AI For Developers

Prompts Engineering

Yaniv Arad – Nice to e-meet you

Technologies trainer and mentor

On our menu

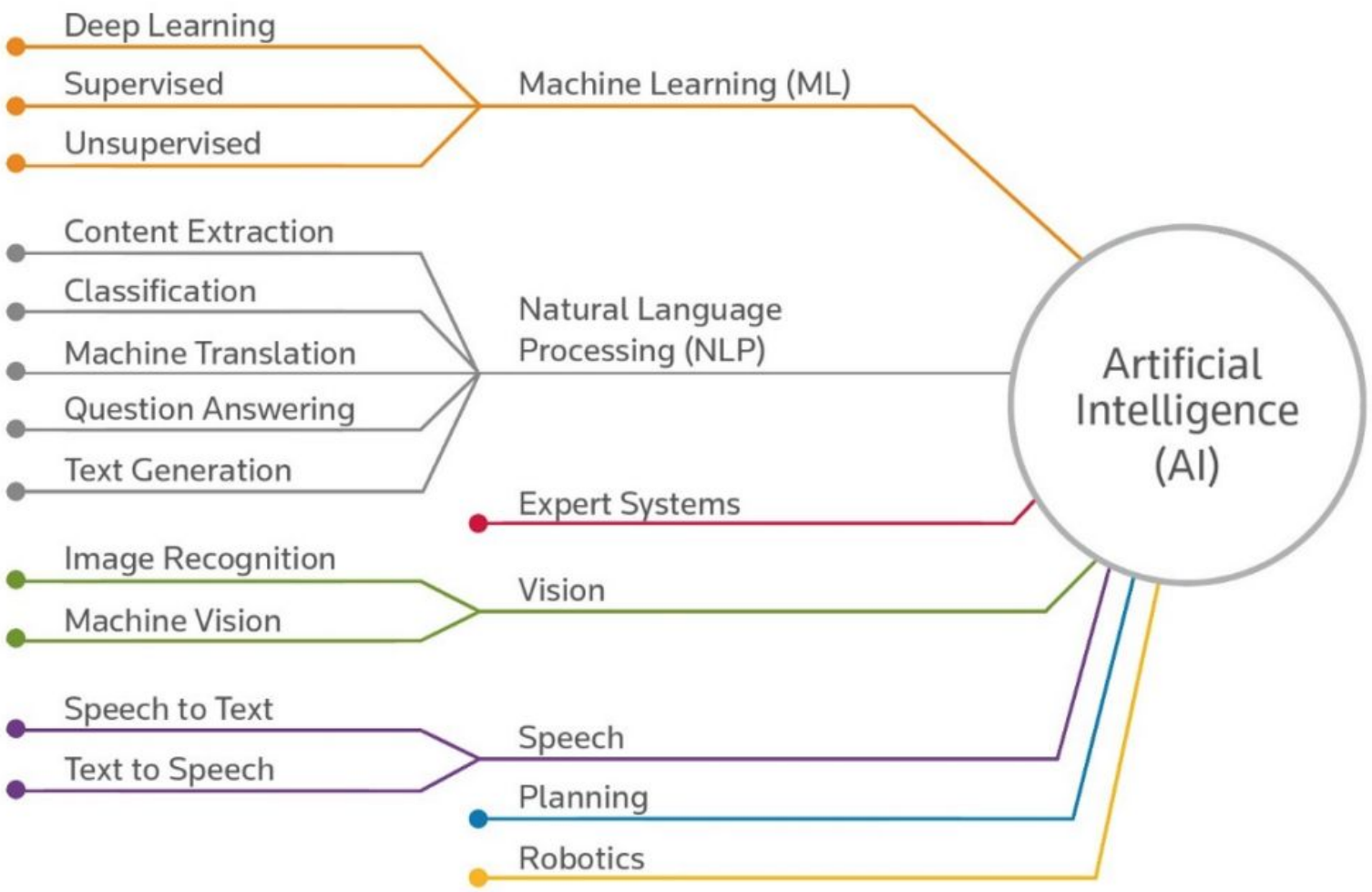
- Intro to AI & Machine Learning
- Generative AI & LLMs Systems
- LLM Parameters
- Prompts Engineering
- Prompts Engineering for Developers
- Integrating LLM API
- LLM for Dev Lifecycle
- LLM Frameworks
- RAG Architecture

Intro to AI and Machine Learning

What is AI ?

Artificial intelligence is the simulation of human intelligence processes by machines, especially computer systems.

AI Applications



Intro to Machine Learning

Machine Learning

The MAIN field in AI !

What is Machine Learning

- Machine learning is the practice of teaching computers how to learn patterns from data, often for making decisions or predictions.

- For true machine learning, the computer must be able to learn patterns **that it's not explicitly programmed to identify**.

EXAMPLE : THE CURIOUS CHILD

- A young child is playing at home... And he sees a **candle**! He cautiously waddles over.
- Out of curiosity, he sticks his hand over the candle flame.
- "Ouch!," he yells, as he yanks his hand back.
- "Hmm... that **red and bright** thing really hurts!"

EXAMPLE : THE CURIOUS CHILD

- Two days later, he's playing in the kitchen... And he sees a **stove-top**! Again, he cautiously waddles over.
- He's curious again, and he's thinking about sticking his hand over it.
- Suddenly, he notices that it's **red and bright**!
- "Ahh..." he thinks to himself, "not today!“
- He remembers that **red and bright** means pain, and he ignores the stove top.

EXAMPLE : THE CURIOUS CHILD

- To be clear, it's only machine learning because the child learned **patterns** from the candle.

- He learned that the pattern of **"red and bright means pain.“**

- On the other hand, if he ignored the stove-top simply *because his parents warned him,* that'd be "explicit programming" instead of machine learning.

MACHINE LEARNING

Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

…It’s all about prediction !

Machine Learning – Use Cases

- Waze
- Email spam detection
- Amazon

And so much more….

YanivArad Training Labs

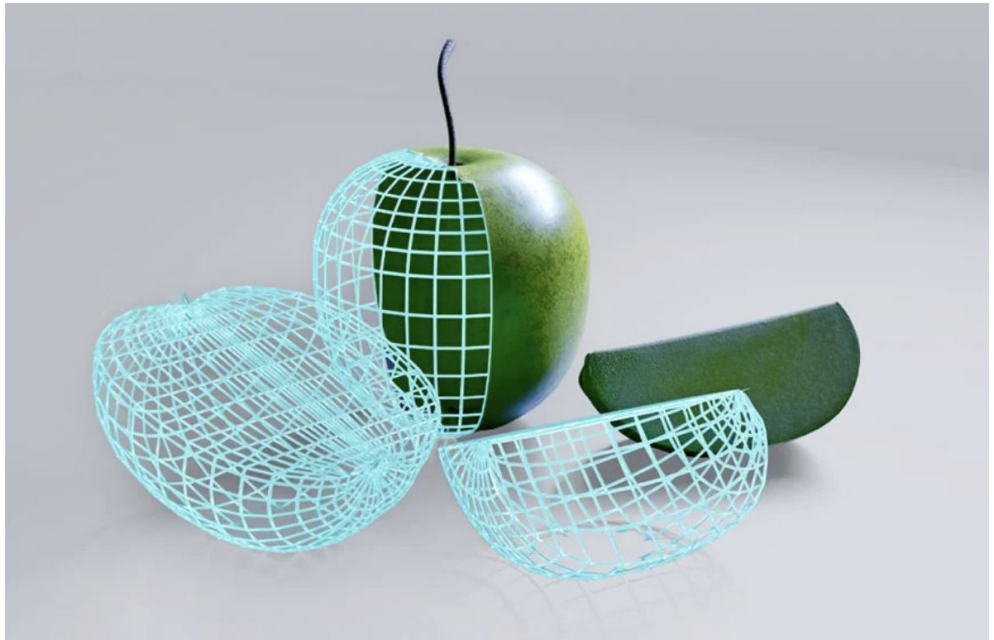
On our menu

- Intro to AI & Machine Learning
- Generative AI & LLMs Systems
- LLM Parameters
- Prompts Engineering
- Prompts Engineering for Developers
- Integrating LLM API
- LLM for Dev Lifecycle
- LLM Frameworks
- RAG Architecture

Generative AI & LLM systems

Generative AI

Generative artificial intelligence (AI) describes algorithms that can be used to create new content, including audio, code, images, text, simulations, and videos



Applications

ANALYTICS

Wand AI
Wand
Wand gives everyone the AI power to solve business problems and create value faster than ever before.
Free

Alteryx
Alterix
The Alteryx Analytics Automation Platform delivers end-to-end automation of analytics, machine learning, and data science processes.

Tellius
Tellius
Get instant ad hoc answers from all your data, dig deeper to easily understand key drivers for business behaviors, and simplify complex analysis with Tellius Decision Intelligence

TEXT

ChatGPT
OpenAI
ChatGPT is a language model, designed to generate human-like text responses.
Free

GPT-4
OpenAI
GPT-4 is the latest iteration of OpenAI's language model capable of generating highly coherent text and and perform complex tasks.

Notion AI
Notion
Access the limitless power of AI, right inside Notion. Work faster. Write better. Think bigger.
Free

IMAGE

Stable Diffusion
Stability AI
Stable Diffusion is a latent text-to-image diffusion model capable of generating photo-realistic images.
Free

Image Creator
Microsoft
Image Creator generates AI images based on text input.
Free

Midjourney
Midjourney
Midjourney is an independent research lab exploring new mediums of thought and expanding the imaginative powers of the human species.

VIDEO

Runway
Runway
Runway is a new kind of creative suite. One where AI is a collaborator and anything you can imagine can be created.
Free

Fliki
Fliki
Create videos from scripts or blog posts using realistic voices in 2 minutes!
Free

Wonder-AI
Wonder
Transform yourself into anything you can think of! Turn into your most favorite characters, represent yourself in fascinating styles, get drawn by world-famous artists.

AUDIO

JukeBox
OpenAI
A neural net that generates music, including rudimentary singing, as raw audio in a variety of genres and artist styles.
Free

MusicLM
Google
MusicLM is a model generating high-fidelity music from text descriptions.
Free

Text to Speech
Microsoft Azure
Neural text to speech is a powerful speech synthesis capability of Azure cognitive services.
Free

3D

Point-E
OpenAI
Point-E is a system for generating 3D point clouds from complex prompts.
Free

Magic3D
NVIDIA
Magic3D is a new text-to-3D content creation tool that creates 3D mesh models with unprecedented quality.

Imagine 3D
Luma AI
An early experiment to prototype and create 3D with text.

Generative AI

Generative AI relies on Machine Learning Algorithms !

In Machine Learning we TRAIN the computers based on large amounts of data

AI allows computers to abstract the underlying patterns related to the input data so that the model can generate or output new content.

Large Language Models (LLM)

What are LLM?

Type of AI that can mimic human intelligence.

They use statistical models to analyze vast amounts of data, learning the patterns and connections between words and phrases.

This allows them to generate new content, such as essays or articles, that are similar in style to a specific author or genre.

LLMs are controlled by billions of <u>parameters</u>

A parameter is something that helps an LLM decide between <u>different</u> answer choices

What LLM does?

A “next word prediction” engine…

Always predict the NEXT word !

What LLM does?

For example, you could type into an LLM prompt window “For lunch today I ate….”

The LLM could come back with a some result.

There’s no 100% right answer, but there is a probability based on the data already ingested in the model.

- Rice (50%)
- Meet (20%)
- Fish (10%)
- Soup (10%)
- Other.. (5%)

For lunch today I ate rice

What LLM does?

"For lunch today I ate rice and I drank…"

From all trained data, those who ate rice for lunch also drank:

- Water (60%)
- Coca Cola (20%)
- Beer (10%)
- Wine (5%)
- Other.. (5%)

For lunch today I ate rice and I drank water

How LLM Works?

The efficacy of an LLM comes largely from its training process. During this phase:

Pre-training: The model is exposed to massive amounts of text data (such as books, articles, or web pages), so that it can learn the patterns and connections between words. The more data it is trained on, the better it will be at generating new content. While doing so, it learns to predict the next word in a sentence.

Fine-tuning: After the initial pre-training, the model can be refined on a narrower dataset to specialize in specific tasks or knowledge areas. This helps in aligning the model's outputs with desired outcomes.

How can LLM be used?

Once the large language model has been trained, it can be used to generate new content based on the parameters set by the user:

Completion: Given a partial sentence, it can predict the next words.

Question answering: You ask a question, and it generates a relevant answer.

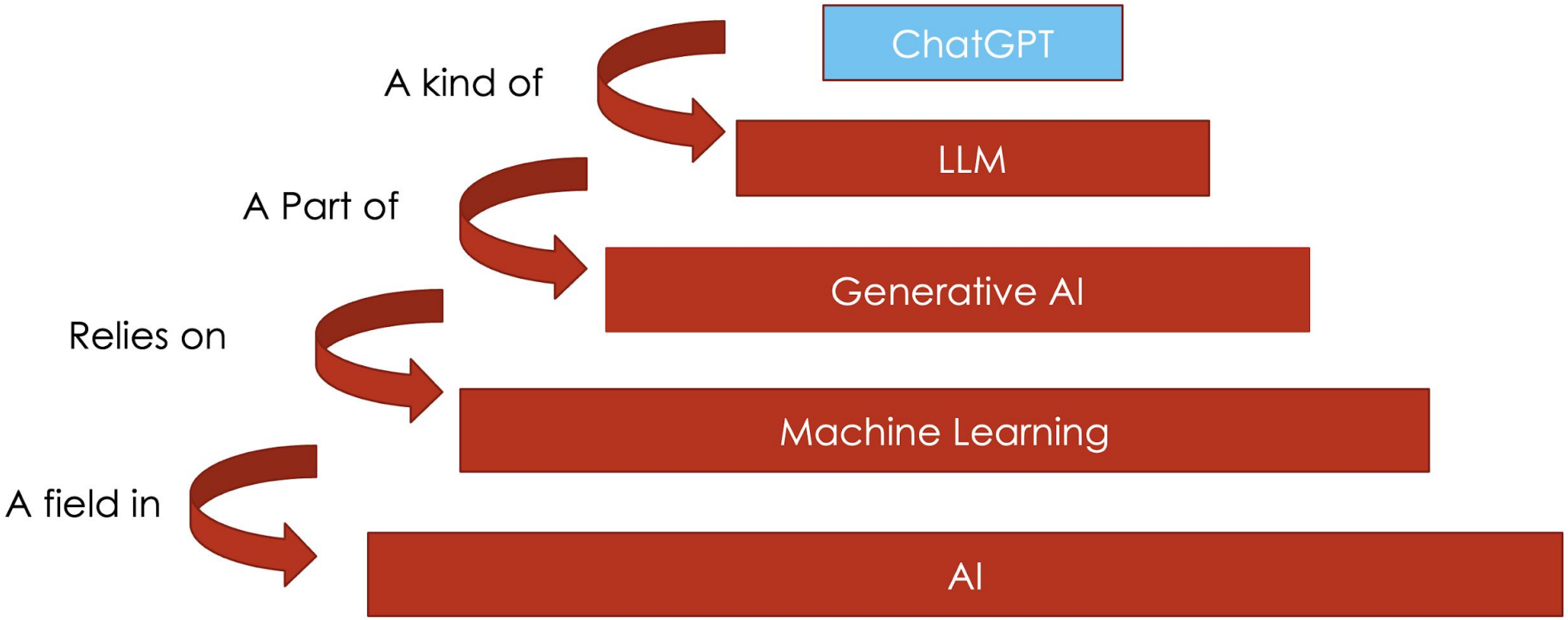
Translation, summarization, and more: The model can be adapted for various text generation tasks.

More on LLM – later…

Famous LLM Engines

ChatGPT (Open AI)
LaMDA, PaLM (Google)
BLOOM (Facebook)
NeMO (Nvidia)

ChatGPT


ChatGPT
A kind of
LLM
A Part of
Generative AI
Relies on
Machine Learning
A field in
AI

YanivArad Training Labs

On our menu

- Intro to AI & Machine Learning
- Generative AI & LLMs Systems
- LLM Parameters
- Prompts Engineering
- Prompts Engineering for Developers
- Integrating LLM API
- LLM for Dev Lifecycle
- LLM Frameworks
- RAG Architecture

LLM Models

LLM Parameters

Temperature

Set the creativity of the response !

A higher temperature value makes the AI more adventurous and creative, and could lead to some unexpected responses!

A lower temperature makes LLM more conservative and focused, sticking to safer and more predictable answers.

Temperature

The sky is...

Blue
Cloudy
Full of stars
Dark
Starry
Beautiful
Falling
...

← Temp = 0.0

← Temp = 0.5

← Temp = 1.0

Max Tokens

Sets the upper limit for the total number of tokens, encompassing both the input provided to the LLM as a prompt and the output tokens generated by .the LLM in response to that prompt

MD Set max_length = 10

MD Give me the history of China

Ancient, dynasties, Great Wall, modern China.

