Skills Network

LangChain: Core Concepts

IBM

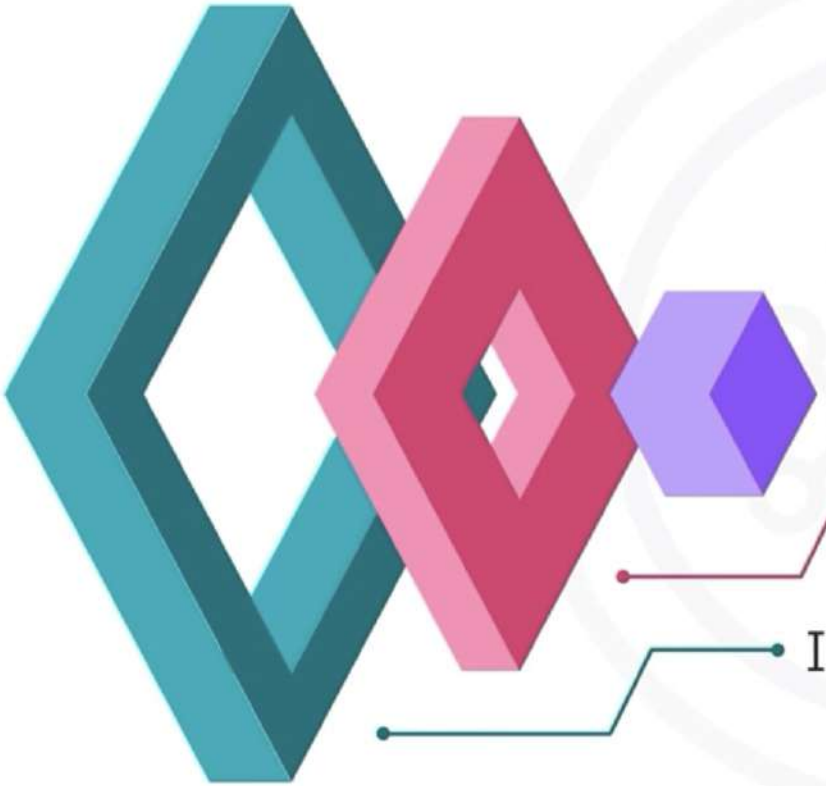
LangChain: Core Concepts

What you will learn

Define LangChain

Describe the components of LangChain

