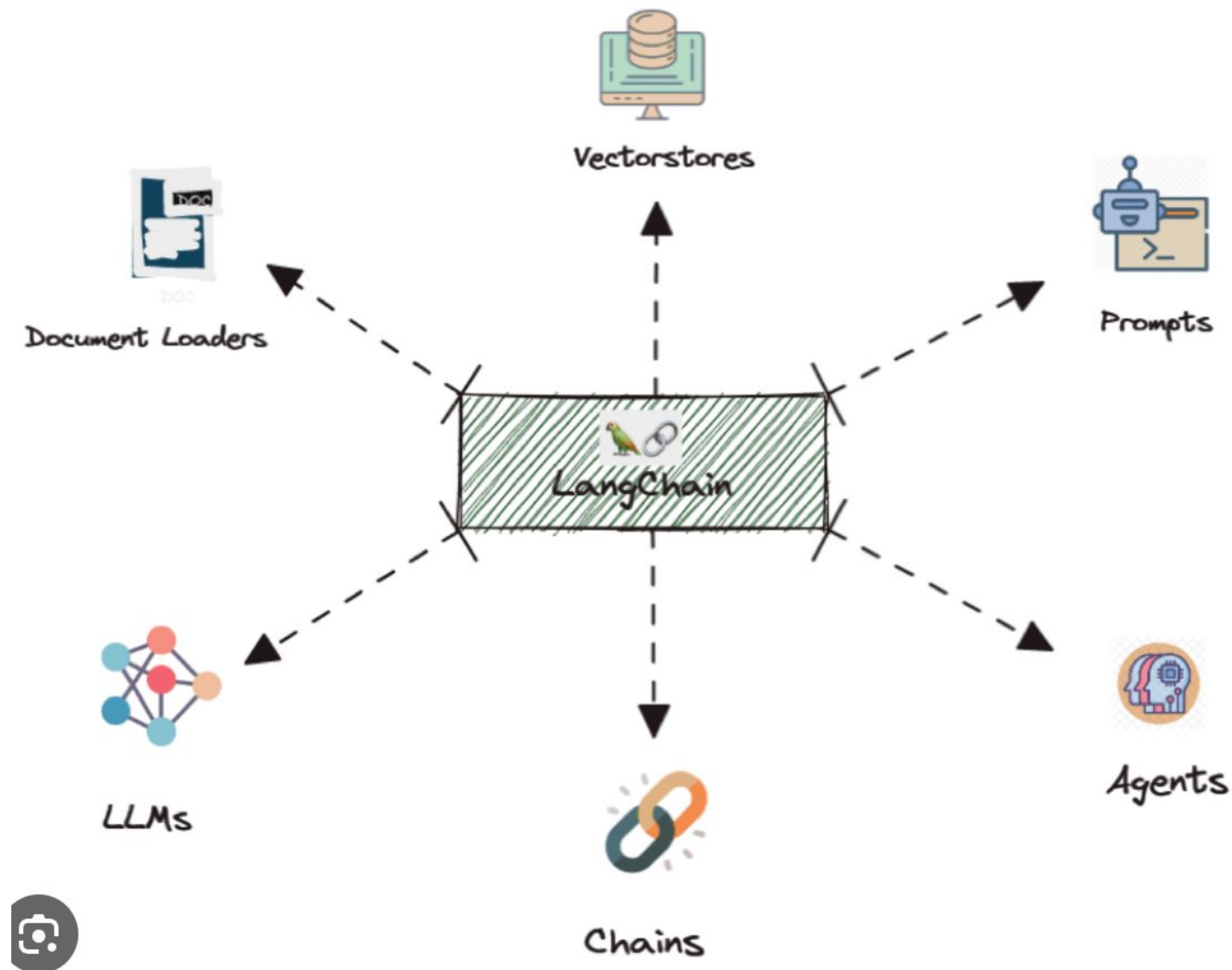
- Document Loaders: As discussed earlier, these tools read and prepare documents, extracting essential information for indexing.
- Vector Stores: These specialized data structures efficiently store the extracted information and document references, enabling fast retrieval based on search queries.
- Retrievers: These tools leverage the index within the vector store to search for documents based on specific criteria, delivering efficient retrieval results.

# Types of Indexing in LangChain:

- Full-text indexing: This captures all words and phrases within a document, allowing for searching based on keywords or full sentences.
- Metadata indexing: This focuses on capturing and indexing specific information embedded within the document (like author, date, or category), enabling searching and filtering based on these attributes.



# Langchain as a whole:



**Thank You!**

PESU

# **WELCOME !!**

## **CodeCraft : Unleashing Langchain & LLMs**

**– Week 2 : Day 3**

**– Nandan Hemanth**



# What is streamlit?

- Streamlit is an open-source Python library that is designed to help developers create web applications for data science and machine learning with minimal effort.
- It simplifies the process of turning data scripts into shareable web apps by providing a simple and intuitive API.
- Streamlit is particularly popular among data scientists and engineers who want to quickly prototype and deploy interactive data applications without delving deep into web development.

# Key Features of Streamlit:

- Simple and Declarative: Streamlit uses a declarative approach where you define the app's layout and functionality using Python code. This makes it easier to create UIs compared to traditional web frameworks.
- Fast Prototyping: With Streamlit, you can quickly prototype data apps and dashboards without needing extensive web development knowledge.