MD Set max_length = 2048 To this...

As an AI language model, I cannot directly set parameters like max_length. However, I can attempt to provide a more in-depth response to your request about the history of China. Here's an overview of China's history, covering some key events and periods:

1. **Prehistoric China**: The earliest evidence of human settlement in China dates back to around 1.7 million years ago. The Xia dynasty (circa 2070-1600 BCE) is considered the first dynasty in China's history, although its existence remains debated due to a lack of archaeological evidence.
2. **Shang Dynasty (circa 1600-1046 BCE)**: The Shang dynasty is the first confirmed dynas with concrete archaeological evidence. It's known for its advanced bronze work, oracle

Top P

Top_p sampling is an alternative to temperature sampling.

Instead of considering all possible tokens,considers only a subset of tokens (the nucleus) whose cumulative probability mass adds up to a certain threshold (top_p).

For example, if top_p is set to 0.1, The ILM will consider only the tokens that make up the top 10% of the probability mass for the next token.

This allows for dynamic vocabulary selection based on context.

Top P

The sky is...

Lower Top-P: model picks from the red zone

Higher Top-P: model picks from the blue zone

Presence Penalty

The Presence Penalty parameter prevents the model from repeating a word, even if it's only been used once.

It basically tells the model, "You've already used that word once — try something else."

- Presence penalty = 0.0: The best sport is **soccer, soccer, soccer.** This is a very repetitive and emphatic text, but also very boring and monotonous.
- Presence penalty = 0.5: The best sport is **soccer, basketball, and tennis.** This is a more varied and novel text, but still somewhat coherent and consistent.
- Presence penalty = 1.0: The best sport is **soccer, rugby, and chess.** This is a very varied and novel text, but also very diverse and unpredictable.

Frequency Penalty

The frequency penalty applies a penalty on the next token proportional to how many times that token already appeared in the response and prompt.

The higher the frequency penalty, the less likely a word will appear again.

This setting reduces the repetition of words in the model's response by giving tokens that appear more a higher penalty.

Frequency Penalty

- Frequency penalty = 0.0: The best sport is soccer, **soccer is fun, and soccer is exciting.** This is a very repetitive and emphatic text, but also very boring and monotonous.
- Frequency penalty = 0.5: The best sport is soccer, **soccer is fun and exciting.** This is a more varied and novel text with less repetitions, but still somewhat coherent and consistent.
- Frequency penalty = 1.0: The best sport is soccer, **football is fun and exciting.** This is a very varied and novel text, but also very diverse and unpredictable, notice that it didn't repeat the word "soccer" present in the prompt.

Frequency vs Presence Penalty

Frequency Penalty helps us avoid using the same words too often. It's like telling the computer, "Hey, don't repeat the same words too much."

Presence Penalty, on the other hand, encourages using different words. It's like saying, "Hey, use a variety of words, not just the same ones."

So, Frequency Penalty stops repetition, while Presence Penalty encourages variety.

the presence_penalty penalizes for having at least one instance, while the frequence_penalty penalizes for each occurrence

Logits

Scores assigned to different words by the model.

These scores are not directly probabilities, but they serve as a basis for calculating probabilities.

if the word “apple” has a logits score of 100 and “orange” has a score of 20, most probably “apple” will come next in the sentence generated

Frequency Penalty

imagine you're using a language model to generate text, and you've set a frequency penalty value of 0.2.

Here's how it works with a sample word:

Let's say a model generated "apple," and by default, it's logit score was 100.

Now if the model generates an apple again (i.e., frequency is **2**), it's score will go down like this.

2 * frequency penalty = 2 * 0.2 = 0.4.
new score = 100-0.4 = **99.6**

Frequency Penalty

If the model generates apple again (i.e., the frequency is **3** now), it‘s score will further go down:

2 * frequency penalty = 3* 0.2 = 0.6.
new score = 100– 0.4 = **99.4**

See how the frequency penalty is a factor here? If the frequency penalty was 0.8, the scores would be 98.4 and 97.6 in each occurrence. It’s remarkable to see how quickly the probability decreases.!

Penalty Penalty

Let’s assume you’ve set a presence penalty of 0.2.

Again, we’ll use the example of the word “apple”.

Let’s say a model generated “apple,” and by default, it’s logit score was 100.

Now if the model generates an apple again, the new score would be 100–0.2 = **99.8**

If the model wants apple again, it’s new score will still be **99.8**, since it will ignore the frequency.

This decrease in probability encourages the model to favour tokens that haven’t been used frequently in the generated text, promoting diversity.

YanivArad Training Labs

On our menu

- Intro to AI & Machine Learning
- Generative AI & LLMs Systems
- LLM Parameters
- Prompts Engineering
- Prompts Engineering for Developers
- Integrating LLM API
- LLM for Dev Lifecycle
- LLM Frameworks
- RAG Architecture

Perplexity AI vs. ChatGPT vs. Claude AI Comparison

Feature	ChatGPT	Claude AI	Perplexity
LLM Used	ChatGPT uses its original LLMs. ChatGPT uses GPT 3.5, which is the free version, while GPT 4 is the paid version. Both versions are still available.	Claude uses its version, Claude 2.1, which is the latest and its only version at the moment.	Perplexity allows access to multiple LLMs and does not have its own.
Accessibility and Cost	Available in most countries worldwide, although banned in China, North Korea, Iran, Cuba, Syria, and Nepal. Free and premium versions available (ChatGPT Plus for a monthly fee).	Mainly available in English-speaking countries: US, UK, Canada, Australia, Ireland, and New Zealand. Free account with access to the latest LLM version; Claude Pro costs $20 monthly.	Present in most countries except Europe. The free version provides access to GPT 3.5 and an autopilot search feature. Perplexity Pro offers access to GPT-4 and Claude for a monthly fee.
Conversational Skills	Provides longer answers with a lot of bullet points. Language is very communicative and natural.	Provides shorter but natural and communicative answers similar to ChatGPT.	Provides natural and conversational language with helpful citations for bullet points, aiding in verification and more research.
Accuracy of Data	Accuracy has reportedly worsened over time, which can be a big issue.	Similar concerns about accuracy as ChatGPT.	Offers more accuracy by providing sources for its answers, which is helpful for verification and research.
Creative Writing	Designed for creative writing and scripting, as it uses its own LLMs.	Designed for creative writing and scripting, as it uses its own LLMs.	Not designed for creative writing, as it depends on multiple LLMs.

YanivArad Training Labs

LLM Models

LLM Models

Penalty Penalty

Let’s assume you’ve set a presence penalty of 0.2.

Again, we’ll use the example of the word “apple”.

Let’s say a model generated “apple,” and by default, it’s logit score was 100.

Now if the model generates an apple again, the new score would be 100–0.2 = **99.8**

If the model wants apple again, it’s new score will still be **99.8**, since it will ignore the frequency.

This decrease in probability encourages the model to favour tokens that haven’t been used frequently in the generated text, promoting diversity.

Prompts Engineering

Prompts Engineering

Prompt engineering is the most crucial aspect of utilizing LLMs effectively

It involves crafting clear and specific instructions or queries to elicit the desired responses from the language model

Prompt Wording

A prompt's wording is paramount, as it guides the LLM in generating the desired output. It's important to phrase the question or statement in a way that LLM understands and can respond to accurately.

Prompt Wording

Lets compare this…

> *Write 10 recipe blogs.*

To this !!

> *I would like you to generate 10 quick-prep dinner meal ideas for recipe blogs, with each idea including a title and a one sentence description of the meal. These blogs will be written for an audience of parents looking for easy-to-prepare family meals. Output the results as a bulleted list.*

Use Delimiters

Using delimiters in prompts can be beneficial in situations where you want to structure your input in a way that is clear and easily parsed by an LLM rely on textual context.

It helps the model to differentiate the actual input versus the prompt/instruction, avoiding mistaking the input content as an instruction, or vice versa.

```
Name;RollNumber;Grade
Aman;21;C
Rahul;22;A
Vishal;12;B
Jyoti;55;A
Swati;27;B
Kishan;23;D
```

Specify Steps

If you're asking the model to solve a problem, break it down step by step.

Like instructing someone on assembling furniture — the clearer the steps, the better the outcome.

Roles & Goals

Roles are personas assigned for the LLM and the intended audience.

Goals are intimately connected to roles. Explicitly stating the goal of a prompt-guided interaction is not only a good idea but also necessary.

Without it, how would LLM have any linking of what output to generate?

Roles & Goals

> *You are to act as a real estate agent with 10 years experience in the Phoenix area. Your goal is to produce a one paragraph summary of each of the top 5 family neighborhoods in the Phoenix metropolitan area. The intended audience is inexperienced home buyers.*
> *Along with the explicitly stated roles and goals, note the relative specificity of the example prompt above.*

Roles & Goals

> *You are to act as a real estate agent with 10 years experience in the Phoenix area. Your goal is to produce a one paragraph summary of each of the top 5 family neighborhoods in the Phoenix metropolitan area. The intended audience is inexperienced home buyers.*
> *Along with the explicitly stated roles and goals, note the relative specificity of the example prompt above.*

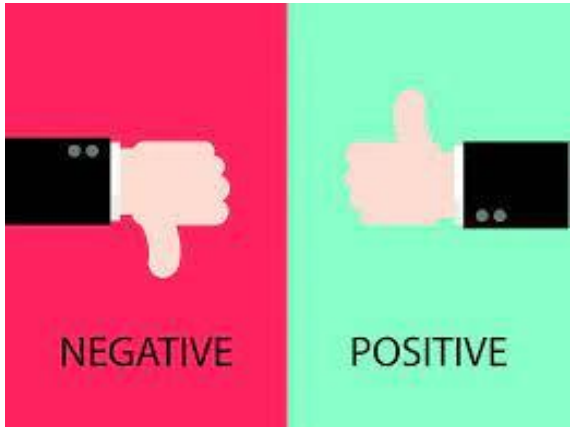
The LLM will take into account the following characteristics:

- Recommendations by agents with <u>10 years experience</u>
- <u>Family</u> neighborhoods in Phoenix
- For <u>inexperienced</u> home buyers

Positive & Negative Prompting

Positive prompts ("*do this*") encourage the model to include specific types of output and generate certain types of responses.

Negative prompts ("*don't do this*"), on the other hand, discourage the model from including specific types of output and generating certain types of responses



Prompts Best Practices - Summary

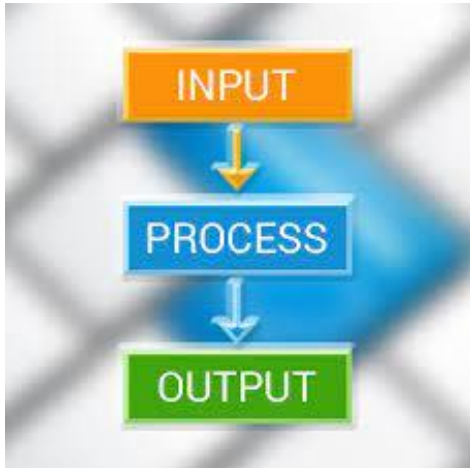
- Wording prompting
- Using delimiters
- Specifying steps
- Roles & Goals
- Negative/Positive

ChatGPT Prompts Strategies

Input/Output Prompting

The input/output prompting strategy involves defining the input that the user provides to the LLM and the output that the LLM is to generate in response.

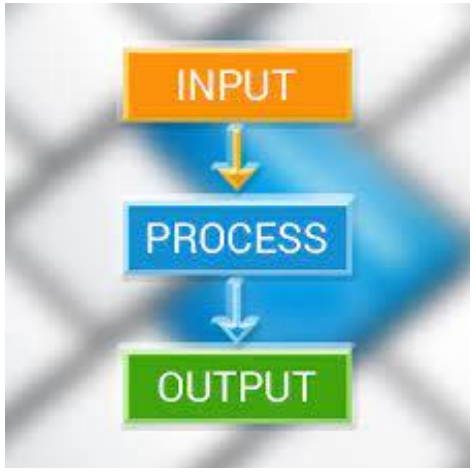
This strategy is fundamental to prompt engineering as it directly influences the quality and relevance of the LLM response.



Input/Output Prompting

Below is an example of the most basic strategy: provide a single input and desire a single output.

> *Generate a Python script that takes a single mandatory command line argument ([project]) and performs the following tasks:*
> *– creates a new folder named [project]*
> *– creates a file within the new folder named [project].py*
> *– writes a simple Python script file header to the [project].py file*



X-Shot Prompting

X-Shot called behalf the numbers of example with give the LLM to rely on

Zero-Shot
One-Shot
Few-Shot

Zero-Shot Prompting

The zero-shot strategy involves the LLM generating an answer without any examples or context.

This strategy can be useful when the user wants a quick answer without providing additional detail, or when the topic is so general that examples would artificially limit the response

One-Shot Prompting

The one-shot strategy involves the LLM generating an answer based on a single example or piece of context provided by the user.

This strategy can guide LLM response and ensure it aligns with the user’s intent.

Few-Shot Prompting

The few-shot strategy involves the LLM generating an answer based on a few examples or pieces of context provided by the user.

This strategy can guide LLM response and ensure it aligns with the user’s intent.

Self Criticism

The self-criticism strategy involves prompting the LLM to assess its output for potential inaccuracies or improvement areas.

This strategy can ensure the information provided by LLM is as accurate as possible.

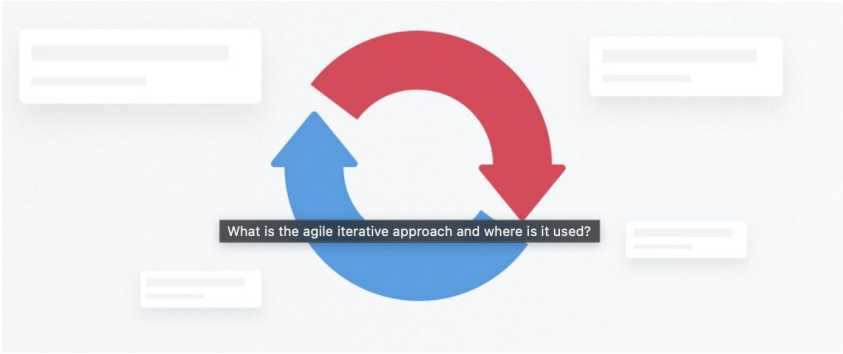
It can aid users in debugging their prompts and determining why they are not getting results that meet expectations.

Iterative

The iterative or expansive strategy involves prompting the LLM with follow-up prompts based on the output of an initial prompt.

This involves iterating on the results by asking further questions or making additional requests from each successive response.

- **Up-Front Iterations**
- **On-Going Iterations**



Chain-of-Thought Prompting

Chain of Thought Prompting is a way to ask better questions (and get better results) with AI.

it’s a systematic approach to getting more nuanced, coherent, and accurate responses from generative AI models.

Think of it as guiding the AI through a topic or question, ensuring it understands what you’re trying to do and helping it process information.

Chain-of-Thought Prompting

It is differentiated from the X-shot prompting techniques as chain-of-thought prompts are structured to encourage critical thinking, and are designed to help uncover new insights or approaches that LLM may not have otherwise considered.

Chain-of-Thought Prompting

JO what's the 4th word in the phrase, I am not what I am?

The 4th word in the phrase "I am not what I am" is "not."

JO what's the 4th word in the phrase, I am not what I am?

think step by step

Of course, let's break down the phrase "I am not what I am" step by step:

1. "I" (1st word)
2. "am" (2nd word)
3. "not" (3rd word)
4. "what" (4th word)

So, the 4th word in the phrase "I am not what I am" is "what."

Chain-of-Thought Prompting

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain-of-Thought Prompting

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

Prompt Strategies - Summary

- Input-Output
- Zero-Shot
- One-Shot
- Few-Shots
- Self-Criticism
- Iterative
- Chain-Of-Thought
- Tree-Of-Thought

YanivArad Training Labs

On our menu

- Intro to AI & Machine Learning
- Generative AI & LLMs Systems
- LLM Parameters
- Prompts Engineering
- Prompts Engineering for Developers
- Integrating LLM API
- LLM for Dev Lifecycle
- LLM Frameworks
- RAG Architecture

Prompts Engineering for developers

YanivArad Training Labs

On our menu

- Intro to AI & Machine Learning
- Generative AI & LLMs Systems
- LLM Parameters
- Prompts Engineering
- Prompts Engineering for Developers
- Integrating LLM API
- LLM for Dev Lifecycle
- LLM Frameworks
- RAG Architecture

Integrating LLM API

OpenAI API Messages

```
import openai

response = openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "You are cricket expert."},
    {"role": "user", "content": "Who won the world cup in 2019?"},
    {"role": "assistant", "content": "England cricket team won the world cup in 2019."},
    {"role": "user", "content": "Who was the opponent?"}
  ]
)
```

3 types of roles

- **System** - Although passing a message with a system role is not required, it is useful to set up the model behavior for conversation

- **User** - The user role indicates that it is an end-user input prompt or a prompt triggered by an application.esults

- **Assistant -** This role indicates that the message was a response from the assistant (model). This role comes in handy when you need to save the previous conversation between the user and the assistant in your application and send it again in a prompt request to maintain continuity. Alternatively, the developer can create the assistant message manually in order to influence a desired behavior in the assistant's subsequent responses.