Introduction: LangChain

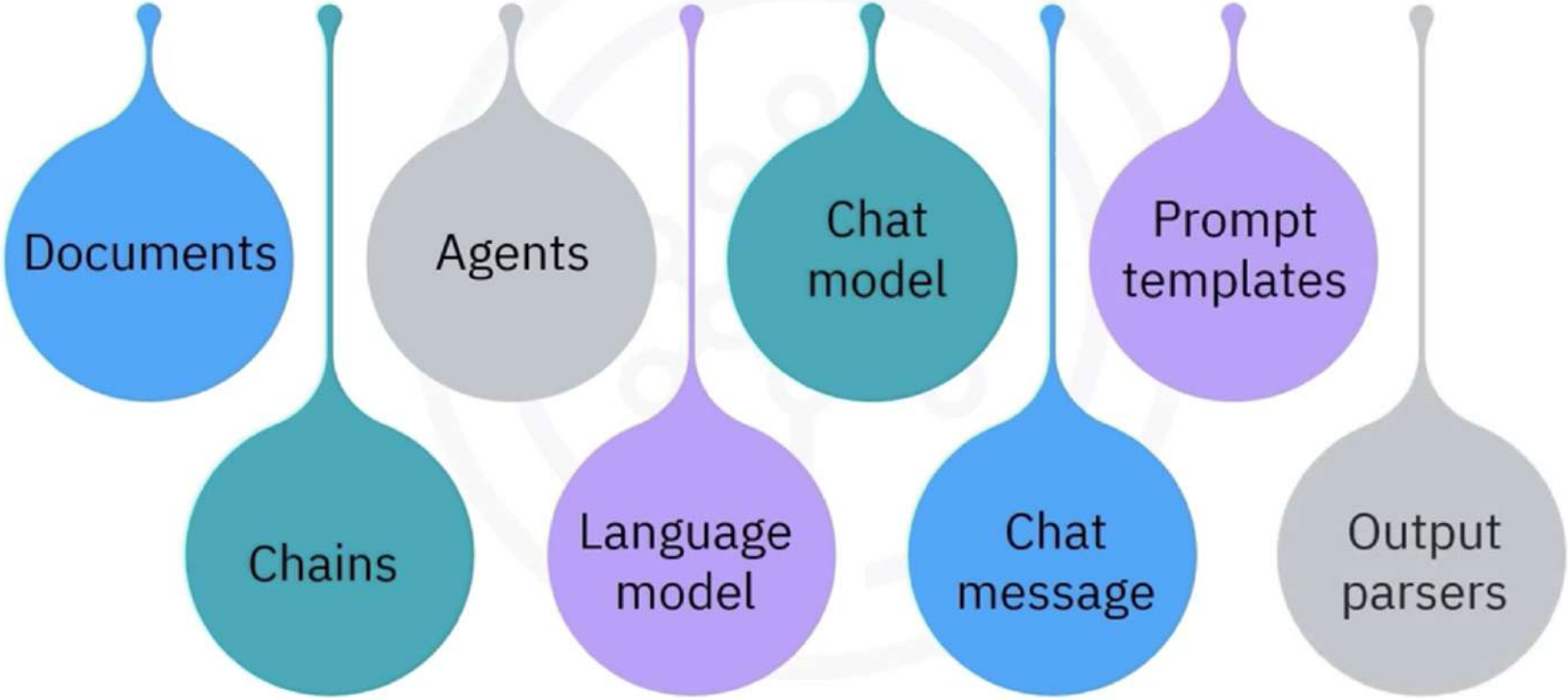


An open-source interface

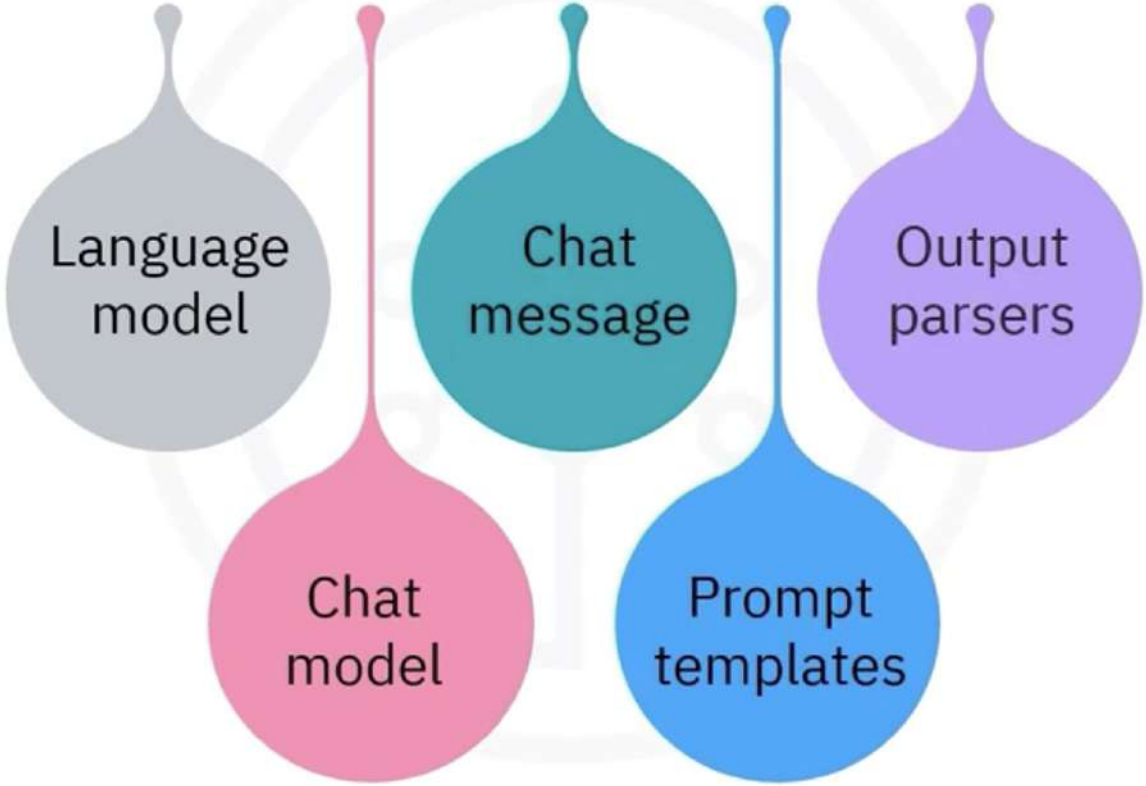
Simplifies application development process