# Key Features of Streamlit:

- Interactive Widgets: It provides various built-in widgets like buttons, sliders, text boxes, and charts for user interaction with your application.
- Sharing and Deployment: Streamlit apps can be easily shared with others by running them locally or deploying them to cloud platforms.



# Use Cases for Streamlit:

- Data Exploration and Visualization: You can create interactive dashboards to explore and visualize datasets. Users can filter, sort, and interact with the data to gain insights.
- Machine Learning Model Deployment: Streamlit helps deploy machine learning models as web applications. Users can input data and see model predictions in a user-friendly interface.

# Benefits of Using Streamlit:

- Reduced Development Time: Streamlit streamlines the UI development process, saving time compared to traditional web frameworks.
- Improved Collaboration: Data scientists and stakeholders can easily interact with data and models through the user-friendly interface.
- Flexibility: Streamlit integrates well with existing Python libraries and tools commonly used in data science.

# Streamlit functions:

- `st.write(any_data)`: The most versatile function, displays various data types directly on your app: text, numbers, HTML, dataframes, plots, and more.
- `st.title(title_text)`: Creates a large heading for your app.
- `st.header(header_text)`: Displays a smaller heading than `st.title`.
- `st.subheader(subheader_text)`: Creates a subheading, even smaller than `st.header`.



# Streamlit functions:

- `st.bar_chart(data)`: Generates a bar chart from a list of lists or NumPy array.
- `st.pyplot(figure)`: Embeds a Matplotlib figure created using pyplot into your Streamlit app.
- `st.video(video_data, format="video/mp4", start_time=0.0, end_time=None)`: Embeds a video file or URL in your app.
- `st.audio(audio_data, format="audio/ogg", start_time=0.0, end_time=None)`: Plays audio content.

# Streamlit functions:

- `st.button(text, on_click=None, key=None)`: Creates a button that triggers a callback function (`on_click`) when clicked.
- `st.radio(label, options, index=0, key=None)`: Generates a radio button group where only one option can be selected at a time.
- `st.slider(label, min_value=None, max_value=None, value=None, key=None)`: Generates a slider widget for selecting a value within a specified range.

# Running Streamlit on localhost:

- Navigate to the current working directory which has the streamlit main file
- Run streamlit on localhost with this command
  - “Streamlit run <name>.py”



**Thank You!**

PESU



PESU

# **WELCOME !!**

## **CodeCraft : Unleashing Langchain & LLMs**

**– Week 3 : Day 1**

**– Nandan Hemanth**



# What is Git and github?

- **Git:** A distributed version control system (DVCS) that tracks changes in computer files over time. It allows you to see who made those changes, when they were made, and revert to previous versions if needed. Think of it as a time machine for your code.
- **GitHub:** A web-based hosting platform specifically designed for Git repositories. It acts like online storage for your code projects managed with Git, enabling collaboration, version control history access, and often used for open-source projects. Imagine it as a social network for code, where you can store, share, and collaborate on projects with others.

# Git: Version Control System

- Imagine yourself writing a novel. With each edit, you might want to save a copy in case you need to revert to a previous version. Git acts like that for software code.
- It's a version control system (VCS) that lets you track changes in computer files and code over time.
- You can:
  - See who made changes and when.
  - Revert to previous versions if needed.
  - Collaborate with others on the same codebase.

# GitHub: Web-Based Git Repository Hosting

- Think of GitHub as a cloud storage specifically designed for Git repositories. It's like a social network for code.
- It allows you to:
  - Store your Git repositories (collections of code and tracked changes) online.
  - Collaborate with others on projects by sharing your code.
  - Access and contribute to code from other developers.
  - Find and share code through open-source projects.



# GitHub: Web-Based Repository Hosting

- Think of GitHub as a cloud storage specifically designed for Git repositories. It's like a social network for code.
- It allows you to:
  - Store your Git repositories (collections of code and tracked changes) online.
  - Collaborate with others on projects by sharing your code.
  - Access and contribute to code from other developers.
  - Find and share code through open-source projects.



# How is Maggi related to Git?





# Git and Github Key differences:

Feature	Git	GitHub
<b>Purpose</b>	Version control system	Web-based Git repository hosting
<b>Offline usage</b>	Yes, works entirely on your local machine	No, requires internet access
<b>Collaboration</b>	Can be used collaboratively, but requires additional setup	Designed specifically for collaboration
<b>Open-source vs. private</b>	Can be used for both, but typically used for private repositories	Can be used for both, but has a strong focus on open-source projects



# How to install Git?

1. Go to the website & download necessary version of Git-  
<https://git-scm.com/downloads>
2. While installing Git, you can keep all the defaults

# Some Git Commands

## 1. Initialization:

**`“git init”`**: This command initializes a new Git repository in the current directory. This creates a hidden folder named `.git` that stores all the version control information.

# Some Git Commands

## 2. Staging Changes:

**`“git add <file>”`**: This command adds a specific file to the staging area. The staging area represents the files you plan to include in your next commit.

**`“git add .”`**: This adds all tracked files (files already added to the Git repository) in the current directory and its subdirectories to the staging area.



## Some Git Commands

### 3. Committing Changes:

`" git commit -m "<message>" "`: This command captures the current state of the staged changes as a new commit. The `<message>` argument is a brief description of the changes you're committing.

