Develop Generative AI Applications Get Started

Artificial Intelligence

AI models

- Learn from massive existing **data** through **training**

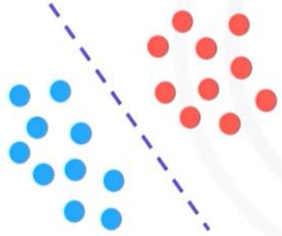
Discriminative AI

Generative AI

Discriminative AI

Distinguishes between different classes of data

Each data point labeled with its class

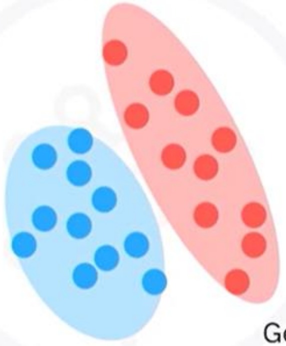


Limitations:

- Cannot understand context
- Cannot generate new content

Generative AI models

Create new content based on training data



Generative AI

Prompt

- Text
- Image
- Video
- Other forms of input

Output

- Text
- Images
- Audio
- Video
- Code
- Data

Text	Text
Text	Image
Image	Video

Discriminative AI versus generative AI

Discriminative AI

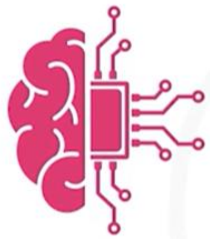
Is this image a drawing of a nest or an egg?

Generative AI

Draw an image of a nest with three eggs in it.

"AI can not only boost our analytic and decision-making abilities but also heighten creativity."
-Harvard Business Review

How does generative AI develop creativity?



Creative skills through generative AI models

- Generative adversarial networks (GANs)
- Variational autoencoders (VAEs)
- Transformers
- Diffusion models

Evolution of generative AI

1950s
- Origin of machine learning
- Exploring algorithmic data creation

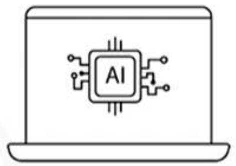
1990s
Neural networks advanced generative AI

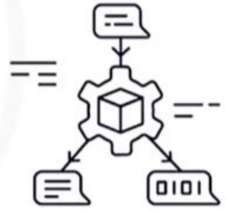
2010s
Deep learning, large data sets, and enhanced computing accelerated generative AI

2014 and beyond
GANs and other models revolutionized generative AI

Foundation models

- AI models with broad capabilities
- Adapted to build specialized and advanced models or tools
- Large language models: Process and generate text





Foundation models: Examples

Examples of LLMs

- **OpenAI**: GPT n-series (GPT 1, GPT2, GPT-3/3.5, and GPT-4)
- **Google**: PaLM
- **Meta**: Llama

Examples of models for image generation

- Stable Diffusion
- DALL·E

Generative AI tools

- Generative AI tools for diverse uses cases

Text generation
- ChatGPT
- Gemini

Image generation
- DALL·E 2
- Midjourney

Video generation
- Synthesia

Code generation
- Copilot
- AlphaCode

Recap

- Generative AI models generate new content based on the training data
- Building blocks of generative AI include GANs, VAEs, transformers, and diffusion models
- Foundation models can be adapted to create specialized models or tools
- Generative AI has potential applications in different domains and industries

In-context learning

- Method of prompt engineering
- Demonstrations of the task provided to the LLM as part of the prompt
- Doesn't require additional training
- New task learned from a small set of examples presented within the context at inference time

In-context learning

Advantages:

- No fine-tuning needed
- Reduces time and resource consumption
- Improves performance

Disadvantages:

- Limited to what can fit in-context
- Complex tasks need gradient steps
- Involves adjustments based on error gradients

What are prompts?

Instructions

- Clear, direct commands that tell the AI what to do
- Need to be specific to ensure LLM understands the task

Context

- Information that helps LLM make sense of instructions
- Relevant details that shape the AI's response

Prompts → IBM → Achieve a desired task

Combine to tailor LLMs to perform tasks:

- Answer queries
- Analyzing data
- Generate content

What is prompt engineering?

- Designing and refining prompts to communicate with AI systems (LLMs)
- Involves crafting questions, commands, or statements to elicit accurate, relevant, and contextually appropriate responses
- Fundamental to customer service and computational linguistics

Why prompt engineering?