Response Format


```
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "message": {
        "content": "The opponent was New Zealand in the 2019 Cricket World Cup final. England defea
        "role": "assistant"
      }
    }
  ],
  "created": 1677823794,
  "id": "chatcmpl-6psu2Yl4ZFP5wb1S2CK9bCu7r2fEz",
  "model": "gpt-3.5-turbo-0301",
  "object": "chat.completion",
  "usage": {
    "completion_tokens": 29,
    "prompt_tokens": 54,
    "total_tokens": 83
  }
}
```

YanivArad Training Labs

On our menu

- Intro to AI & Machine Learning
- Generative AI & LLMs Systems
- LLM Parameters
- Prompts Engineering
- Prompts Engineering for Developers
- Integrating LLM API
- LLM for Dev Lifecycle
- LLM Frameworks
- RAG Architecture

LLM for Dev Lifecycle

Dev lifecycle tools

Code Generators

Code Analysis

Unit Tests

Refactoring

Documentation

….

YanivArad Training Labs

On our menu

- Intro to AI & Machine Learning
- Generative AI & LLMs Systems
- LLM Parameters
- Prompts Engineering
- Prompts Engineering for Developers
- Integrating LLM API
- LLM for Dev Lifecycle
- LLM Frameworks
- RAG Architecture

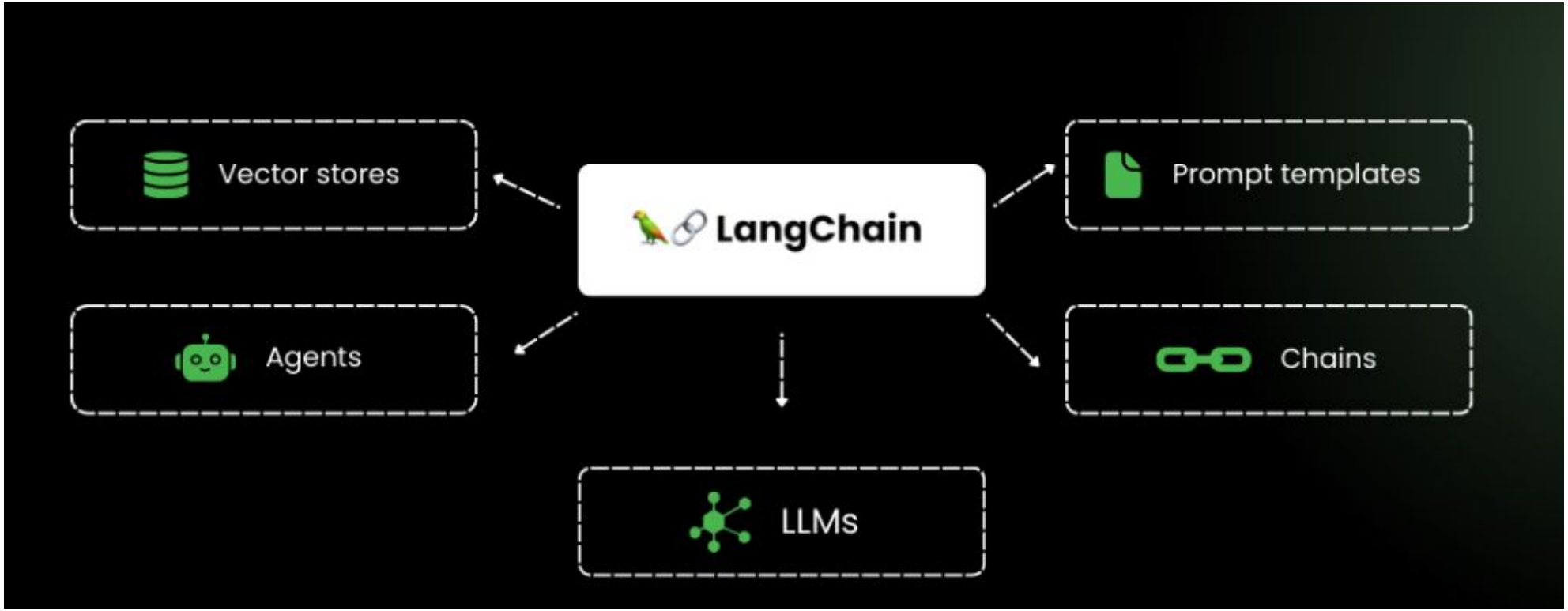
LLM Frameworks

LangChain

LangChain

LangChain is an open-source framework that gives developers the tools they need to create applications using large language models (LLMs).

LangChain Components

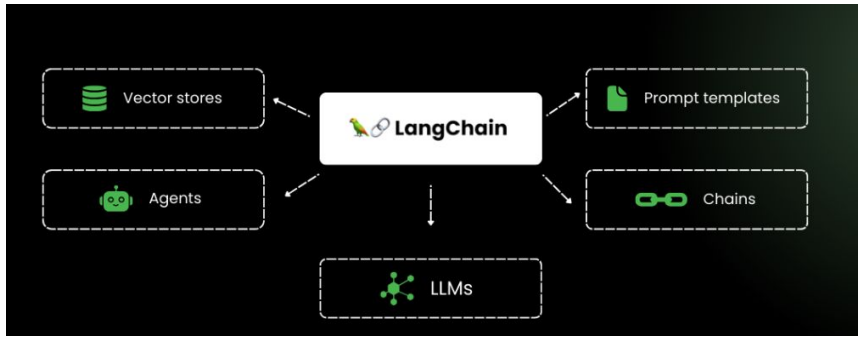


LLMs

Language Models (LLMs) are the core engines behind LangChain applications.

They are responsible for generating human-like text based on the input prompts.

LangChain supports various LLMs, including OpenAI's GPT models, Hugging Face models, and more.

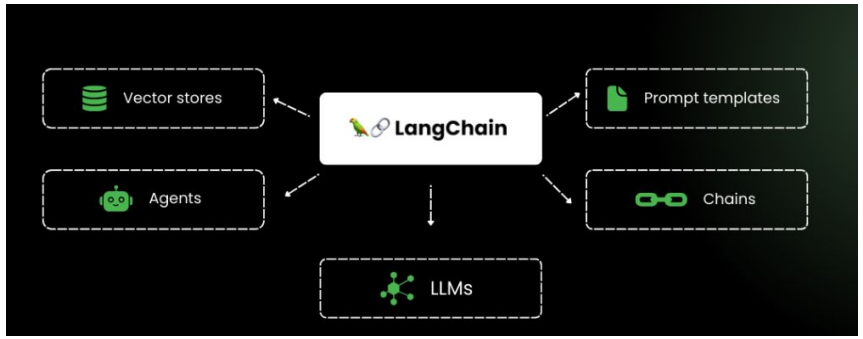


Prompts

Prompts are the input text you provide to an LLM to generate a response.

They are crucial in guiding the LLM's output and defining the task.

LangChain offers a variety of prompt templates and utilities to help you create effective prompts.

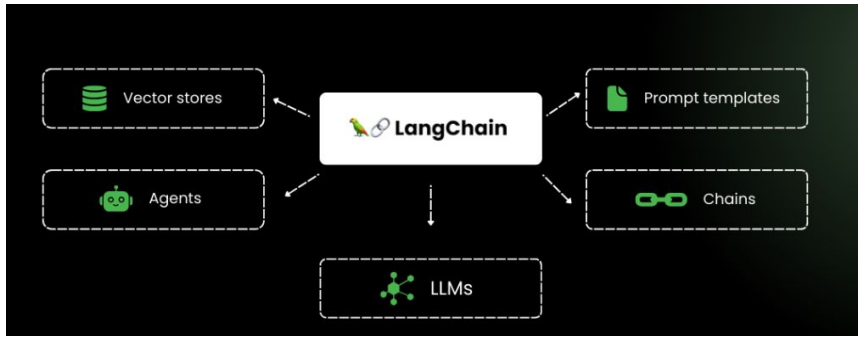


Chains

Chains allow you to combine multiple components, such as prompts and LLMs, to create more complex applications.

They define a sequence of steps to process input, generate output, and perform additional tasks.

LangChain provides a variety of built-in chains and supports the creation of custom chains.

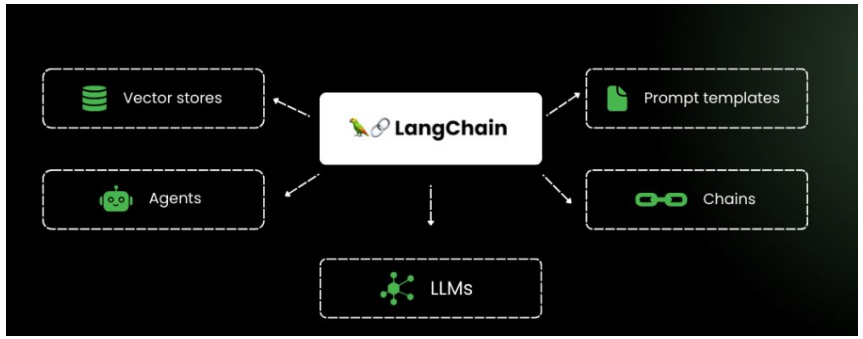


Agents

Agents are high-level abstractions that use chains and tools to accomplish specific goals.

They can make decisions, interact with external tools, and retrieve information to complete tasks.

Agents are particularly useful for building conversational AI applications or automating complex workflows.

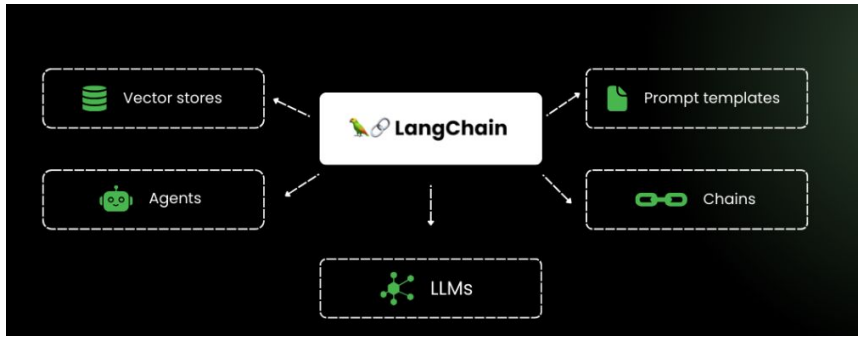


Vector Stores

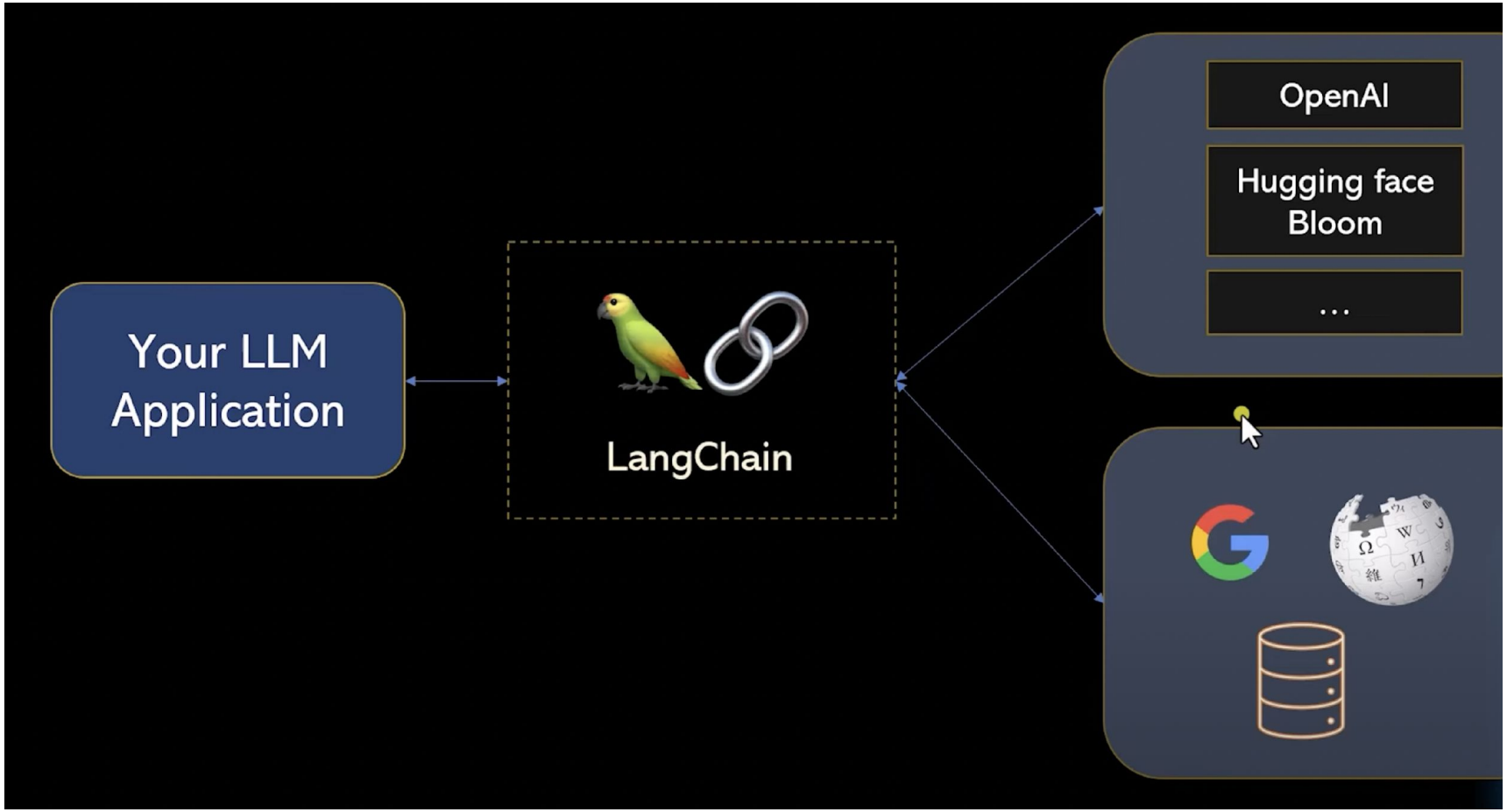
vector stores are databases that store vectors.

Vectors are mathematical representations of words or phrases.

When stored within a vector store, it’s much easier to find similar words or phrases, which comes in pretty handy in NLP tasks like question answering and summarization.



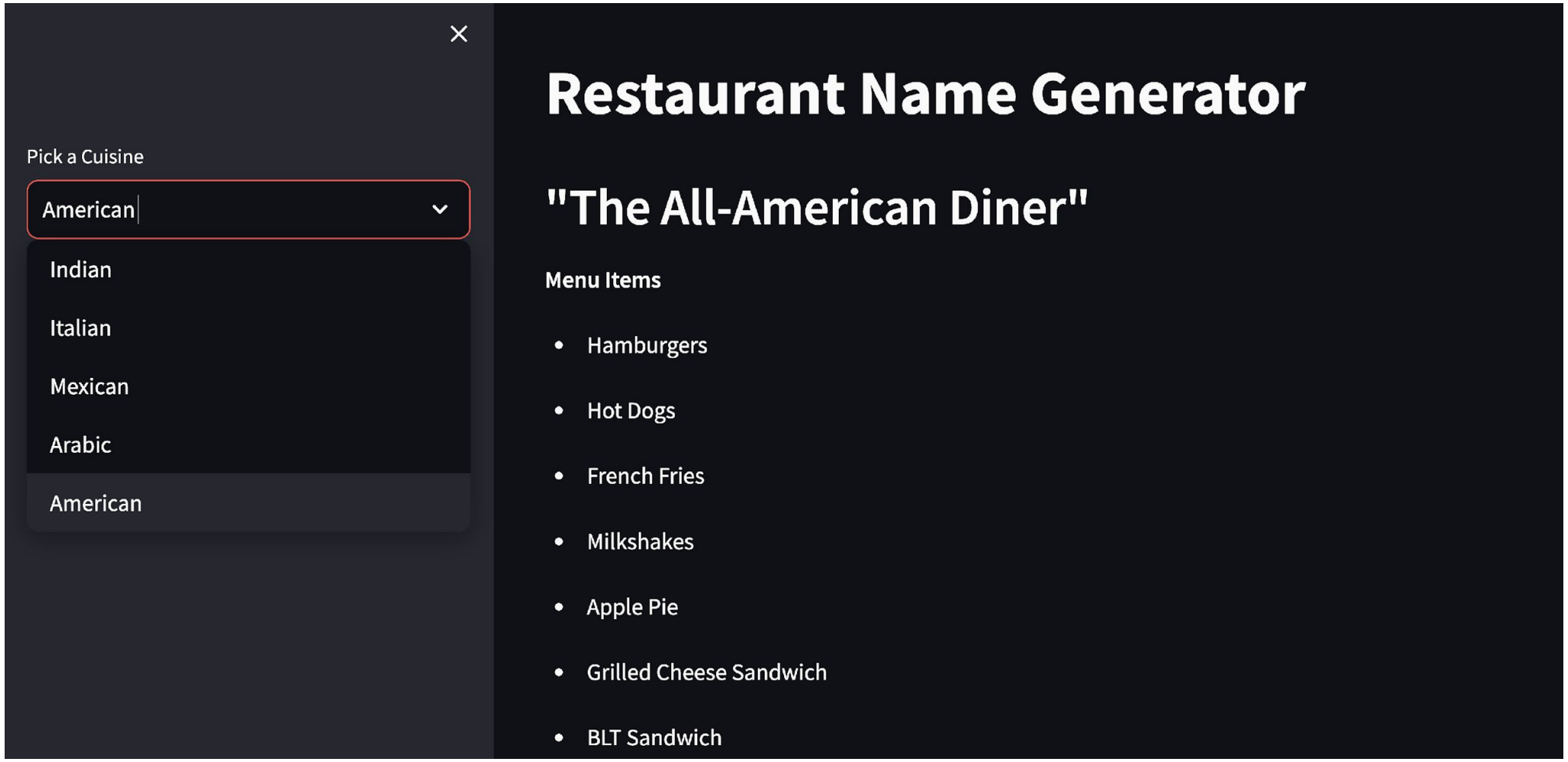
LangChain - LLM



Lets Build a Restaurant

Recommendation Application

Restaurant Recommendation Application



Restaurant Recommendation Application

Step 1 – We'll choose the relevant LLM (out of 40!!)