## Some Git Commands

### 4. Viewing Changes:

**“git status”**: This command displays the status of your working directory, including which files are modified, staged, or untracked (not yet added to the Git repository).

**“git diff”**: This command shows the difference between the working directory and the index (staging area) or between the index and the HEAD commit (latest commit).

# Setting up SSH Key:

SSH\_Key : An SSH key is a credential used for secure communication in the realm of Secure Shell (SSH) protocol.

1. Open GitBash
2. Paste this - `ssh-keygen -t ed25519 -C "your\_email@example.com"`  
& keep all the default setting by pressing "ENTER"
3. Navigate to `.ssh/` folder
4. Cat `.ssh/id_ed25519.pub` & copy the key



# Adding SSH Key:

SSH\_Key : An SSH key is a credential used for secure communication in the realm of Secure Shell (SSH) protocol.

1. Go to settings in Github.com
2. In the sidebar, select SSH & GPG keys
3. Create new SSH key

# Try it out!

Go to <https://github.com/NandanHemanth/github-workshop>

**\*\* Please star★ the repo if you found it useful\*\***

# Thank You!

PESU



# **WELCOME !!**

## **CodeCraft : Unleashing Langchain & LLMs**

**– Week 3 : Day 3**

**– Nandan Hemanth**

# What is LangServe?

LangServe is a library within the LangChain framework that acts as a bridge between your LangChain applications and the world. It allows you to deploy your LangChain runnables and chains as REST APIs. This means you can interact with your powerful language processing applications through simple API calls.

# REST API:

**APIs (Application Programming Interfaces):** These act as intermediaries between different software applications, allowing them to exchange data and functionality.

- **Standardized communication:** Ensures consistent interaction between different applications, regardless of programming languages or platforms.
- **Stateless communication:** Each request from an application to the API is treated independently, without relying on information from previous requests. This simplifies communication and improves scalability.



# REST API:

- Resource-based: APIs interact with resources (data or functionality) identified by URLs. This makes the interaction intuitive and web-like.

# LangServe Core Functionalities:

- **Transforms Code into APIs:** LangServe takes your LangChain code (runnables and chains) and transforms it into a format that web applications and other systems can understand – a REST API. This allows you to easily integrate your LangChain applications into various projects and workflows.
- **Handles Data Flow:** LangServe manages the flow of data between your application and the LangChain models. It takes user input or data from external sources, feeds it to the LangChain models, and returns the processed results back to the user or application.

# LangServe Core Functionalities:

- **Input and Output Validation:** LangServe utilizes Pydantic (a Python data validation library) to ensure that data sent to your LangChain models is in the correct format. This helps prevent errors and unexpected behavior.
- **Documentation Generation:** LangServe can automatically generate API documentation in formats like OpenAPI (Swagger) to facilitate easy integration and usage of your LangChain applications.



# Benefits of Using LangServe

- **Accessibility:** LangServe makes your LangChain applications accessible to a wider audience. Developers and non-programmers alike can interact with your applications through simple API calls without needing to delve into the complexities of LangChain code.
- **Integration:** LangServe streamlines the integration of your LangChain applications into various web applications, mobile apps, or other systems that can leverage REST APIs.

# Benefits of Using LangServe

- **Maintainability:** By separating the LangChain logic from the API layer, LangServe promotes cleaner code organization and easier maintenance of your applications.
- **Scalability:** LangServe allows you to scale your LangChain applications efficiently by handling API requests and data flow effectively.

# Analogy of LangServe

Imagine building a robot that can answer your questions. LangChain defines the robot's "brain" with the logic to process information and generate responses. LangServe acts as the robot's "external interface." It allows you to send questions (API requests) to the robot and receive answers (API responses) without needing to directly interact with its complex internal workings (LangChain code).



# What is Langsmith?

LangSmith is a comprehensive DevOps platform specifically designed to streamline the development, deployment, and management of applications powered by Large Language Models (LLMs). It acts as a bridge between you and the immense potential of LLMs, simplifying various stages of the development lifecycle.

# Key Functionalities of LangSmith

- **User-Friendly Interface:** Built on top of LangChain, LangSmith provides a visual interface that makes interacting with LLMs more accessible. This is particularly beneficial for users who might not have extensive coding experience.
- **Streamlined Development:** LangSmith offers tools and functionalities to efficiently develop LLM applications. You can create chains (instructions for LLMs), manage data flow, build complex workflows, and debug your applications with ease.

# Key Functionalities of LangSmith

- **Collaboration Features:** LangSmith allows multiple users to work together on LLM projects, facilitating teamwork and efficient development processes. This is crucial for large-scale projects involving multiple developers.
- **Debugging and Testing:** A crucial aspect of LLM application development is ensuring they function as intended. LangSmith provides tools to:
  - **Debug chains:** Identify and fix errors within the logic of your LLM application.
  - **Test application outputs:** Verify if your application produces the expected results for various inputs.



# Key Functionalities of LangSmith

- **Deployment and Monitoring:** Once your LLM application is ready, LangSmith streamlines deployment to various environments and offers monitoring tools to track performance and identify areas for improvement. This allows you to ensure your application runs smoothly in production and identify potential issues proactively.

# Benefits of Using LangSmith

- **Reduced Development Time:** By offering a user-friendly interface, pre-built functionalities, and debugging tools, LangSmith significantly accelerates the development process for LLM applications.
- **Improved Quality and Reliability:** The debugging and testing tools within LangSmith help ensure your LLM applications function as intended, leading to more reliable and robust solutions.