Directly influences the effectiveness and accuracy of LLMs

Ensures LLMs generate relevant, precise, and contextually appropriate responses

Meets user needs through clearer prompts and reduced misunderstanding

Eliminates the need for continual fine-tuning

First basic prompt

The wind is

```
Blowing gently through the trees,
whispering secrets and stories to anyone
who cares to listen.
```

Elements of a prompt

Instructions	Context	Input data	Output indicator
Tell LLM what needs to be done	Helps LLM understand the scenario	The actual data the LLM will process	Part of the prompt where the LLM's response is expected
'Classify the following customer review into neutral, negative, or positive sentiment.'	'This review is part of feedback for a recently launched product.'	'The product arrived late, but the quality exceeded my expectations.'	'Sentiment:'

Recap

- In-context learning: Method of prompt engineering where demonstrations of the task are provided to the model
- Prompts are inputs given to an LLM, to guide it toward performing a specific task
- Prompt engineering: Process to design and refine the questions, commands, or statements to get relevant and accurate responses
- Advantages of prompt engineering :
  - Boosts the effectiveness and accuracy of LLMs
  - Ensures relevant responses
  - Facilitates meeting user expectations
  - Eliminates the need for continual fine-tuning
- Key prompt elements: Instructions, context, input data, and output indicator

What you will learn

- Describe LangChain's purpose
- Discuss its benefits
- Devise practical uses for LangChain
- Explain how LangChain works with other data types

AI Python framework

- Open-source
- Integrate LLMs into AI apps

Purpose

So, what is considered sustainable or green publishing? These terms do not yet have a clear, concise definition, but in its new stages of life, there are many various descriptors thought of and considered when looking at this type of publishing. To Dennis Stovall, director of publishing at Portland State University, "environmental and cultural" factors are the two most important concerns when discussing it. In Rethinking Paper & Ink, "sustainable" is defined as "using human and natural resources to meet human needs in a way that does not jeopardize the needs of future generations", and is also expressed when discussing the "triple bottom line", which measures performance "based on the economic, environmental,

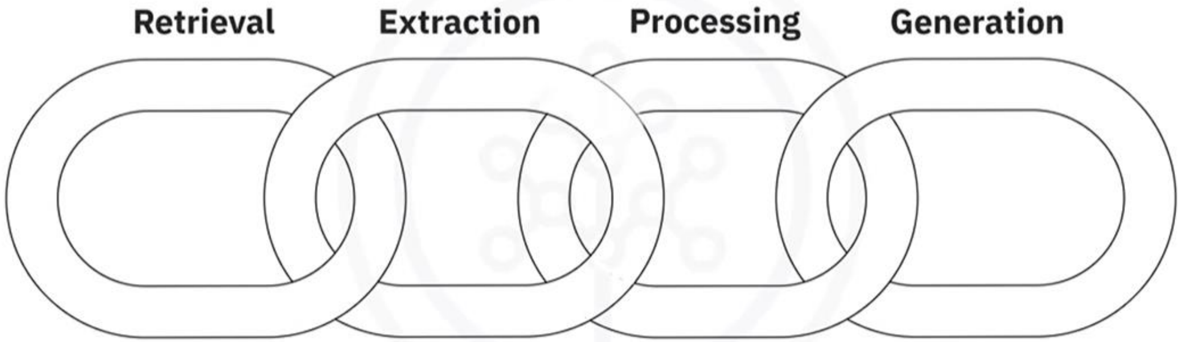
and social performances of a business" (5). Ultimately, Stovall also explains that this form of publishing "now, is about education, research, and best practices that both inform the production choices we make and provide the momentum for continual innovation and improvement". Essentially, what this paper aims to do then is just that - and to explore not just the definition of green or sustainable publishing, but also the efforts and practices being made under this umbrella in the midst of a major publishing shift. This will include the exploration of environmentally friendly efforts such as book materials, sustainable practices within book-making, the cultural and humanistic importance and emphasis on proper labor, green

marketing, ethics within green publishing companies, and how the internet and e-book can be considered "green" too.

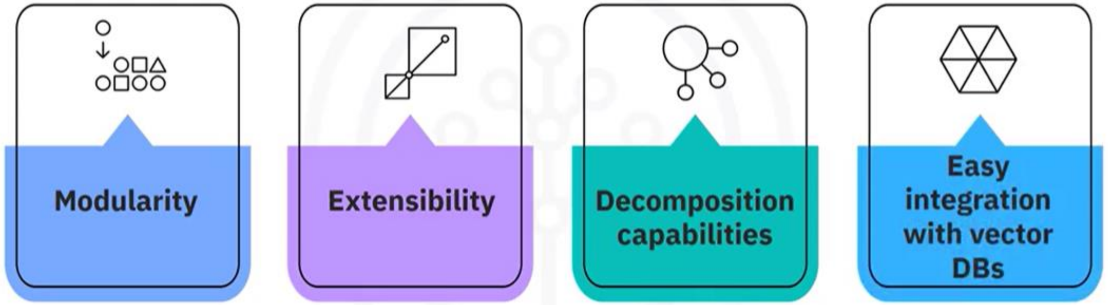
In recent years we have seen a dramatic improvement in recycling efforts, and this is certainly the most widely used sustainable printing practice among both big companies and small publishing houses. Lantern Books, who have been awarded for their paper practices, have been, according to an article from the April, 2007 issue of Publisher's Weekly, "among the first book publishers to develop a meaningful environmentally friendly paper policy" and has "printed most of its books on 100-percent recycled paper". Chelsea Green, a leading publishing company in sustainability practices and

- Complex prompts
- Extensive text-based data
- Concise summaries

Chaining



Benefits



Recap

In this video, you learned that:

- LangChain is a Python framework for pinpointing relevant information in text and providing methods for responding to complex prompts
- Benefits include modularity, extensibility, decomposition capabilities, and easy integration with vector databases
- Several practical applications include deciphering complex legal documents, extracting key statistics from reports, customer support, and automating routine writing tasks
- LangChain can be used with other data types by using external libraries and models

Zero-shot prompt

Instructs an LLM to perform a task without any prior specific training or examples.