Restaurant Recommendation Application

Step 2 – We'll use Prompt Template !

(Prompt templates are pre-defined recipes for generating prompts for language models)

Restaurant Recommendation Application

Step 3 – We'll use LLMChain

(A consists of a PromptTemplate, a model and an optional output parser. This chain takes multiple input variables, uses the PromptTemplate to format them into a prompt. It then passes that to the model.

Finally, it uses the OutputParser (if provided) to parse the output of the LLM into a final format)

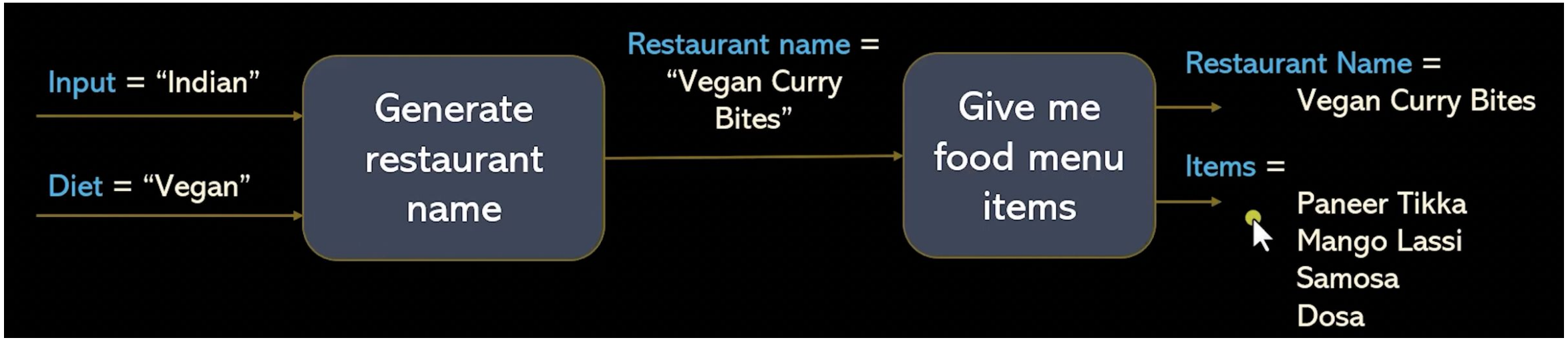
Restaurant Recommendation Application

Step 4 – We'll use SequentialChain

(The next step after calling a language model is make a series of calls to a language model. This is particularly useful when you want to take the output from one call and use it as the input to another)

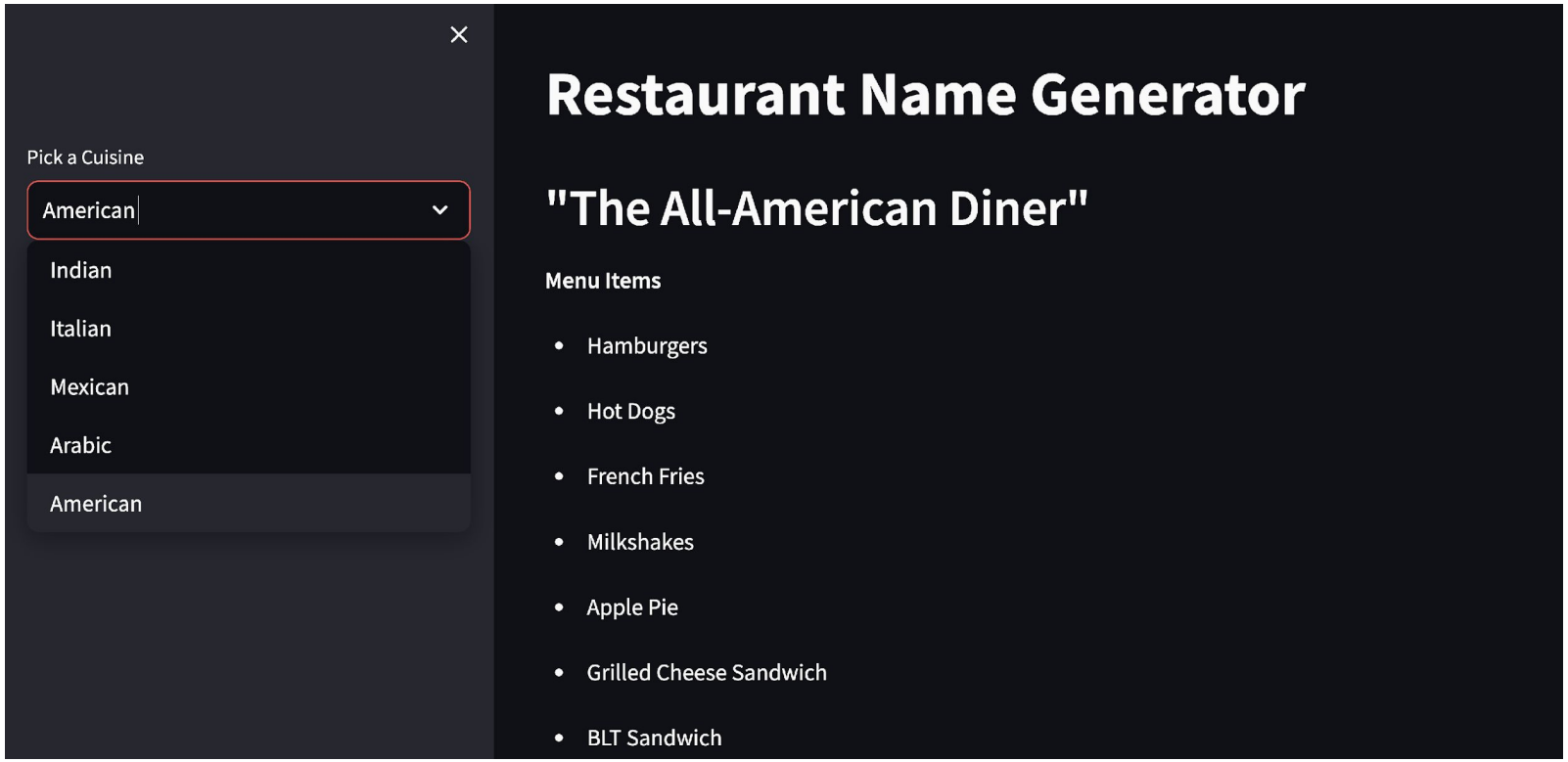
Simple Sequential Chain

Sequential Chain



Restaurant Recommendation Application

Step 5 – We'll build a UI with Streamlit !!



Haystack

Open-source LLM framework to build production-ready applications

Haystack Building Blocks

Nodes

performing different kinds of tasks.

For example, to perform question answering with a PromptNode,
all you need to do is provide it with documents,
a PromptTemplate designed for question answering, and a query.

```
from haystack.nodes import PromptNode

prompt_node = PromptNode(model_name_or_path="gpt-4", api_key='YOUR_OPENAI_KEY')
result = prompt_node.prompt(query="What is Haystack?",
                            documents=documents,
                            prompt_template="deepset/question-answering")
```

Pipeline

Combination of Nodes

When adding nodes to a pipeline, you can define how data flows from one node to the next, until the pipeline has reached a final result.

```
from haystack import Pipeline

p = Pipeline()
p.add_node(component=retriever, name="Retriever", inputs=["Query"])
p.add_node(component=prompt_node, name="PromptNode", inputs=["Retriever"])
result = p.run(query="What did Einstein work on?")
```

Agent

Uses a large language model to reason and answer complex queries that require multiple steps to find the correct answer, including <u>external data sources</u>

```
agent = Agent(
    prompt_node=prompt_node,
    prompt_template=few_shot_agent_template,
    tools=[web_qa_tool],
    final_answer_pattern=r"Final Answer\s*:\s*(.*)",
)
```

REST API

You can deploy Haystack as a REST API.

Using Haystack through an API is useful if you want to deploy NLP functionality in your web or mobile app

```
curl -X 'POST' \
  'http://127.0.0.1:8000/query' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "query": "Who is the father of Arya Stark?",
  "params": {}
}'
```

Retrieval-Augmented-Generation (RAG) Architecture

RAG

RAG integrates information retrieval (or searching) into LLM text generation.

It uses the user input prompt to retrieve external “context” information from a data store that is then included with the user-entered prompt to build a richer prompt containing context information that otherwise would not have been available to the LLM.

The need for RAG?

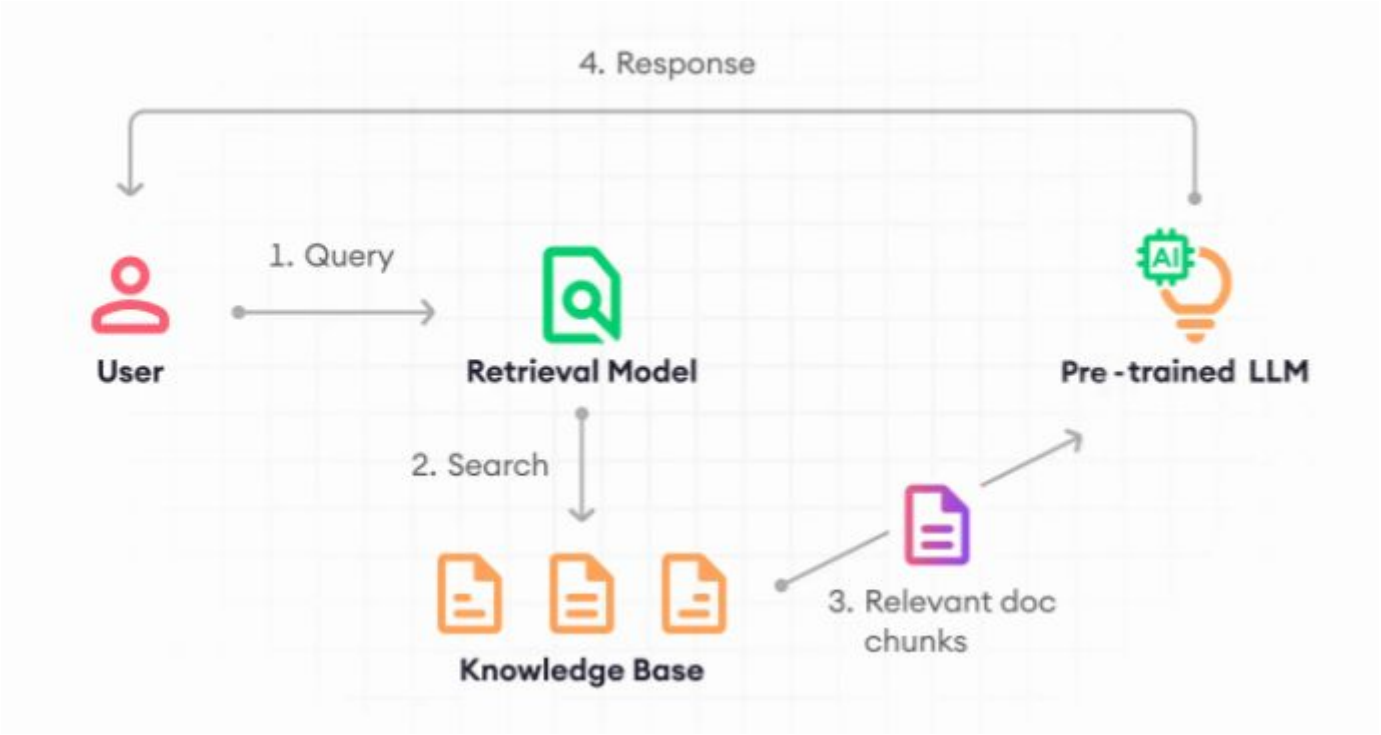
Pre-trained LLMs (foundation models) do not learn over time, often hallucinate, and may leak private data from the training corpus.

To overcome these limitations, there has been growing interest in retrieval-augmented generation which incorporate a vector database and/or feature store with their LLM to provide context to prompts, also known as RAG LLMs.

RAG

The most common systems used to provide external data for RAG LLMs are vector databases and feature stores.

RAG for LLMs works because of the ability of LLMs to perform in-context learning.



RAG real world use case

Some examples of context information used by RAG include:

- real-time context (the weather, your location, etc);
- user-specific information (orders the user has made at this website, actions the user has taken on the website, the user's status, etc);
- relevant factual information (documents not included in the LLM's training data - either because they are private or they were updated after the LLM was trained).

RAG- real world use case

Imagine you have a vast database of scientific articles, and you want to answer a specific question using an LLM like GPT-4:

“What are the latest advancements in CRISPR technology?”

RAG real world use case

Without RAG

The LLM might provide a general overview of CRISPR based on its training data, which could be outdated or lack the latest research insights.

With RAG:

First, the system would query the database of scientific articles to retrieve the most relevant and recent articles on CRISPR advancements.

These retrieved articles would then be used as context when posing the question to the LLM.

The LLM, now equipped with this fresh context, would generate a response that incorporates the latest findings from the retrieved articles, offering a more up-to-date and detailed answer.

Implementing RAG

A typical RAG application has two main components:

Indexing: a pipeline for ingesting data from a source and indexing it. *This usually happens offline.*

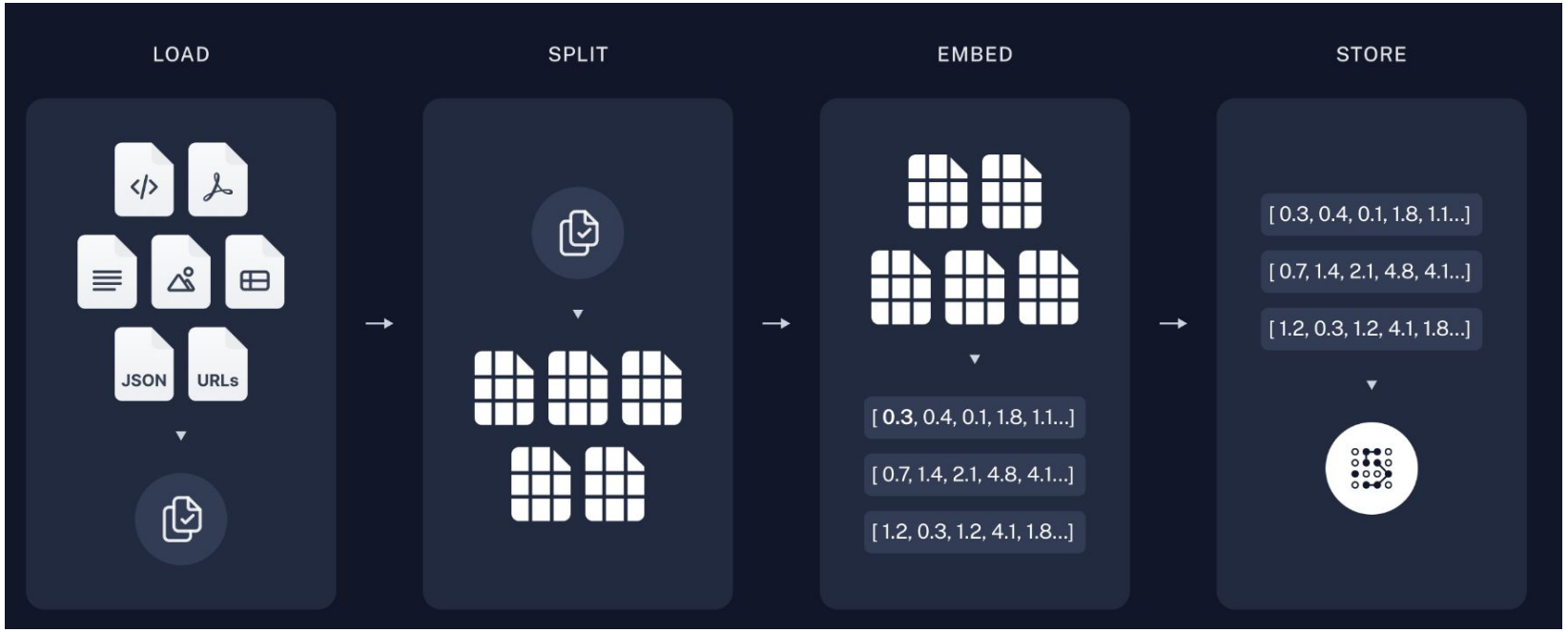
Retrieval and generation: the actual RAG chain, which takes the user query at run time and retrieves the relevant data from the index, then passes that to the model.