Integrates language models

Components of LangChain



Components of LangChain



Language model

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)
```

Language model

```
{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)

msg = model.generate("In today's sales meeting, we ")

print(msg['results'][0]['generated_text'])
```

Model

```
Agreed to a new approach to sales:

1. We will only sell to those who are interested in our product.
2. We will not sell to anyone who we don't think will benefit from our product.
3. We will not sell to anyone who we think will not use our product.
4. We will not sell to anyone who we think will not pay for our product.

I know this may seem obvious, but I've worked in companies where these rules
were not followed. And I've seen the damage that can be done when a company
sells to the wrong customer.

So, from now on, we will only sell to the right customer. We will not waste our
time or theirs on a sale that is not in their best interest. We will focus on
building long-term relationships with our customers, not just making a quick
sale.

I believe this new approach will lead to more successful sales and happier
customers. And that's what we're all about.
```

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 model

A dog, of course! Dogs are known for their loyalty, intelligence, and affection. They are also known for their ability to help humans in various ways. From guide dogs for the blind to police dogs that help catch criminals, dogs are man's best friend.

But did you know that dogs can also be trained to detect human diseases? That's right! Dogs have an incredible sense of smell and can be trained to detect the subtle changes in a person's scent associated with certain diseases.

Chat message

AIMessage
- Generated by model

HumanMessage
- Helps user inputs

SystemMessage
- Helps to instruct the model

ToolMessage
- Helps in tool interaction

. . . .

FunctionMessage
- Helps function to call outcomes

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

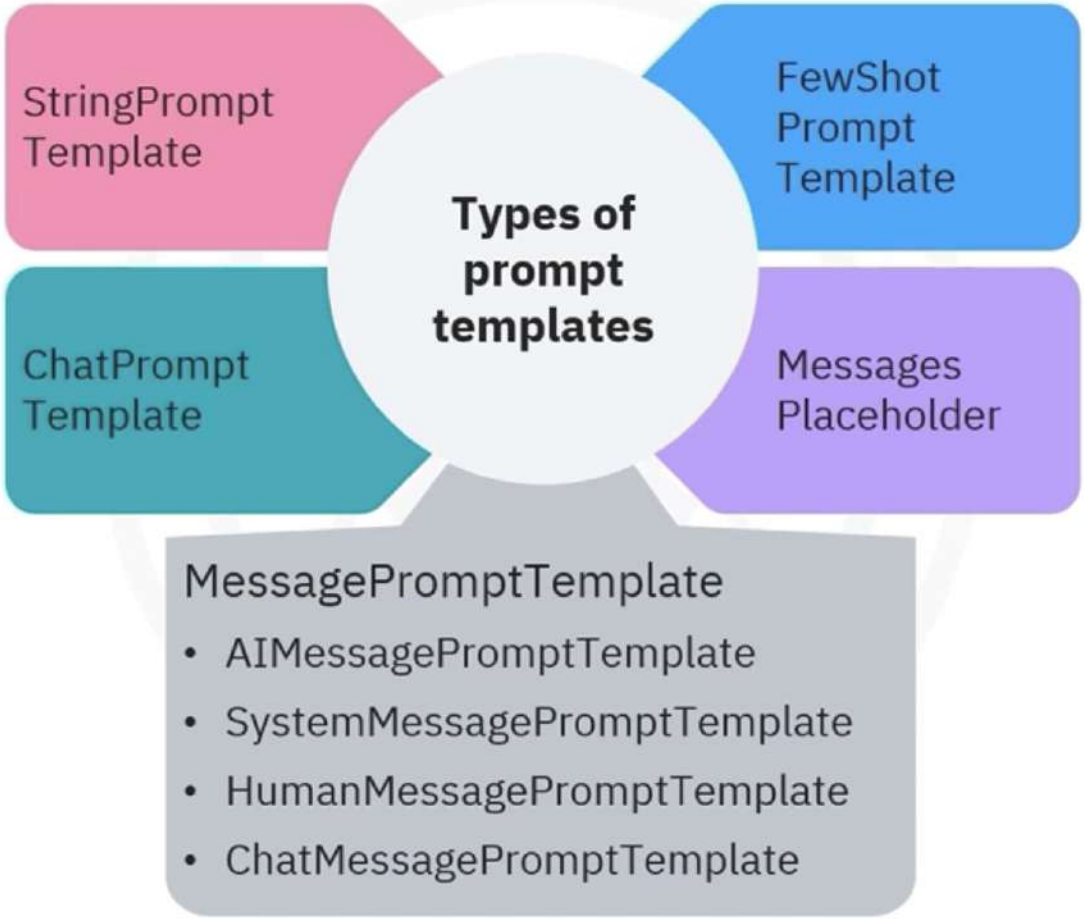
```
msg = mixtral_llm.invoke(
    [
        SystemMessage(content="You are a supportive AI bot that
        suggests fitness activities to a user in one short
        sentence"),
        HumanMessage(content="I like high-intensity workouts, what
              should I do?"),
        AIMessage(content="You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