# Benefits of Using LangSmith

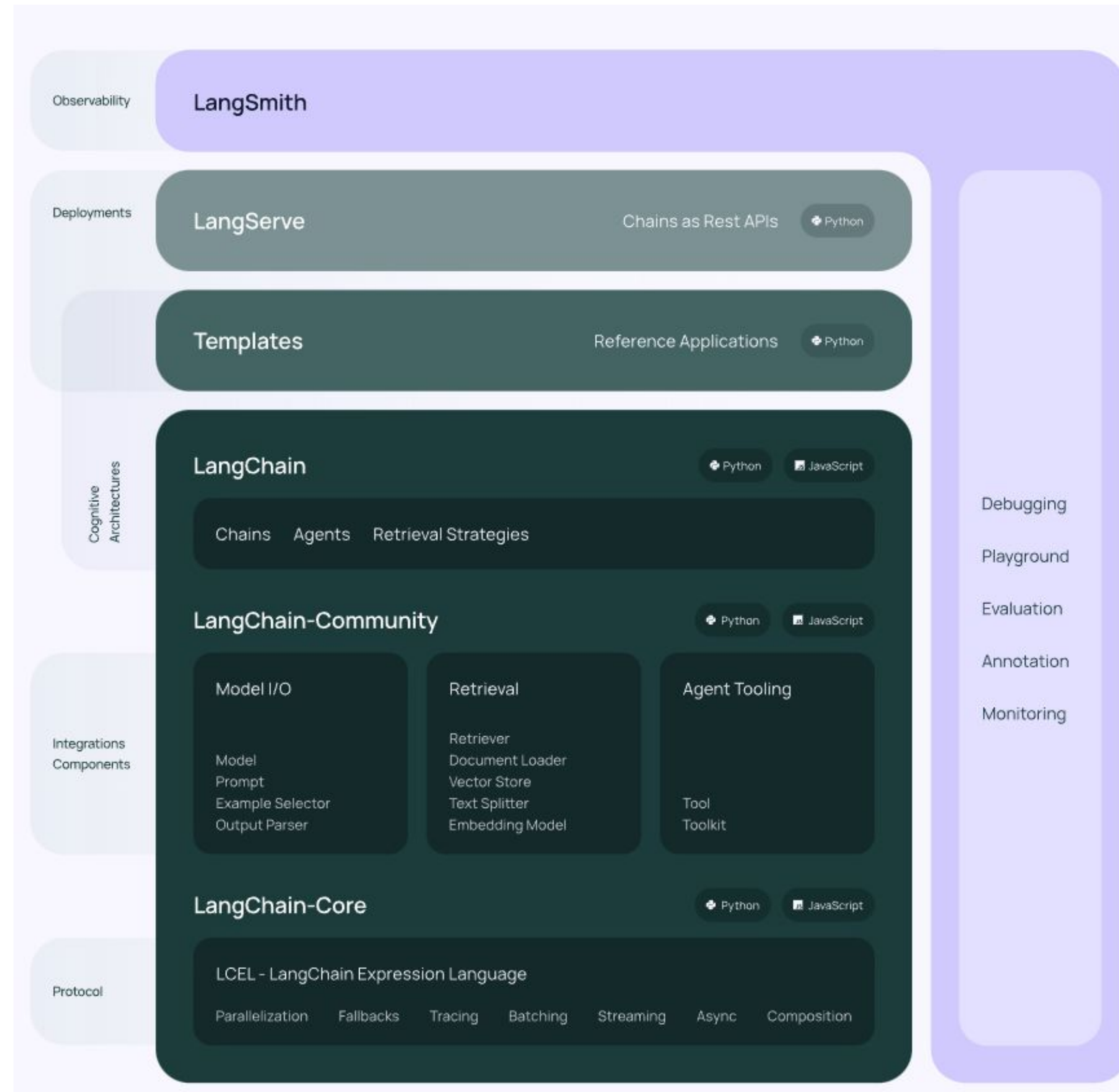
- **Enhanced Collaboration:** Streamlined collaboration features promote teamwork and efficient development workflows, allowing multiple developers to work on projects simultaneously.
- **Simplified Monitoring and Maintenance:** The monitoring tools within LangSmith allow you to keep a close eye on the performance of your LLM applications and identify potential issues proactively, reducing maintenance overhead.



# Analogy of Using LangSmith

Imagine building a house. The bricks and mortar are the raw materials (LLMs), the blueprint is your application logic (chains), and the construction workers are the developers. LangSmith acts as the foreman, providing the tools, managing the workflow, ensuring smooth construction (development) of the house (LLM application), and offering functionalities to monitor its condition (application performance).

# Langchain Environment as a whole!



# Thank You!

PESU



# **WELCOME !!**

## **CodeCraft : Unleashing Langchain & LLMs**

**– Week 3 : Day 4**

**– Nandan Hemanth**

# What are Large Language Models(LLMs)?

LLMs are super-powered AI systems trained on massive amounts of text data.

Imagine a student who has read millions of books – that's kind of like an LLM! This vast exposure allows them to understand and generate human-like text, making them useful for a variety of tasks.