Example:

```
Classify the following statement as true or false:
'The Eiffel Tower is located in Berlin.'
```

```
Answer:
```

One-shot prompt

Gives the LLM a single example to help it perform a similar task.

Example:

```
Translating a sentence from English to French:
English: “How is the weather today?”
French: “Comment est le temps aujourd'hui?”
Now, translate the following sentence from English to French:
English: “Where is the nearest supermarket?”
French:
```

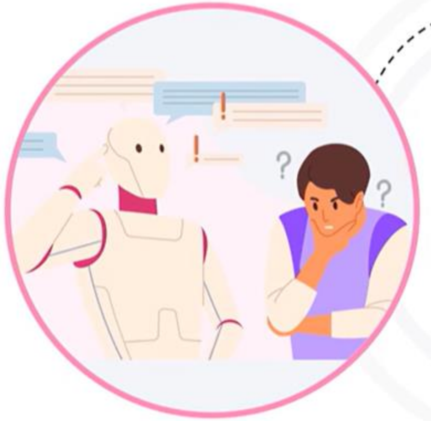
Few-shot prompt

Learns from a small set of examples before tackling a similar task to generalize from a few instances to new data.

Example:

```
Here are few examples of classifying emotions in statements:
Statement: 'I just won my first marathon!'
Emotion: Joy
Statement: 'I can't believe I lost my keys again.'
Emotion: Frustration
Statement: 'My best friend is moving to another country.'
Emotion: Sadness
Now, classify the emotion in the following statement:
Statement: 'That movie was so scary I had to cover my eyes.'
Emotion:
```

Chain-of-thought (CoT) prompting



Guides LLMs through complex reasoning in a step-by-step manner

Effective where the solution requires multiple intermediate steps or reasoning that mimics human thought processes

Chain-of-thought (CoT) prompting

Example:

Model input

```
Consider the problem: 'A store had 22 apples. They sold 15 apples today
and got a new delivery of 8 apples. How many apples are there now?'

Break down each step of your calculation
```

Model output

```
Sure, let's break down the calculation step by step:

1. Initial number of apples: Start with the number of apples the store had
   initially, which is 22.
2. Apples sold: Subtract the number of apples sold today from the initial number.
   The store sold 15 apples, so subtract 15 from 22: 22 - 15 = 7
3. New delivery: Add the number of apples from the new delivery. The store received
   8 new apples, so add 8 to the remaining apples: 7 + 8 = 15

So, after selling 15 apples and receiving a new delivery of 8 apples, the store now
has 15 apples.
```

Self-consistency

Model input

```
When I was 6, my sister was half of my age. Now I am 70, what age is my
sister?

Provide three independent calculations and explanations, then determine
the most consistent result.
```

Self-consistency

Self-consistency

Model output

```
When you were 6, your sister was half your age, making her 3 years old. The age difference between you and your sister is 6-3 = 3 years.

Now that you are 70, your sister, being 3 years younger, is 70 - 3 = 67.
```

```
You were 6 and your sister was half your age, so she was 3.

Since that time, 64 years have passed (from when you were 6 to when you are now 70).

Your sister's age would then be 3 + 64 = 67.
```

```
The ratio when you were 6 (you were twice as old as your sister) means she was 3.

The consistent age gap is 3 years, as previously calculated.

Apply this age gap to your current age: 70 - 3 = 67.
```

```
It's consistent that your sister is 67 years old now.
```

Tools and applications

Facilitate interactions with LLMs:

Playground | LangChain | HuggingFace | IBM AI classroom

Interactive experimentation: Allows real-time tweaking and testing of prompts

Diverse model access: Provides access to various pre-trained models suitable for different tasks and languages

Collaboration and sharing: Facilitates sharing and collaborative editing of prompts

Performance tracking: Offers tools to track changes, analyze results, and optimize prompts

Prompt template

LangChain

- Uses prompt templates: Predefined recipes for generating effective prompts for LLMs
- Prompt templates include:
  - Instructions for the language model
  - A few-shot examples to help model understand context and expected responses
  - Specific question directed at the language model

Prompt template

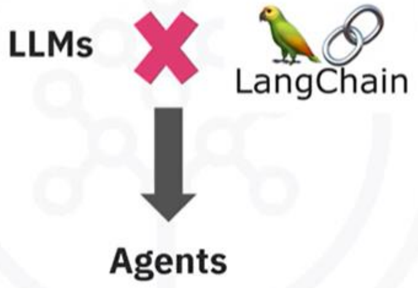
```python
from langchain_core.prompts import PromptTemplate

prompt_template = PromptTemplate.from_template(

    "Tell me a {adjective} joke about {content}."
)
```

```python
prompt_template.format(adjective="funny", content="chickens")
```

```python
"Tell me a funny joke about chickens."
```