Indexing

:**Load**

.First we need to load our data

.This is done with Document Loaders

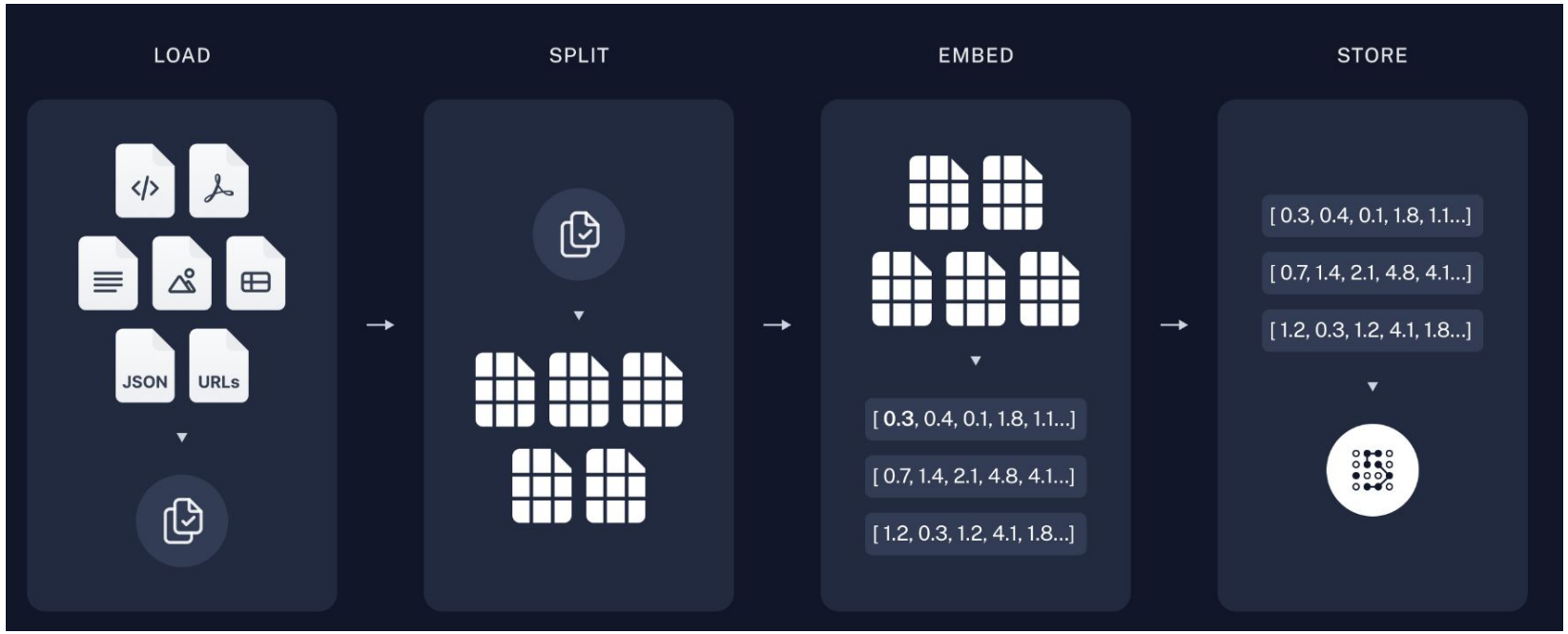


Indexing

:**Split**

.Text splitters break large Documents into smaller chunks

This is useful both for indexing data and passing it into a model, as large chunks are harder to search over and won't fit in a model's finite context .window

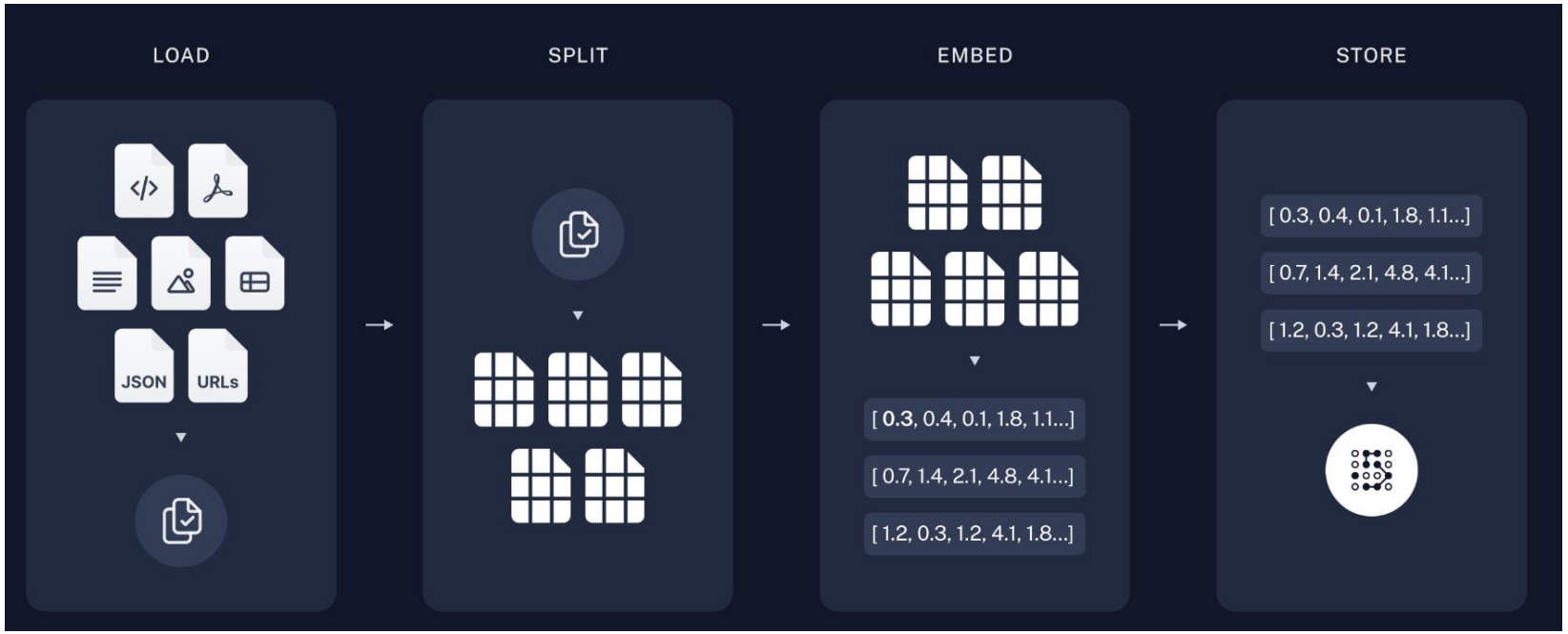


Indexing

:**Store**

We need somewhere to store and index our splits, so that they can be searched .over later

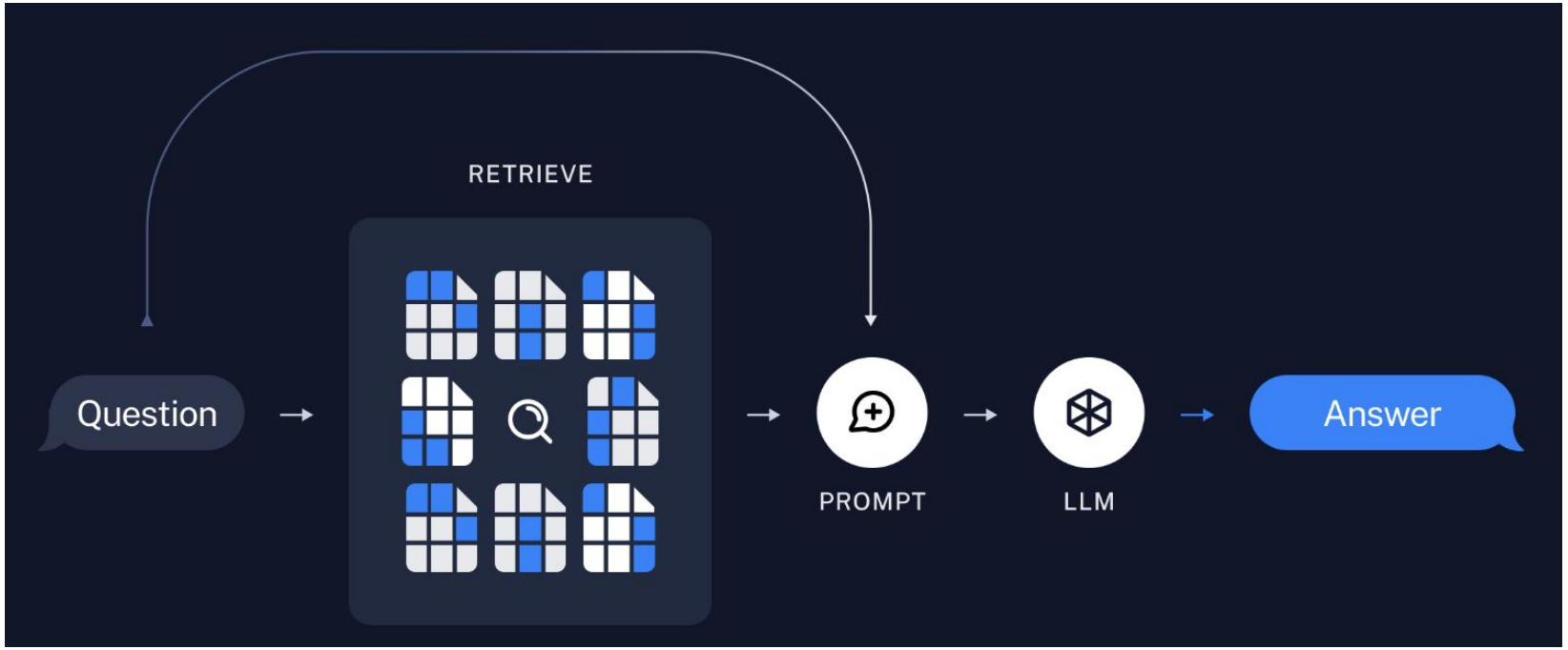
.This is often done using a VectorStore and Embeddings model



Retrieval and Generation

Retrieve: Given a user input, relevant splits are retrieved from storage using a Retriever.

Generate: A ChatModel / LLM produces an answer using a prompt that includes both the question with the retrieved data



YanivArad Training Labs

On our menu

- Intro to AI & Machine Learning
- Generative AI & LLMs Systems
- LLM Parameters
- Prompts Engineering
- Prompts Engineering for Developers
- Integrating LLM API
- LLM for Dev Lifecycle
- LLM Frameworks
- RAG Architecture

AI Agents

AI Agents

An artificial intelligence (AI) agent is a software program that can interact with its environment, collect data, and use the data to perform self-determined tasks to meet predetermined goals.

Humans set goals, but an AI agent independently chooses the best actions it needs to perform to achieve those goals

AI Agent - Example

A contact center AI agent that wants to resolves customer queries.

The agent will automatically ask the customer different questions, look up information in internal documents, and respond with a solution.

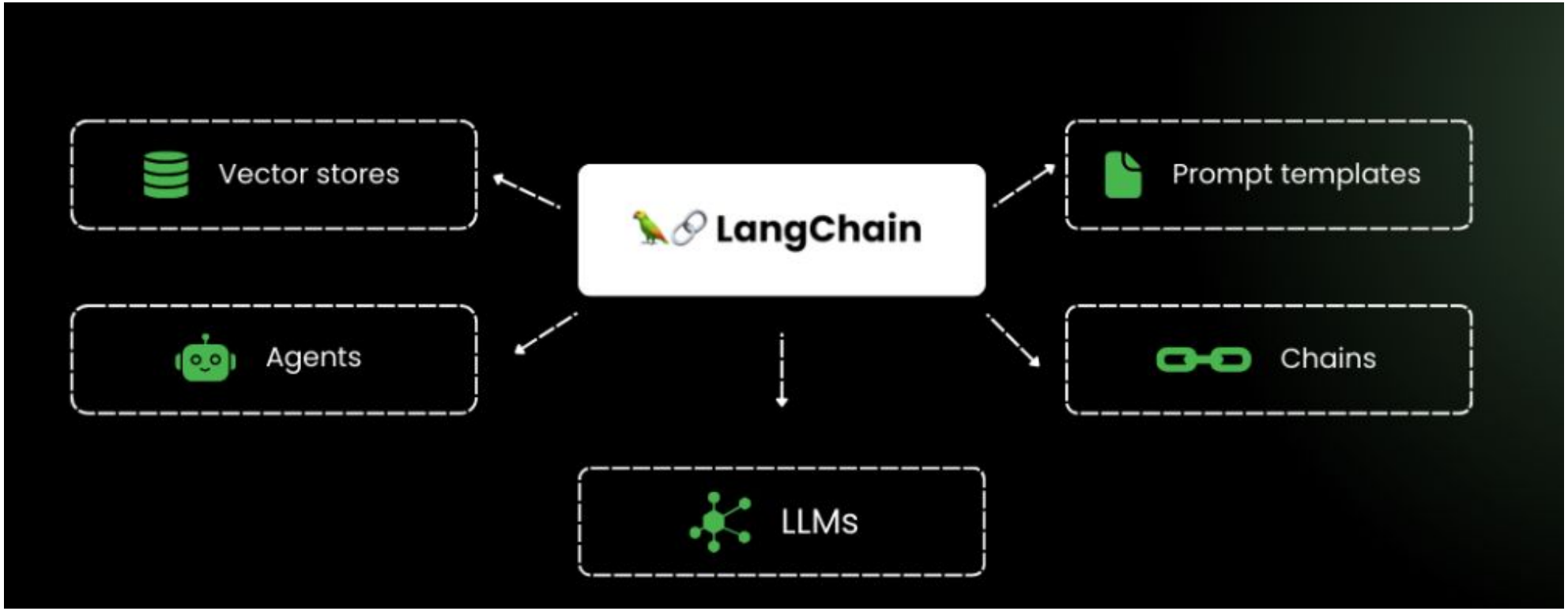
Based on the customer responses, it determines if it can resolve the query itself or pass it on to a human.

LangChain

LangChain

LangChain is an open-source framework that gives developers the tools they need to create applications using large language models (LLMs).

LangChain Components

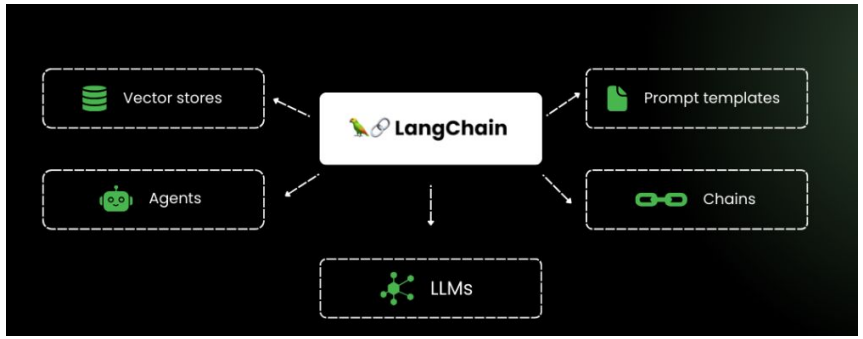


LLMs

Language Models (LLMs) are the core engines behind LangChain applications.

They are responsible for generating human-like text based on the input prompts.

LangChain supports various LLMs, including OpenAI's GPT models, Hugging Face models, and more.