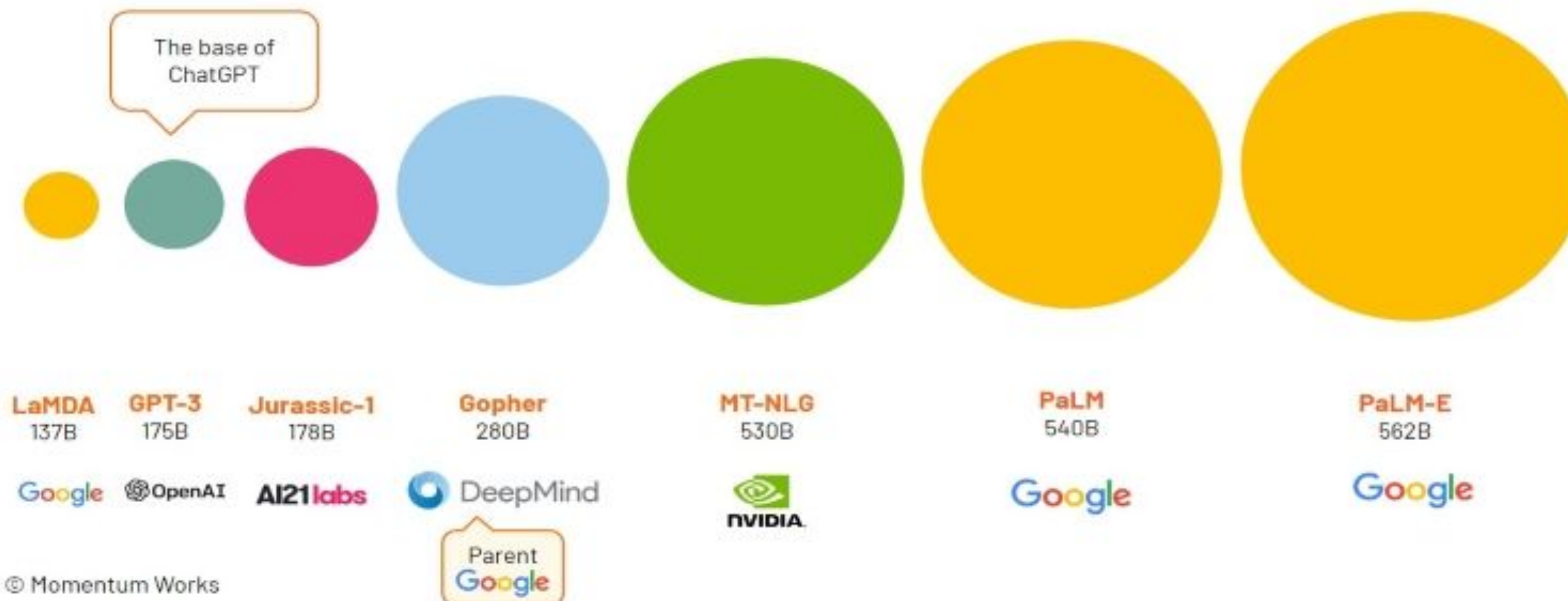
# What are Large Language Models(LLMs)?

Large Language Models are becoming very large indeed

## Small models (<= 100b parameters)



## Large models (>100b parameters)





# Key Capabilities:

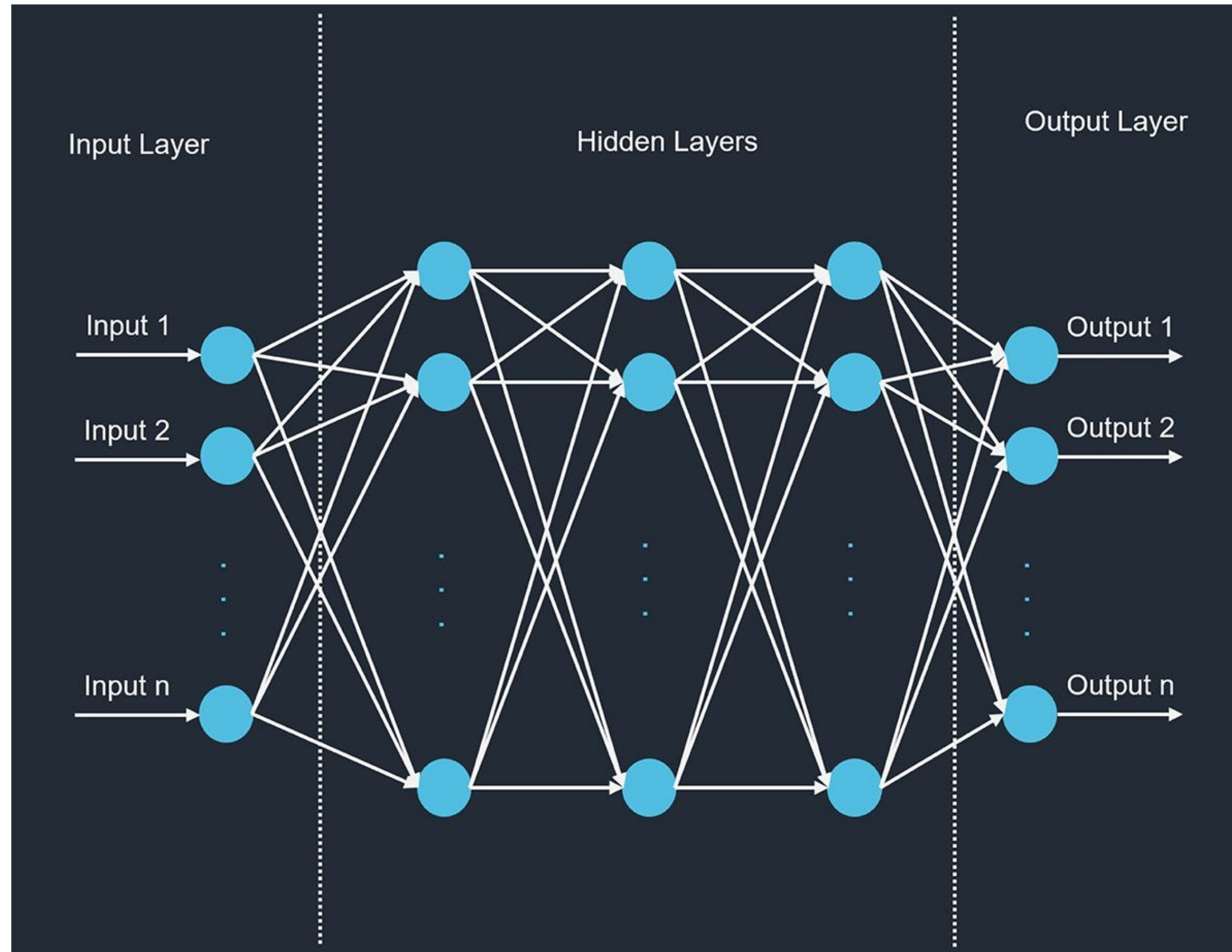
- **Understanding Text:** LLMs can analyze text, identify relationships between words, and grasp the overall meaning. They're like super-smart readers who can untangle complex sentences and concepts.
- **Generating Text:** Based on their understanding, LLMs can create different creative text formats, like poems, code, scripts, or even answer your questions in an informative way. Think of them as super-powered writers who can adapt their style to different needs.