Applications

LLM- and LangChain-powered agents



Applications

Agents perform complex tasks across domains using different prompts

Examples:

- Q and A agents with sources
- Content agents for creation and summarization
- Analytic agents for data analysis and business intelligence
- Multilingual agents for seamless, context-aware translation and communication

Recap

- Advanced methods for prompt engineering: Zero-shot prompt, few-shot prompt, chain-of-thought prompting, and self-consistency
- Prompt engineering tools facilitate interactions with LLMs
- LangChain uses ‘prompt templates,’ which are predefined recipes for generating effective prompts for LLMs
- Agent: Key component in prompt applications that can perform complex tasks across various domains using different prompts

What is LCEL?

LangChain Expression Language or LCEL

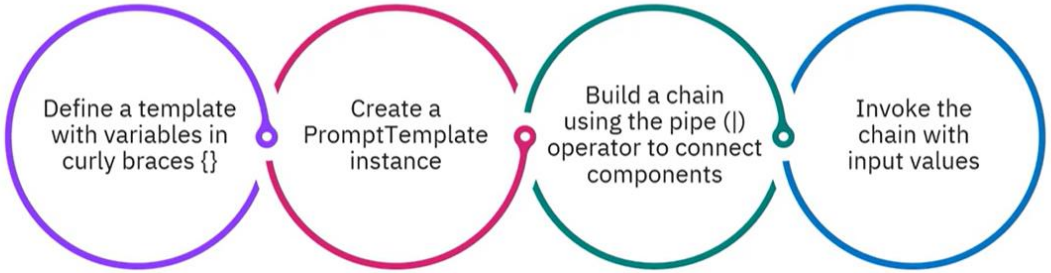
- Builds applications using the pipe (|) operator
- Ensures a clean, readable flow of data

What is LCEL?

The new, recommended LCEL pattern provides:

- Better composability
- Clearer visualization of data flow
- Greater flexibility when constructing complex chains

Create an LCEL pattern



LCEL main Runnable composition primitives

Runnables

- Interface and building blocks
- Connects the following components into a pipeline:
  - LLMs
  - Retrievers
  - Tools

LCEL main Runnable composition primitives

RunnableSequence

- Chains components sequentially
- Passes output from one component as input to the next

```
from langchain_core.runnables import
RunnableSequence


chain = RunnableSequence([runnable1,
runnable2])
```

LCEL main Runnable composition primitives

RunnableParallel

- Runs multiple components concurrently
- Uses the same input

```
from langchain_core.runnables import
RunnableParallel

chain = RunnableParallel({

    "key1": runnable1,

    "key2": runnable2,

})
```

LCEL main Runnable composition primitives

LCEL

- Avoids usage of RunnableSequence
- Connects runnable1 and runnable 2 with a pipe (|)

```
chain = Runnable1 | Runnable2
```

LCEL main Runnable composition primitives

LCEL:

Handles type coercion automatically

Converts regular code into runnable components

Converts dictionaries to RunnableParallel and functions to RunnableLambda

LCEL main Runnable composition primitives

Pipe (|) operator

- Combines the prompt templates with the LLM

Dictionary structure

- Creates a RunnableParallel that processes all three tasks

```
ize this text:
 {text}") | llm,

"translation":
ChatPromptTemplate.from_template("Transl
ate this text to French: {text}") | llm,

"sentiment":
ChatPromptTemplate.from_template(

"What is the sentiment of this text?
Answer with positive, negative, or
neutral: {text}") | llm
```

Creating templates and chains

- Pipe (|) operator creates a sequence by connecting runnable components
- RunnableLambda formats the prompt with variables
- Pipe (|) operator passes the formatted prompt to the LLM
- Another pipe passes the response to the StrOutputParser

```
joke_chain = (

RunnableLambda(format_prompt)

| llm

| StrOutputParser()

)


# Run the chain
```

What next?

We covered:

- Essentials of LCEL
- Benefits and composition primitives

LCEL: Suited for simpler orchestration tasks

LangGraph: Suited for complex workflows

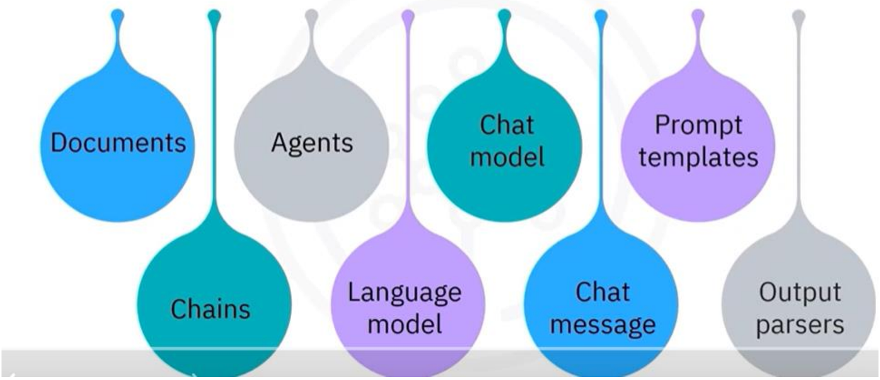
LCEL's strengths:

- Parallel execution
- Async support
- Simplified streaming
- Automatic tracing

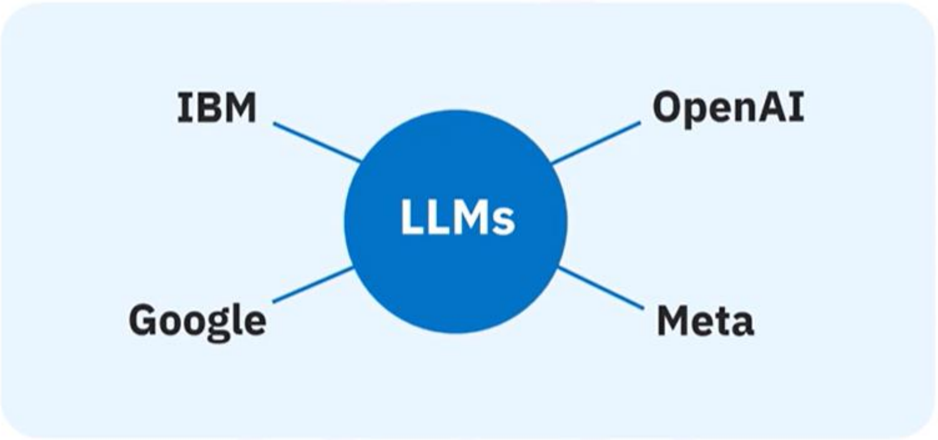
Recap

- LCEL pattern structures workflows use the pipe (|) operator
- Prompts use templates with variables in curly braces {}
- RunnableSequence links components for sequential execution
- RunnableParallel runs multiple components concurrently with the same input
- LCEL simplifies syntax by replacing RunnableSequence with the pipe operator
- Type coercion automatically converts functions and dictionaries into compatible components

Components of LangChain



Introduction



Language model

```
model_id = 'mistralai/mixtral-8x7b-instruct-v01'

parameters =

{GenParams.MAX_NEW_TOKENS: 256, # this controls the maximum number of tokens in
the generated output
    GenParams.TEMPERATURE: 0.5, # this randomness or creativity of the model's
    responses}

credentials = {"url": "https://us-south.ml.cloud.ibm.com"}

project_id = "skills-network"

model = ModelInference (model_id=model_id, params=parameters,
credentials=credentials, project_id=project_id)
```

Chat model

- A type of language model
- Designed for efficient conversations

Chat model

```
model = ModelInference(......)

mixtral_llm = WatsonxLLM (model = model)

print(mixtral_llm.invoke("Who is man's best friend?"))
```

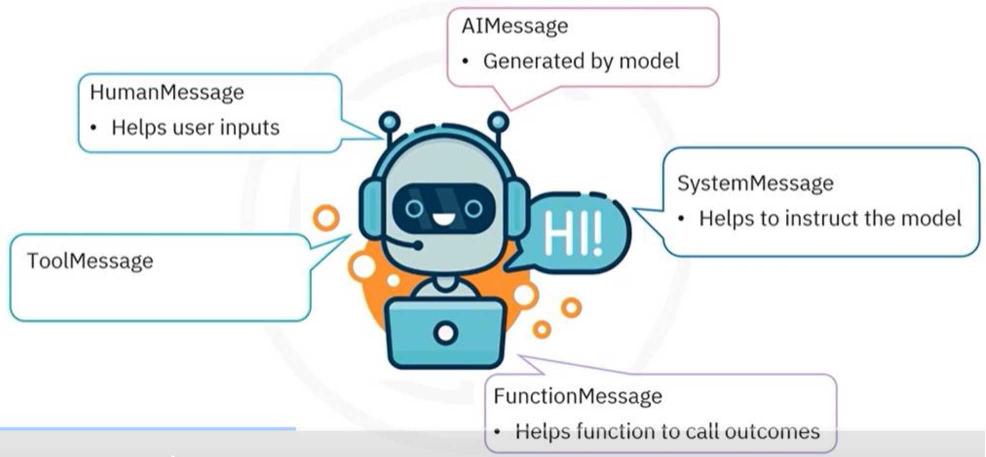
Chat message

```
     role      content
SystemMessage(content ="You are a nice AI bot that helps a user Z
figure out   what to eat in one short sentence")
```

Chat message

```
msg = mixtral_llm.invoke(
   [
     HumanMessage(content="What month follows June?")
   ]
)

print(msg)
```