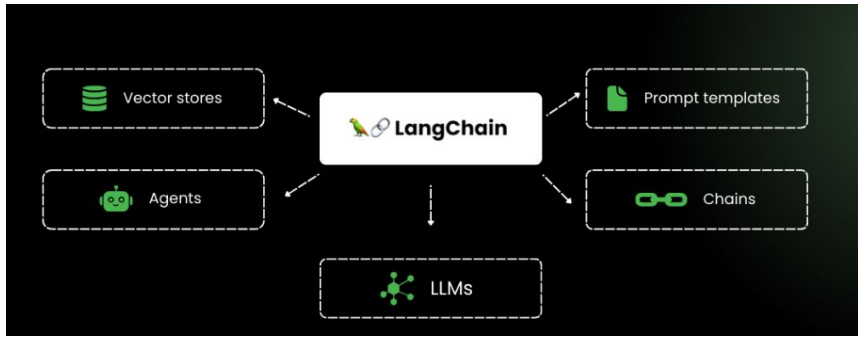


Prompts

Prompts are the input text you provide to an LLM to generate a response.

They are crucial in guiding the LLM's output and defining the task.

LangChain offers a variety of prompt templates and utilities to help you create effective prompts.

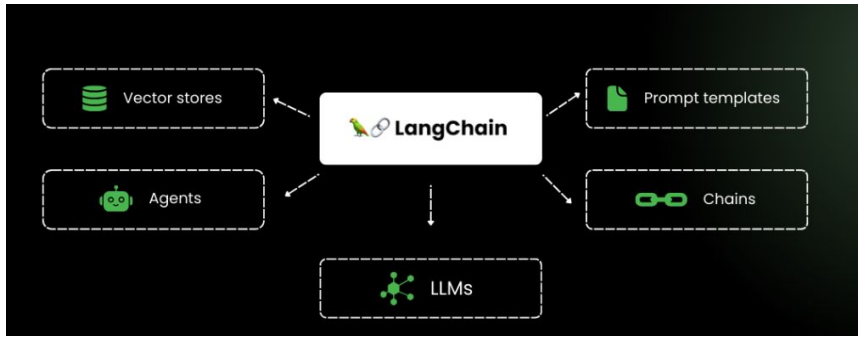


Chains

Chains allow you to combine multiple components, such as prompts and LLMs, to create more complex applications.

They define a sequence of steps to process input, generate output, and perform additional tasks.

LangChain provides a variety of built-in chains and supports the creation of custom chains.

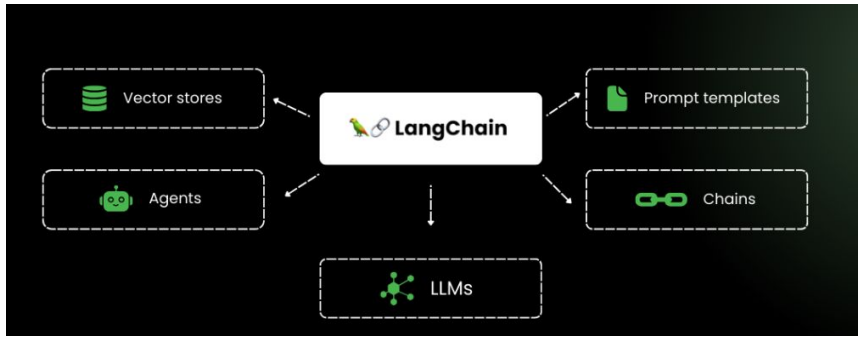


Agents

Agents are high-level abstractions that use chains and tools to accomplish specific goals.

They can make decisions, interact with external tools, and retrieve information to complete tasks.

Agents are particularly useful for building conversational AI applications or automating complex workflows.

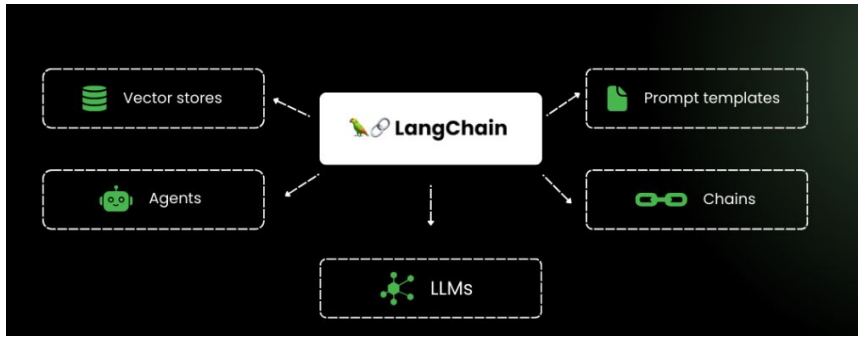


Vector Stores

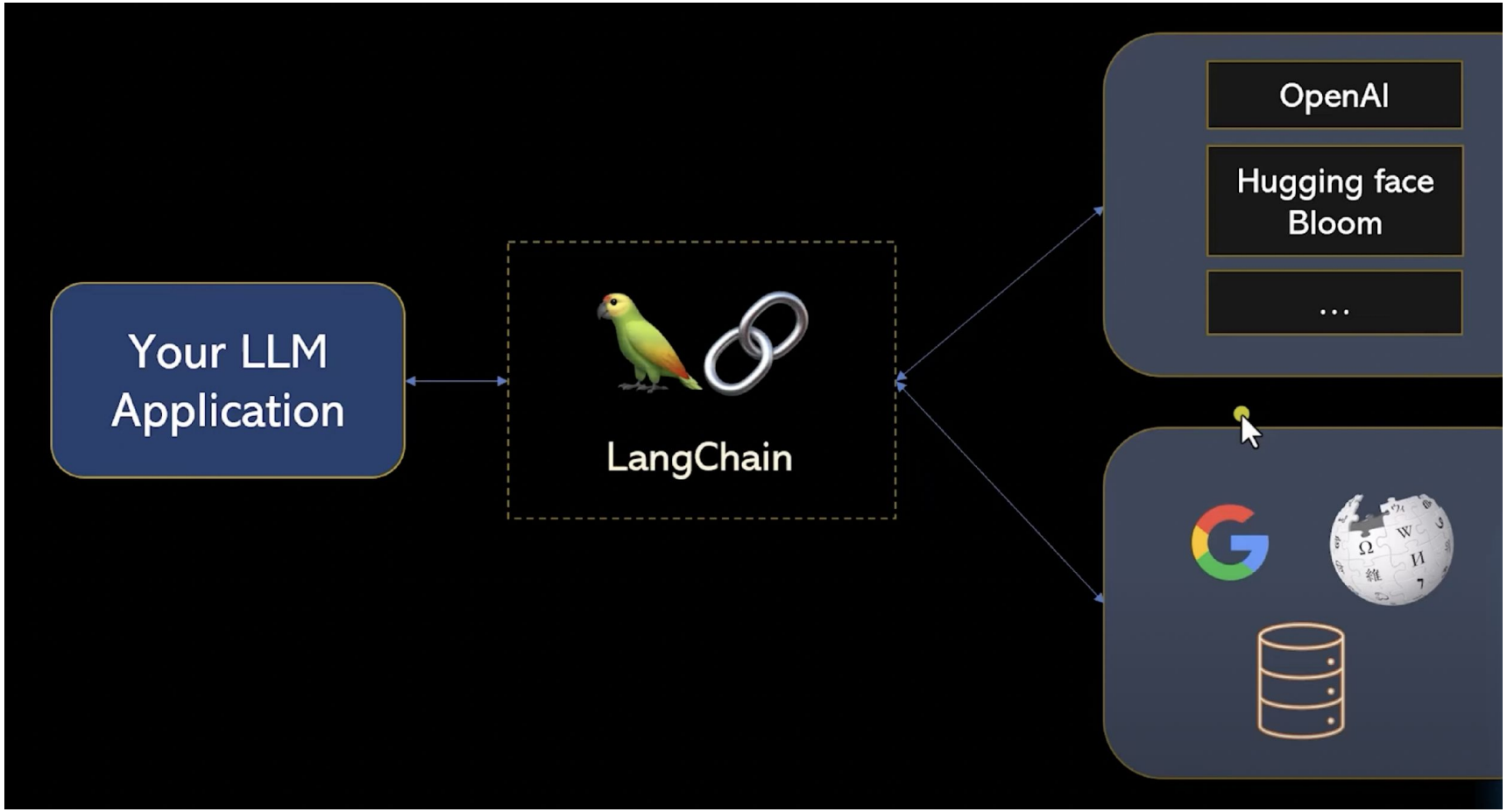
vector stores are databases that store vectors.

Vectors are mathematical representations of words or phrases.

When stored within a vector store, it’s much easier to find similar words or phrases, which comes in pretty handy in NLP tasks like question answering and summarization.



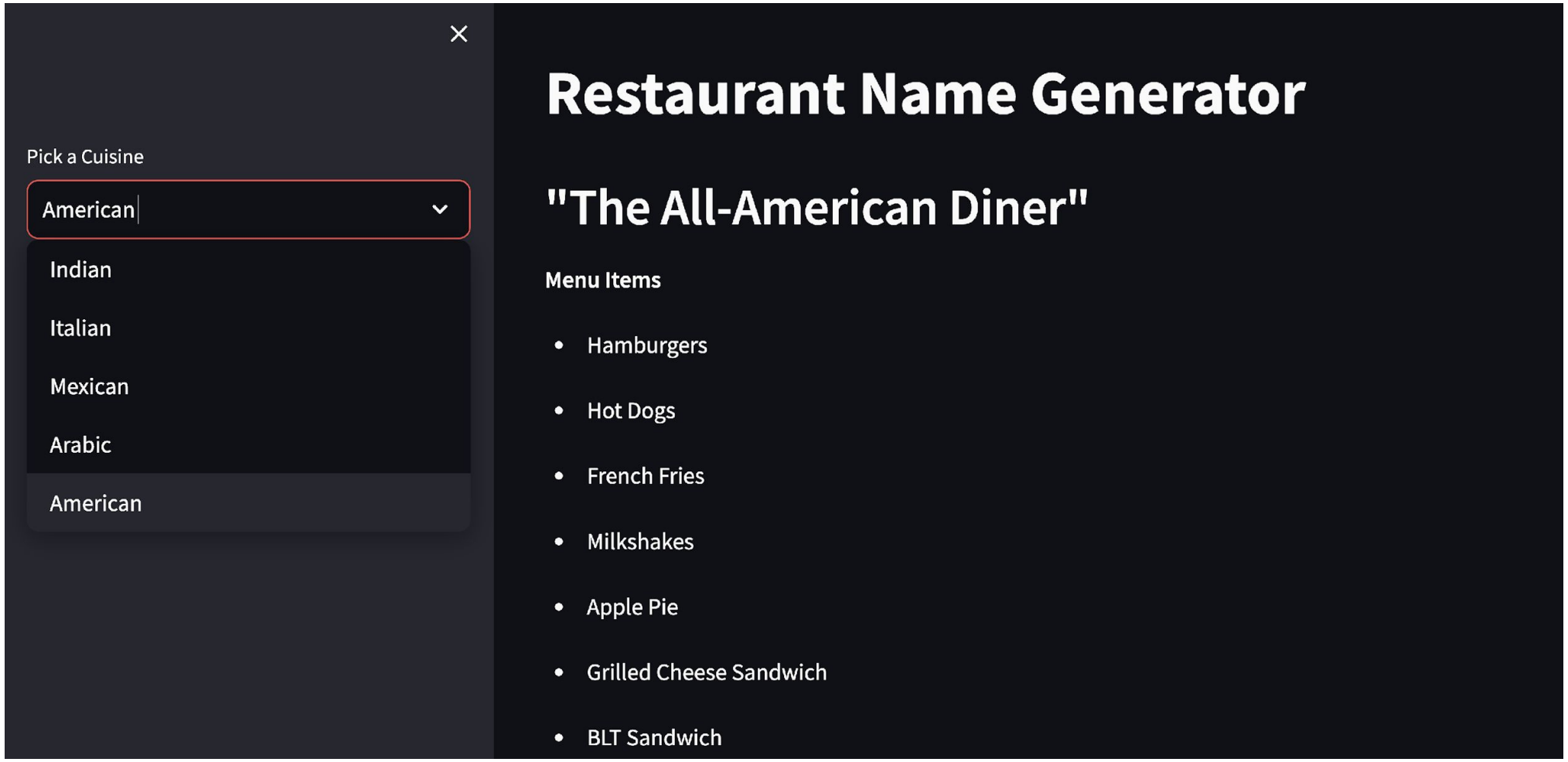
LangChain - LLM



Lets Build a Restaurant

Recommendation Application

Restaurant Recommendation Application



Restaurant Recommendation Application

Step 1 – We'll choose the relevant LLM (out of 40!!)

Restaurant Recommendation Application

Step 2 – We'll use Prompt Template !

(Prompt templates are pre-defined recipes for generating prompts for language models)

Restaurant Recommendation Application

Step 3 – We'll use LLMChain

(A consists of a PromptTemplate, a model and an optional output parser. This chain takes multiple input variables, uses the PromptTemplate to format them into a prompt. It then passes that to the model.

Finally, it uses the OutputParser (if provided) to parse the output of the LLM into a final format)

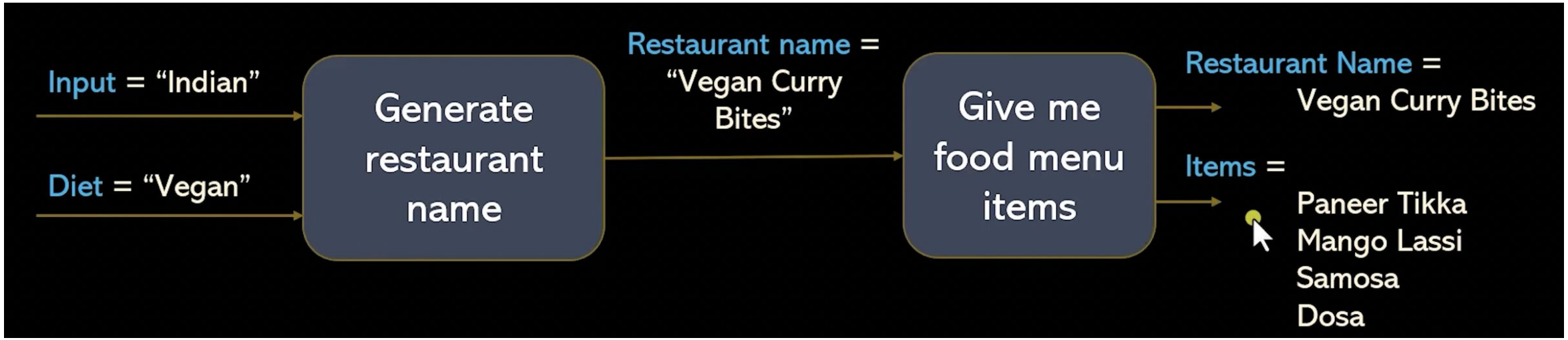
Restaurant Recommendation Application

Step 4 – We'll use SequentialChain

(The next step after calling a language model is make a series of calls to a language model. This is particularly useful when you want to take the output from one call and use it as the input to another)

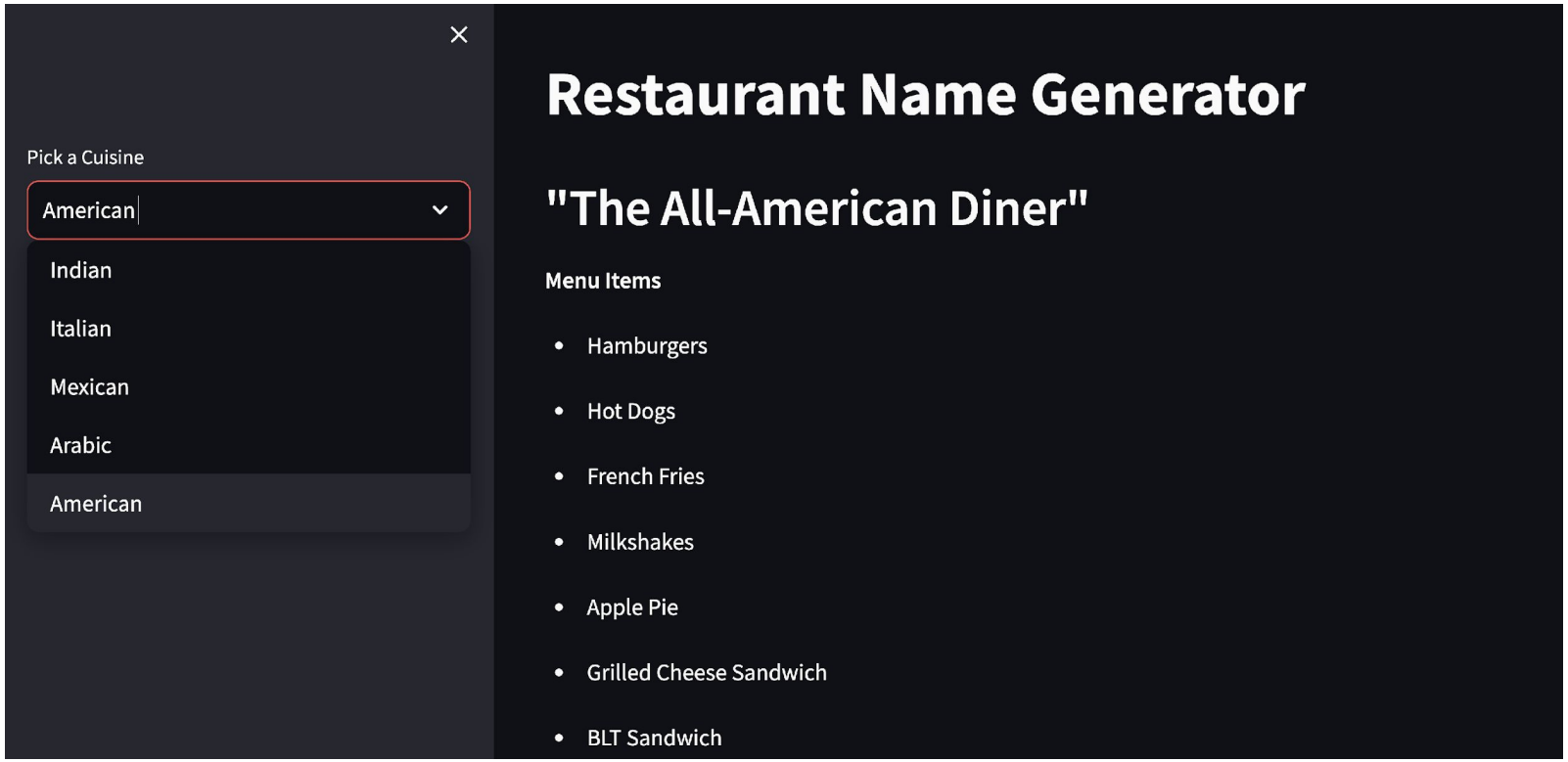
Simple Sequential Chain

Sequential Chain



Restaurant Recommendation Application

Step 5 – We'll build a UI with Streamlit !!