# How do LLMs work?

- Deep Learning: LLMs use complex algorithms called transformers to process and understand text.
- Massive Datasets: They train on enormous amounts of text data, like books, articles, and code, which helps them learn the nuances of language.
- Statistical Predictions: At their core, LLMs predict the next word in a sequence based on the words before it. This ability allows them to generate coherent and relevant text.



# How do LLMs work?





# Benefits of LLMs:

- Automation: LLMs can automate tasks involving text processing, saving time and effort.
- Enhanced User Experience: LLMs create more natural interactions with chatbots and virtual assistants.
- Improved Content Creation: LLMs can assist with brainstorming ideas and generating different creative text formats.
- Data-Driven Insights: LLMs can analyze large amounts of text data to extract insights that might be difficult for humans to identify.

# What are transformers?

- Transformers are a type of neural network architecture that have become the backbone of modern natural language processing (NLP) tasks.
- They excel at understanding the relationships between words in a sentence and have revolutionized tasks like machine translation, text summarization, and question answering.

# What are transformers?

- Unlike traditional recurrent neural networks (RNNs) that process text word by word, transformers can analyze the entire sentence at once. This allows them to capture long-range dependencies between words, crucial for understanding complex language.
- Imagine this: RNNs are like reading a sentence one word at a time, while transformers can see the entire sentence like you do, grasping how words connect even if they're far apart.



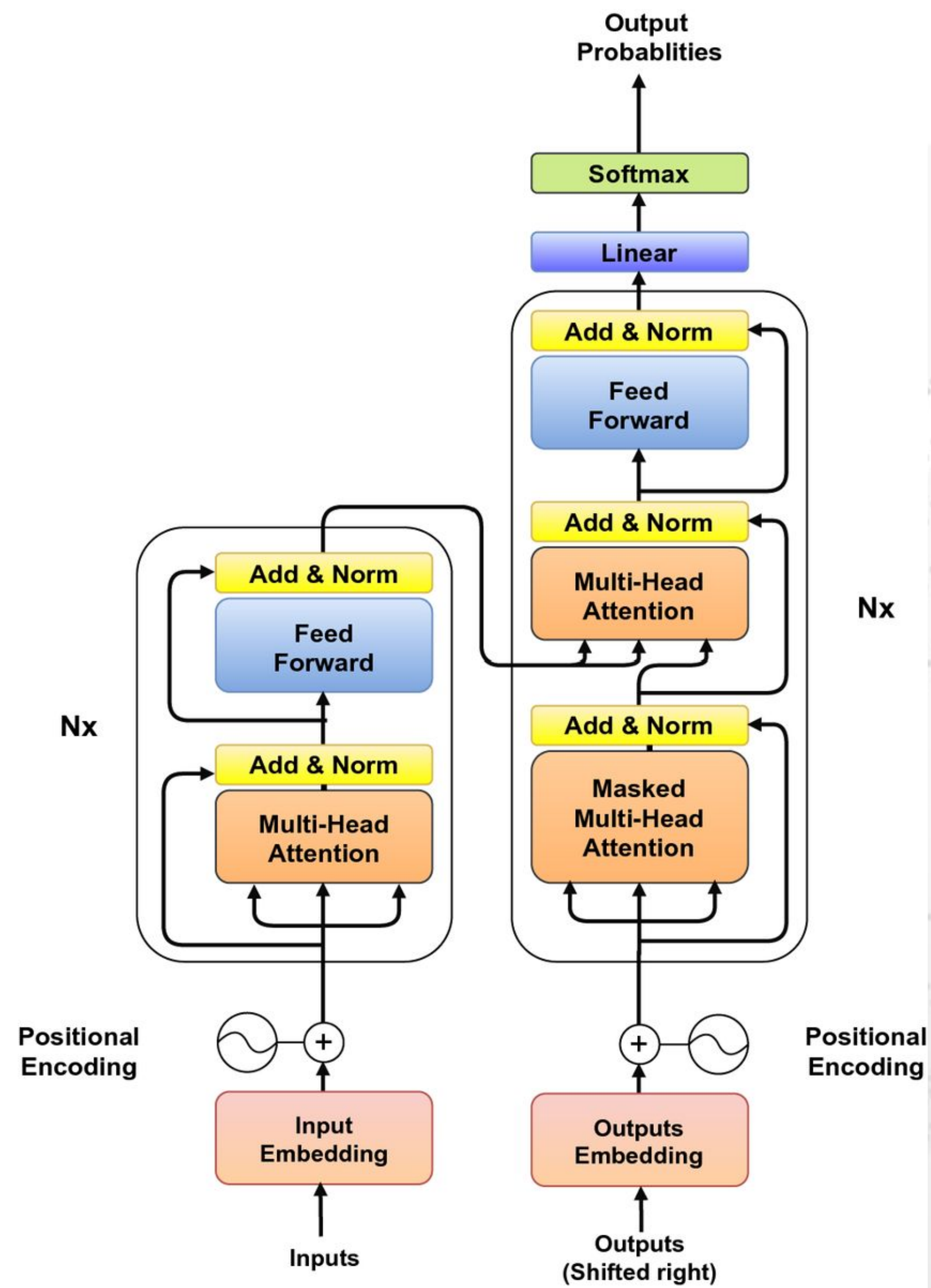
# Internal working & architecture:

- Encoder-Decoder Structure: Most transformers follow this structure:
  - **Encoder**: Takes the input text (sentence) and generates a contextual representation, capturing the meaning and relationships between words.
  - **Decoder**: Uses the encoded representation from the encoder to generate the output text (translation, summary, or answer).

# Internal working & architecture:

- Self-Attention Mechanism: This is the core of a transformer. It allows the model to focus on (attend to) specific parts of the input sentence relevant to the current word being processed. Think of highlighting important words in a sentence as you read it. Self-attention does something similar within the transformer!
- Multi-Head Attention: Transformers often use multiple self-attention layers stacked together. This allows them to learn different aspects of the relationships between words, providing a more nuanced understanding.

# Internal working & architecture:





# Benefits of transformers:

- Parallel Processing: Unlike RNNs, transformers can process entire sentences simultaneously, making them faster to train and use.
- Long-Range Dependency Capture: They excel at understanding how words relate to each other, even if they're far apart in the sentence.
- State-of-the-Art Results: Transformer-based models have achieved remarkable results on various NLP tasks, pushing the boundaries of what's possible with machine learning.

# What is fine-tuning?

- Fine-tuning is a powerful technique in machine learning, particularly for large language models (LLMs) like the ones you might encounter in LangChain. It allows you to take a pre-trained LLM, which has already learned a vast amount of general knowledge from a massive dataset, and specialize it for a specific task.



# What is fine-tuning?

Taking a pre-trained model and training at least one model parameter

## Base Model

(davinci (GPT-3))

**Prompt:** Tell me how to fine-tune a model

**Completion:**

How can I control the complexity of a model?  
How do I know when my model is done?  
How do I test a model?  
How do I define a task for my model?  
How do I scale a model?  
How do I make a probabilistic model?  
How do I start using a model once it's complete?  
How do I cope with missing data?  
How do I make my modeling efforts sustainable?

## Fine-tuned Model

(text-davinci-003 )

**Prompt:** Tell me how to fine-tune a model

**Completion:**

Fine-tuning a model involves adjusting the parameters of a pre-trained model in order to make it better suited for a given task. There are generally three steps to fine-tuning a model:

1. Select a base model - Select a pre-trained deep learning model that has been trained on a large dataset.
2. Adjust parameters - Adjust parameters of the pre-trained model to better suit the desired task. This may include changing the number of layers, adjusting learning rate, adding regularization, or tweaking the optimizer.
3. Train the model - Train the new model on the desired dataset. The amount of data and the amount of training required will depend on the task and the model.



## How it works?

- Pre-Trained LLM: You start with a pre-trained LLM like GPT-3 or Jurassic-1 Jumbo, which has been trained on a massive dataset of text and code. This model has learned general patterns and relationships within language.
- Fine-Tuning Dataset: You provide the LLM with a new, smaller dataset specific to your desired task. This dataset could be labeled examples of text and desired outputs (e.g., question-answer pairs, translation examples, or different creative writing prompts and corresponding creative text formats).

## How it works?

- **Focused Training:** The LLM leverages its pre-trained knowledge as a foundation and fine-tunes its parameters (internal adjustments) based on the new, specific dataset. This allows the model to specialize in understanding the nuances and patterns relevant to your task.

# Benefits of Fine-tuning LLMs:

- Leveraging Pre-Trained Knowledge: You don't need to train a giant LLM from scratch, saving time and computational resources. The pre-trained model acts as a strong starting point.
- Improved Performance on Specific Tasks: Fine-tuning allows the LLM to excel at your particular task by adapting its knowledge to the specific data it encounters.
- Customization for Diverse Applications: You can fine-tune LLMs for various tasks, from writing different creative text formats to generating code or translating languages.