Chat message

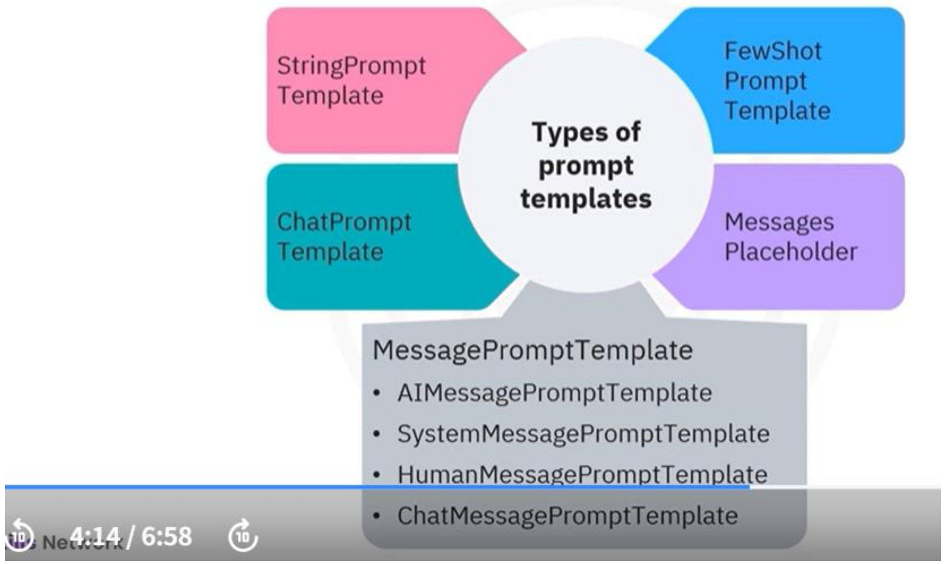


Chat message

```
msg = mixtral_llm.invoke(
    [
        SystemMessage(content="You are a supportive AI bot that
        suggests fitness activities to a user in one short
        sentence"),
        HumanMessage(content="I like high-intensity workouts, what
              should I do?"),
        AIMessage(content="You should try a CrossFit class"),
        HumanMessage(content="How often should I attend?")
    ]
)

print(msg)

AI: Aim to do CrossFit 3 times a week with rest days in between.
```

Prompt templates

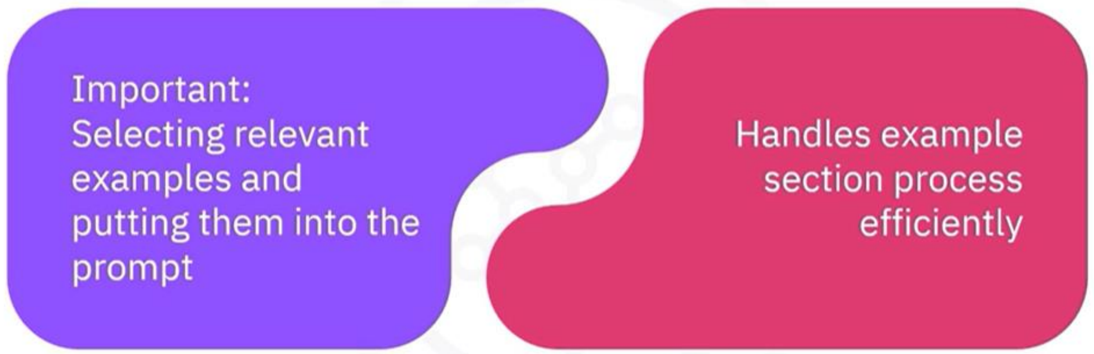
Prompt templates



Prompt templates

```
prompt = ChatPromptTemplate.from_messages([
   ("system", "You are a helpful assistant"),
   ("user", "Tell me a joke about {topic}")
])

input_ = {"topic": "cats"}

prompt.invoke(input_)


ChatPromptValue(messages=[SystemMessage(content='You are a helpful
assistant'), HumanMessage(content='Tell me a joke about cats')])
```