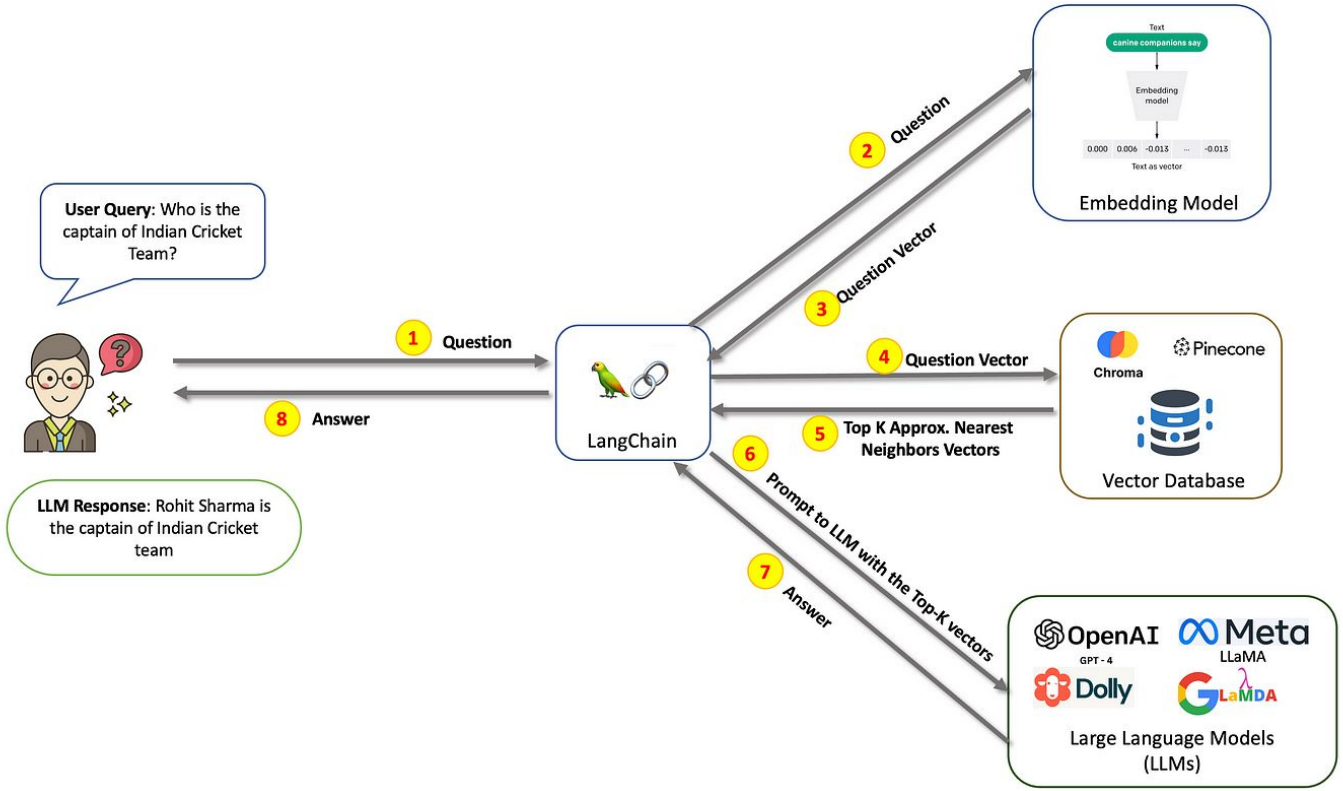
More LangChain Components

Agents

Some applications will require not just a predetermined chain of calls to LLMs/other tools, but potentially an unknown chain that depends on the user's input

.In these types of chains, there is a “agent” which has access to a suite of tools

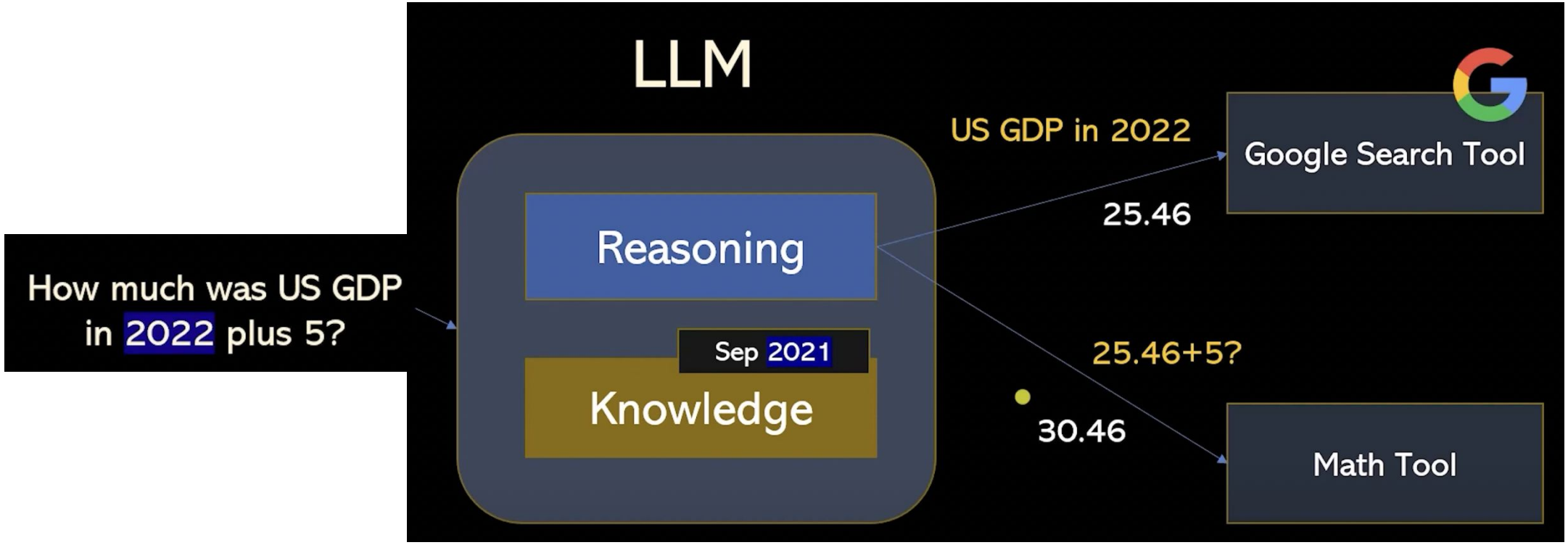
.Depending on the user input, the agent can then decide which, if any, of these tools to call



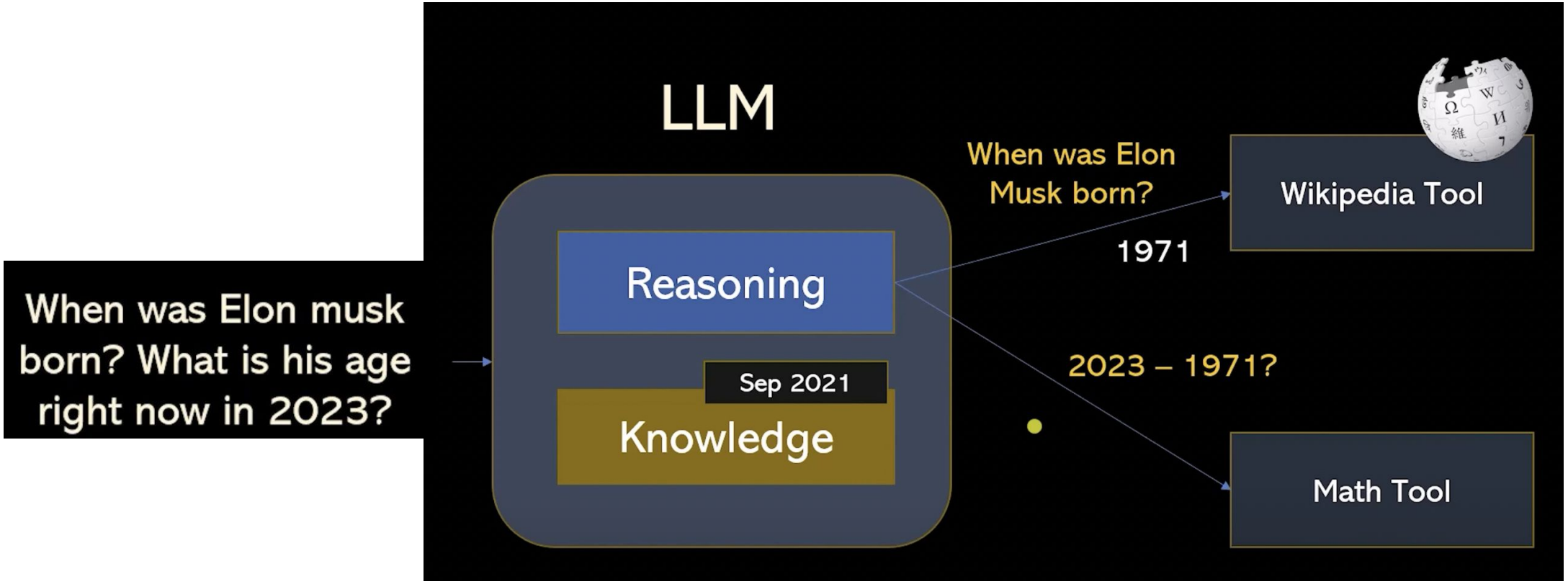
Agents

LLM
Give me two flight options from New York to Delhi on 1 August 2023
Reasoning
Sep 2021
Knowledge
Source: New York
Destination: Delhi
Date: 1 Aug 2023
Results: 2
Expedia Plugin

Agents


LLM
How much was US GDP in 2022 plus 5?
Reasoning
Sep 2021
Knowledge
US GDP in 2022
Google Search Tool
25.46
25.46+5?
30.46
Math Tool

Agents


When was Elon musk born? What is his age right now in 2023?
LLM
Reasoning
Sep 2021
Knowledge
When was Elon Musk born?
1971
Wikipedia Tool
2023 – 1971?
Math Tool

LangGrpag

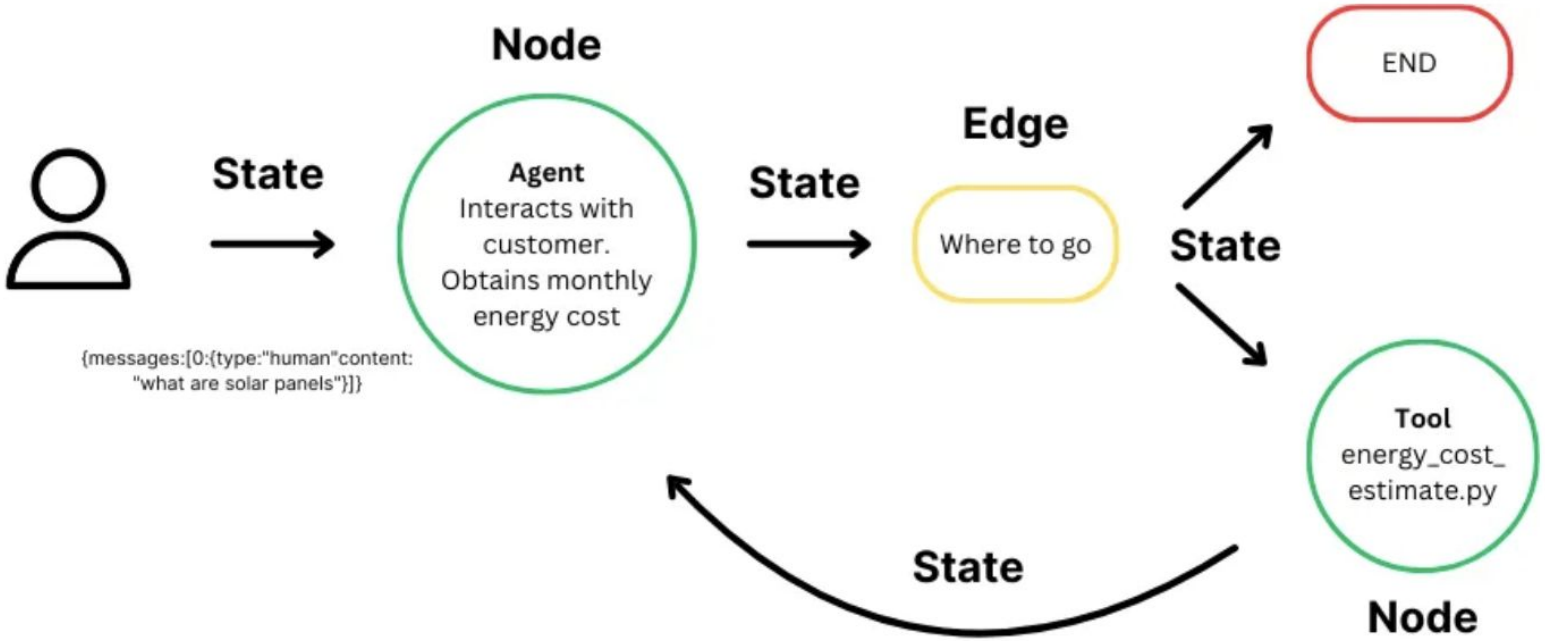
What is LangGrpah?

LangGraph is an advanced library built on top of LangChain, designed to enhance your Large Language Model (LLM) applications by introducing cyclic computational capabilities.

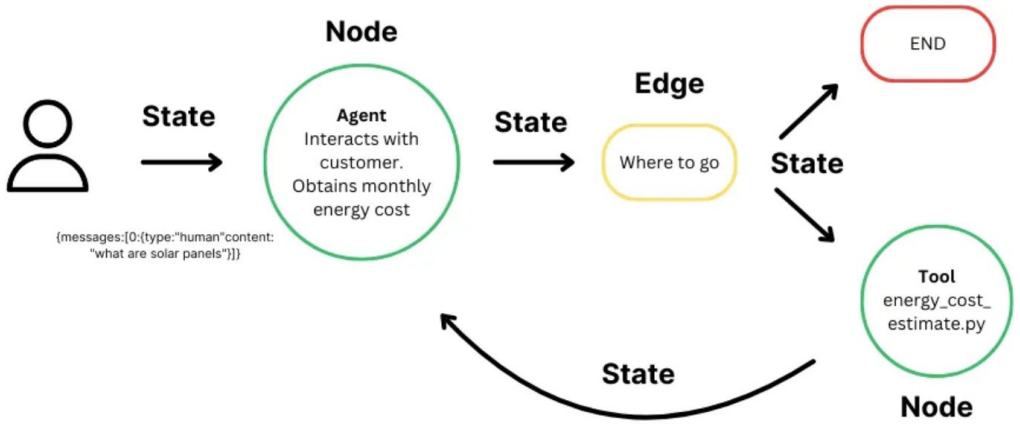
While LangChain allows the creation of Directed Acyclic Graphs (DAGs) for linear workflows, LangGraph takes this a step further by enabling the addition of cycles, which are essential for developing complex, agent-like behaviors.

These behaviors allow LLMs to continuously loop through a process, dynamically deciding what action to take next based on evolving conditions.

What is LangGrpah?