# Different kinds of Fine-tuning:

- Full Fine-Tuning: This involves training all the parameters of the pre-trained LLM on the new dataset. This can be computationally expensive and might lead to overfitting (the model performs well on the fine-tuning data but not on unseen data).
- Adapter Modules: Here, you add a small trainable module (adapter) on top of the pre-trained LLM. This adapter focuses on learning task-specific knowledge while keeping the core LLM parameters mostly unchanged. This is often more efficient and avoids overfitting.

# Different kinds of Fine-tuning:

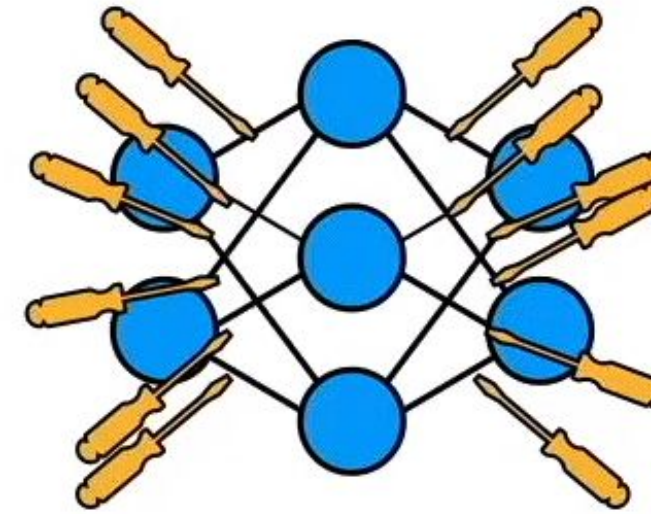
- Low-Rank Adaptation (LoRA): This is a recent technique that utilizes even smaller trainable parameters to achieve good results. It focuses on updating only the most important parts of the pre-trained LLM, making it particularly efficient for memory-constrained environments.



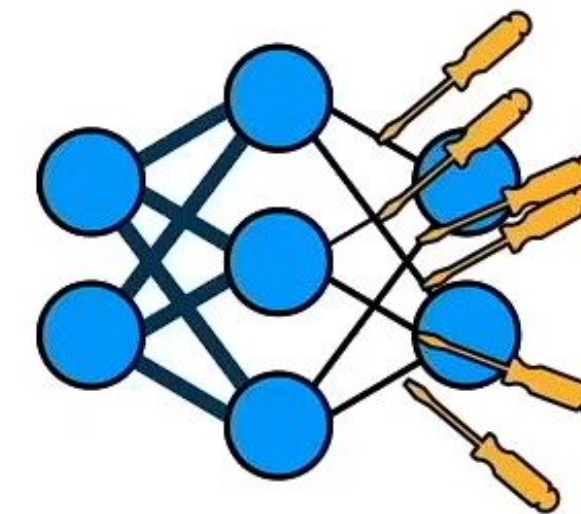
# Fine-tuning methods:

## 3 Options for Parameter Training

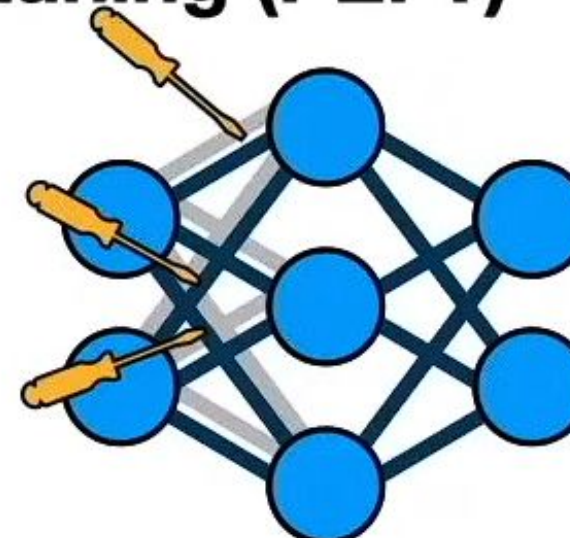
**1) Retrain all parameters**



**2) Transfer Learning**



**3) Parameter Efficient Fine-tuning (PEFT)**





# What is LoRA (Low-Rank Adaptation)?

- LoRA stands for Low-Rank Adaptation. It's a parameter-efficient fine-tuning technique specifically designed for large language models.
- In essence, fine-tuning allows you to harness the power of pre-trained LLMs and tailor them to your specific needs. LoRA offers a particularly efficient way to achieve this, making it a valuable tool for working with large language models.

# What is LoRA (Low-Rank Adaptation)?

- Reduced Memory Footprint: LoRA uses two low-rank matrices to update the pre-trained LLM parameters, significantly reducing memory requirements compared to full fine-tuning. Imagine using a smaller paintbrush for touch-ups instead of repainting the whole canvas.
- Focus on Important Updates: LoRA targets only the most impactful parts of the pre-trained model for adjustments, leading to efficient fine-tuning.
- Competitive Performance: Despite using fewer parameters, LoRA can often achieve performance comparable to full fine-tuning, making it an attractive option for resource-constrained scenarios.

# Thank You!

PESU



PESU