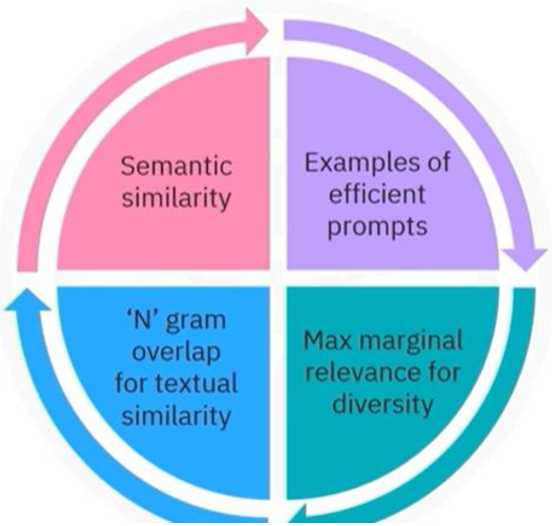
Prompt templates: Example selector



Prompt templates: Example selector

- For example: FewShotPromptTemplate

Prompt templates: Example selector



Prompt templates: Example selector

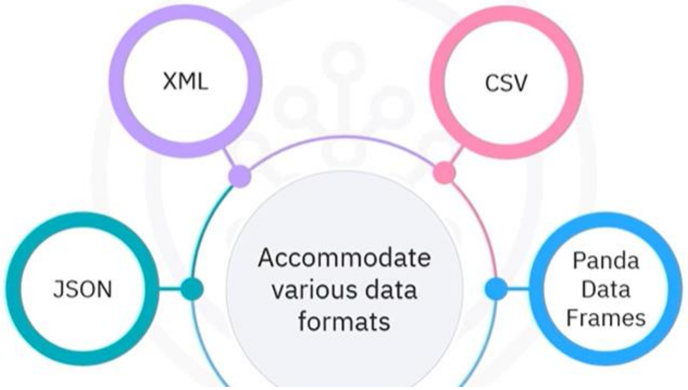
```
From langchain.prompts import PromptTemplate, FewShotPromptTemplate

From langchain.prompts.example_selector import NGramOverlapExampleSelector
```

```
NGramOverlapExampleSelector (
    examples=examples,
    example_prompt=example_prompt,
    threshold=-0.1
)
dynamic_prompt = FewShotPromptTemplate(
    example_selector=example_selector,
    example_prompt=example_prompt,
    prefix="Give the location an item is
    usually found in",
    suffix="Input: {item}\nOutput:",
    input_variables=["item"],
)
```



```
Give the location an item is
usually found in

Example Input: pirate
Example Output: ship

Example Input: pilot
Example Output: plane

Input: plant
Output:
```

Output parsers



Output parsers

```python
from langchain.output_parsers import CommaSeparatedListOutputParser

output_parser = CommaSeparatedListOutputParser()

format_instructions = output_parser.get_format_instructions()
prompt = PromptTemplate(
    template="Answer the user query. {format_instructions}\nList five
    {subject}.",
    input_variables=["subject"],
    partial_variables={"format_instructions": format_instructions},
)

chain = prompt | mixtral_llm | output_parser

chain.invoke({"subject": "ice cream flavors"})

['Chocolate', 'Vanilla', 'Strawberry', 'Mint Chocolate Chip', 'Butter Pecan']
```

Recap

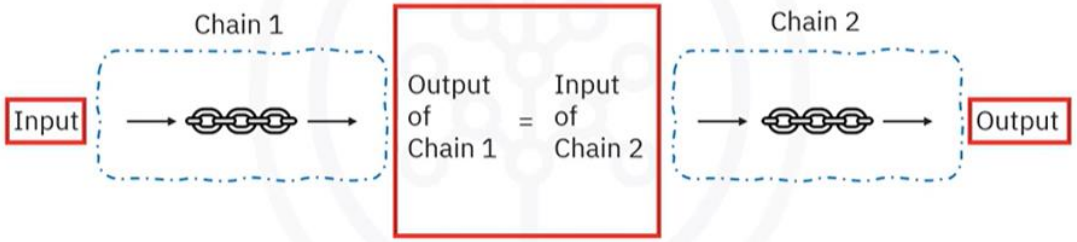
- LangChain is an open-source interface
- Core components of LangChain
    - Language model: Foundation of LLMs
    - Chat models: Designed for efficient conversations
    - Chat messages: Handled by chat models
        - HumanMessage
        - AIMessage
        - SystemMessage
        - FunctionMessage
        - ToolMessage

Recap

- Prompt templates: Translates user's query to clear instructions
- Example selector: Informs model about the input context
- Output parsers: Transform the output from LLM to a suitable format

Chains

Refer to sequences of calls, whether to an LLM, a tool, or a data preprocessing step



Chains

```python
from langchain.chains import LLMChain, SequentialChain

template = """Your job is to come up with a classic dish from the
       area that the users suggests.
       {location}
       YOUR RESPONSE:
"""
prompt_template = PromptTemplate(template=template,
       input_variables=['location'])

# chain 1
location_chain = LLMChain(llm=mixtral_llm, prompt=prompt_template,
       output_key='meal')
```

Chains

```python
template = """Given a meal {meal}, give a short and simple recipe on how
to make that dish at home.

YOUR RESPONSE:
"""
prompt_template = PromptTemplate(template=template,
       input_variables=['meal'])

# chain 2
dish_chain = LLMChain(llm=mixtral_llm, prompt=prompt_template,
       output_key='recipe')
```

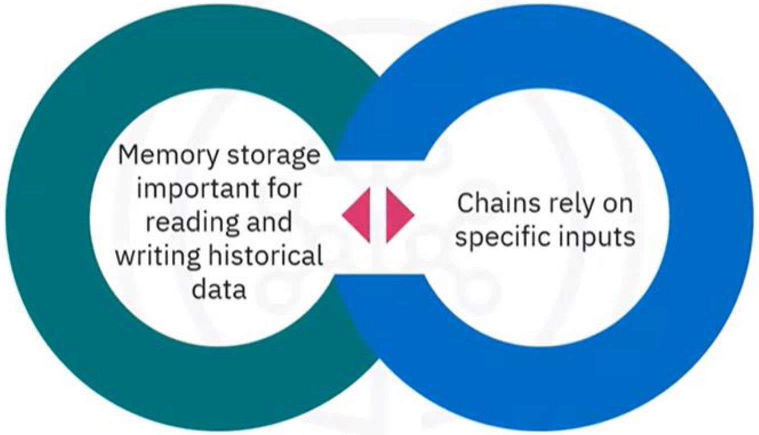
Chains

```python
template = """Given the recipe {recipe}, estimate how much time I need to
cook it.

YOUR RESPONSE:
"""
prompt_template = PromptTemplate(template=template,
       input_variables=['recipe'])

# chain 3
recipe_chain = LLMChain(llm=mixtral_llm, prompt=prompt_template,
       output_key='time')
```