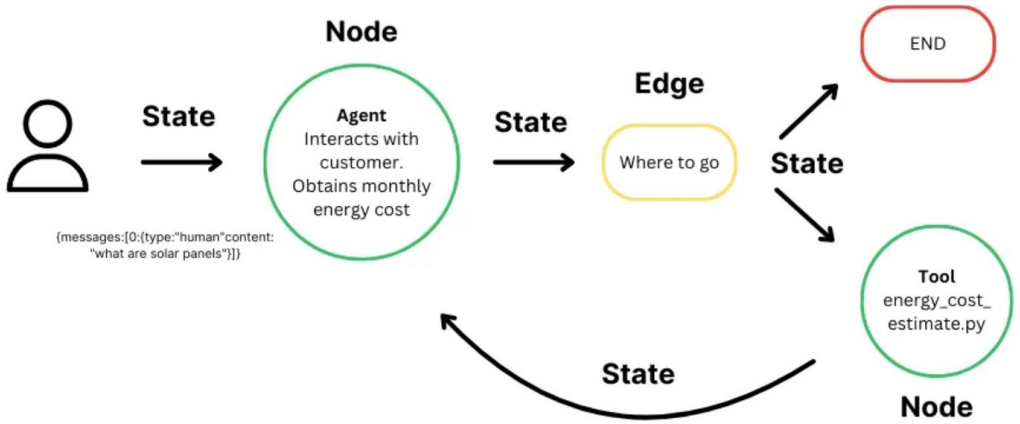
LangGrpah Components



State:

Represents the context or memory that is maintained and updated as the computation progresses. It ensures that each step in the graph can access relevant information from previous steps, allowing for dynamic decision-making based on accumulated data throughout the process.

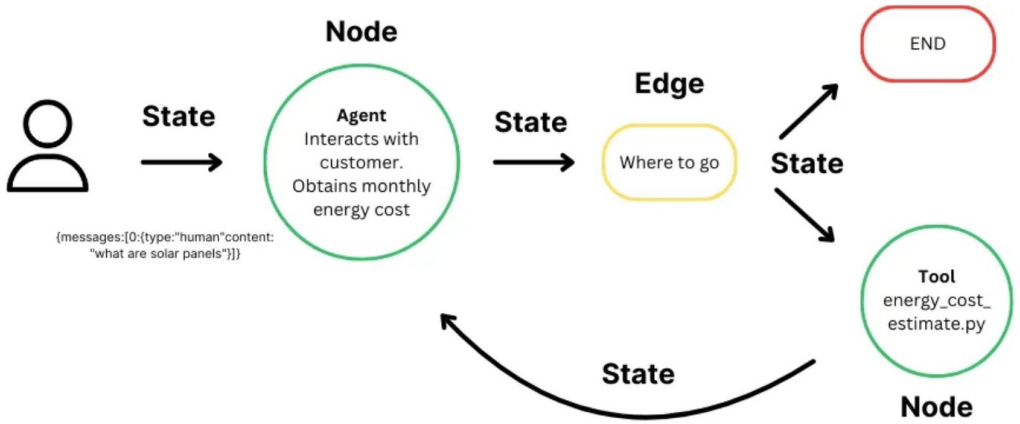
LangGrpah Components



Nodes:

Serve as the building blocks of the graph, representing individual computation steps or functions. Each node performs a specific task, such as processing inputs, making decisions, or interacting with external systems. Nodes can be customized to execute a wide range of operations within the workflow.

LangGrpah Components



Edges:

Connect nodes within the graph, defining the flow of computation from one step to the next. They support conditional logic, allowing the path of execution to change based on the current state and facilitate the movement of data and control between nodes, enabling complex, multi-step workflows.

LangGrpah

LangGraph redefines AI development by seamlessly managing graph structure, state, and coordination, empowering the creation of sophisticated, multi-actor applications.

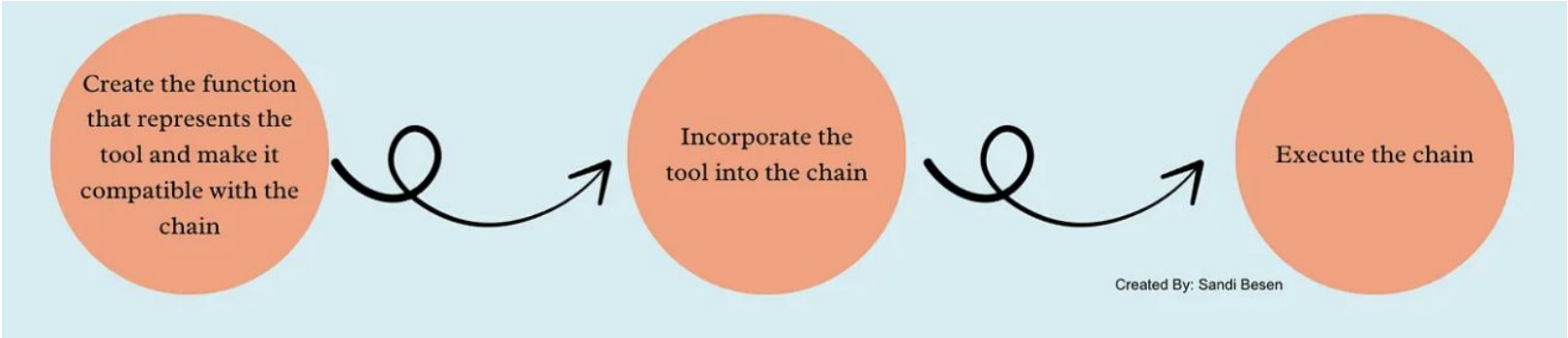
With automatic state management, LangGraph ensures that context is preserved across interactions, enabling your AI to respond intelligently to changing inputs

LangChain Agent vs. LangGraph Agent

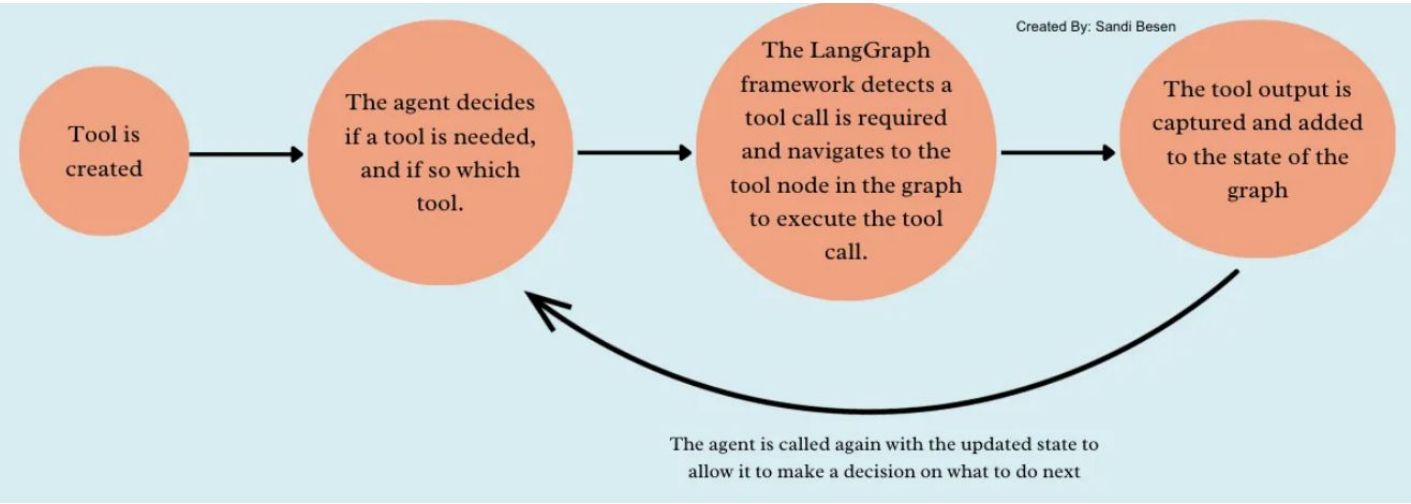
LangChain focuses on either linear workflows through the use of chains or different AI agent patterns

LangGraph focuses on creating a more flexible, granular, process based workflow that can include AI agents, tool calls, procedural code, and more.

LangChain Agent



1. Create the function that represents the tool and make it compatible with the chain
2. Incorporate the tool into the chain
3. Execute the chain



1. The tool is defined
2. the tool is bound to the agent
3. The agent decides if a tool is needed, and if so which tool.
4. The LangGraph framework detects a tool call is required and navigates to the tool node in the graph to execute the tool call.
5. The tool output is captured and added to the state of the graph
6. The agent is called again with the updated state to allow it to make a decision on what to do next

Multi-Agents

Multi Agent

A multiagent system (MAS) consists of multiple

artificial intelligence (AI) agents working collectively to perform tasks on behalf of a user or another system.

Single Agent vs Multi Agent

Single AI agents are intelligent systems that make independent decisions, adapt to their environments, and pursue varying means to achieve predefined goals

multi-agent systems orchestrate multiple AI agents to work together toward shared goals. These systems create a collaborative environment where specialized agents can communicate, coordinate their actions, and divide complex tasks into manageable portions.

Multi-Agent

YanivArad Training Labs

multiple independent actors powered by language models connected in a specific way.

Each agent can have its own prompt, LLM, tools, and other custom code to best collaborate with the other agents.

That means there are two main considerations when thinking about different multi-agent workflows:

- What are the multiple independent agents?
- How are those agents connected?

Benefits of Multi-Agent

- Grouping tools/responsibilities can give better results. An agent is more likely to succeed on a focused task than if it has to select from dozens of tools.
- Separate prompts can give better results. Each prompt can have its own instructions and few-shot examples. Each agent could even be powered by a separate fine-tuned LLM!
- Helpful conceptual model to develop. You can evaluate and improve each agent individually without breaking the larger application.
- Multi-agent designs allow you to divide complicated problems into tractable units of work that can be targeted by specialized agents and LLM programs.

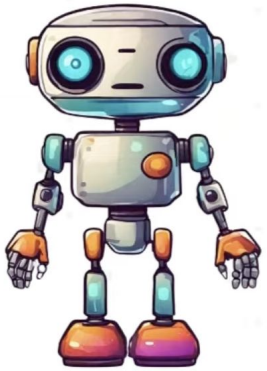
CrewAI

What is CrewAI?

CrewAI is an Agent Framework that allows us to create agents to help us solve simple or complex tasks.

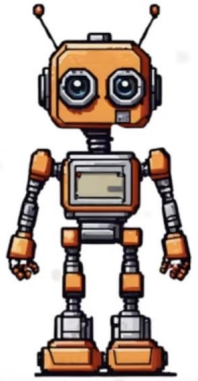
AI agents assume **roles**, share **goals**, and operate in a cohesive unit - much like a well-oiled **crew**.

Task: develop a game



Task: blog post

Task: trip planner



crewai

YanivArad Training Labs

5 Main Concepts

- Agent
- Task
- Tool
- Process
- Crew

Agents

Attribute	Description
role	the role of the agent, what do you want this agent for
goal	the objective of the agent
backstory	the backstory of the agent, gives more context about the role and goal
LLM	you can choose the specific language model to run this agent
function_calling_llm	the language model that will be for tool calling
max_iter	max iterations for an agent to perform a task
memory	whether the agent should have memory or not
verbose	gives more outpout to what the agent is doing behind the scenes in the terminal
tools	the tools this agent can use, such as SearchTools or GitHub Tools

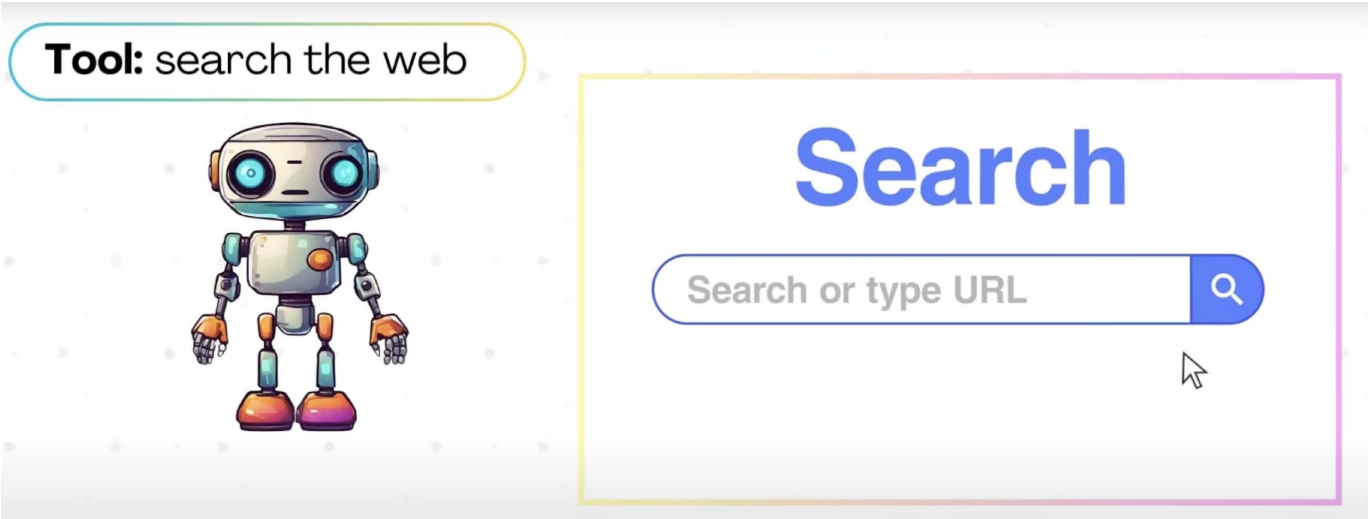
Tasks

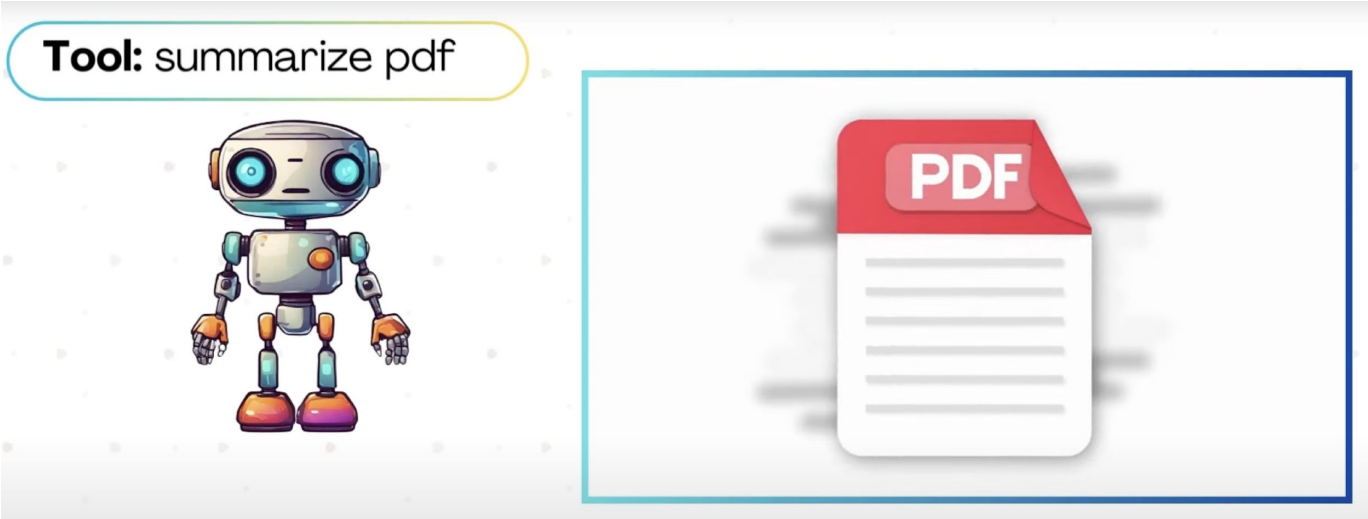
Attribute	Description
agent	agent responsible for task execution, represents entity performing task
async_execution	boolean flag indicating asynchronous task execution
description	descriptive text detailing task's purpose and execution
expected_output	clear definition of expected task outcome
output_file	file path for storing task output
output_json	pydantic model for structuring JSON output
output_pydantic	pydantic model for task output
tools	list of tools/resources limited for task execution

Tasks

Attribute	Description
agent	agent responsible for task execution, represents entity performing task
async_execution	boolean flag indicating asynchronous task execution
description	descriptive text detailing task's purpose and execution
expected_output	clear definition of expected task outcome
output_file	file path for storing task output
output_json	pydantic model for structuring JSON output
output_pydantic	pydantic model for task output
tools	list of tools/resources limited for task execution

Tool


Tool: search the web
Search
Search or type URL


Tool: summarize pdf
PDF

Process

Detailed guide on workflow management through processes in CrewAI, with updated implementation details.

Crew

Represents a collaborative group of agents working together to achieve a set of tasks.

Each crew defines the strategy for task execution, agent collaboration, and the overall workflow.