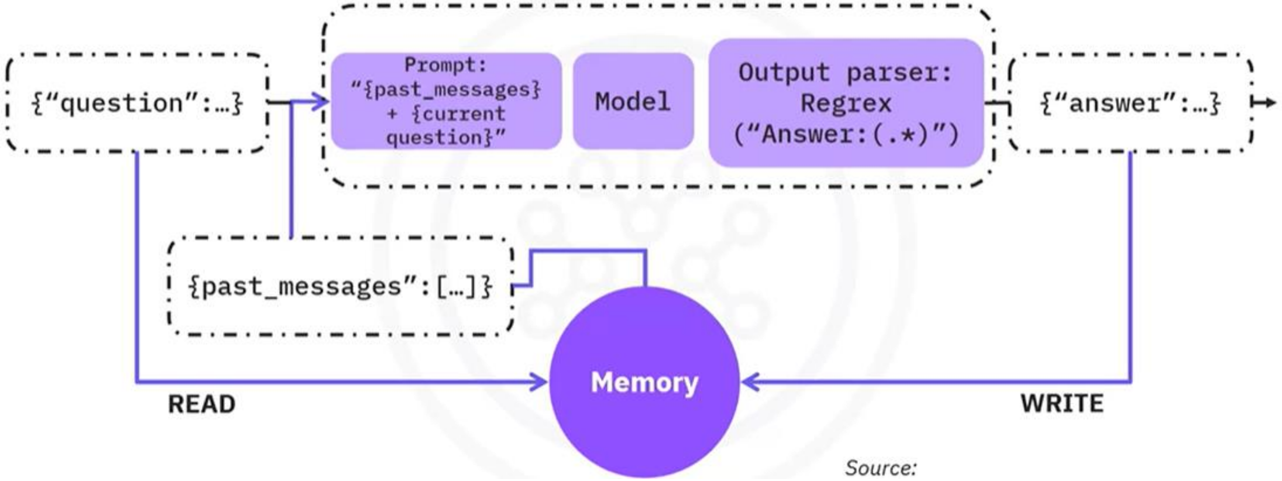
Chains

```
# overall chain
overall_chain = SequentialChain(chains=[location_chain, dish_chain,
        recipe_chain],
input_variables=['location'],
output_variables=['meal', 'recipe', 'time'],
        verbose= True)

overall_chain.invoke(input={'location':'China'})
```

Memory



Memory



Source:

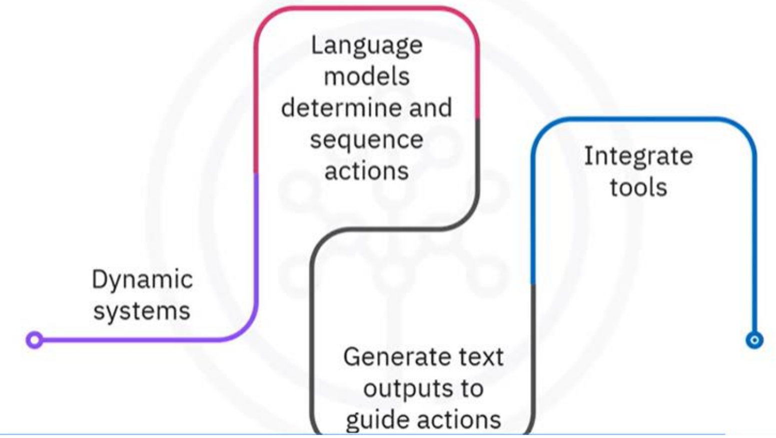
Memory

```
from langchain.memory import ChatMessageHistory

chat = mixtral_llm

history = ChatMessageHistory()

history.add_ai_message("hi!")

history.add_user_message("what is the capital of France?")
```

Memory

```
[AIMessage(content='hi!'),
 HumanMessage(content='what is the capital of France?')]
```

Agents



Agents

Agents

```python
df = pd.read_csv(
      "example.csv"
)

agent = create_pandas_dataframe_agent(
      mixtral_llm,
      df,
      verbose=True,
      return_intermediate_steps=True
)

agent.invoke("How many rows in the dataframe?")
```

There are 139 rows in the dataframe.

Recap

- LangChain is a platform for embedding APIs
- Chains
  - Sequences of calls in LangChain
  - Output from step 1 becomes the input for step 2
  - Defines the template string → Creates prompt templates → Creates LLMChain object name
- Memory
  - Important for reading and writing historical data
- Agents
  - Dynamic systems where the language model determines the sequence of actions
  - Integrate with tools