print(msg)
```

Chat message

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

print(msg)

Assistant: The month that follows June is July.
```

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')])
```

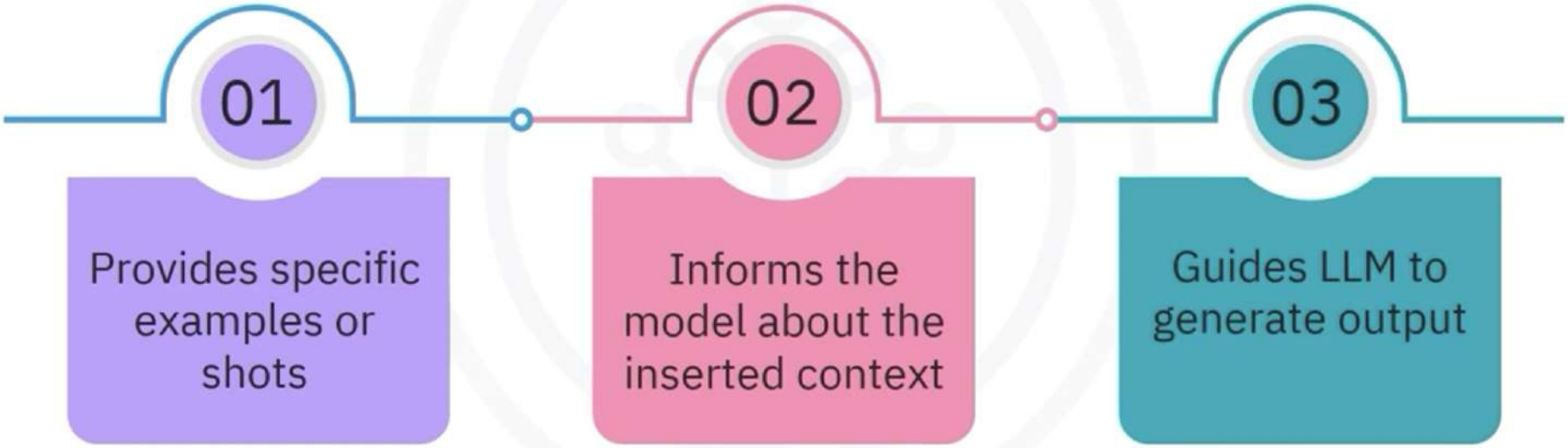
Prompt templates: Example selector

Important:
Selecting relevant examples and putting them into the prompt

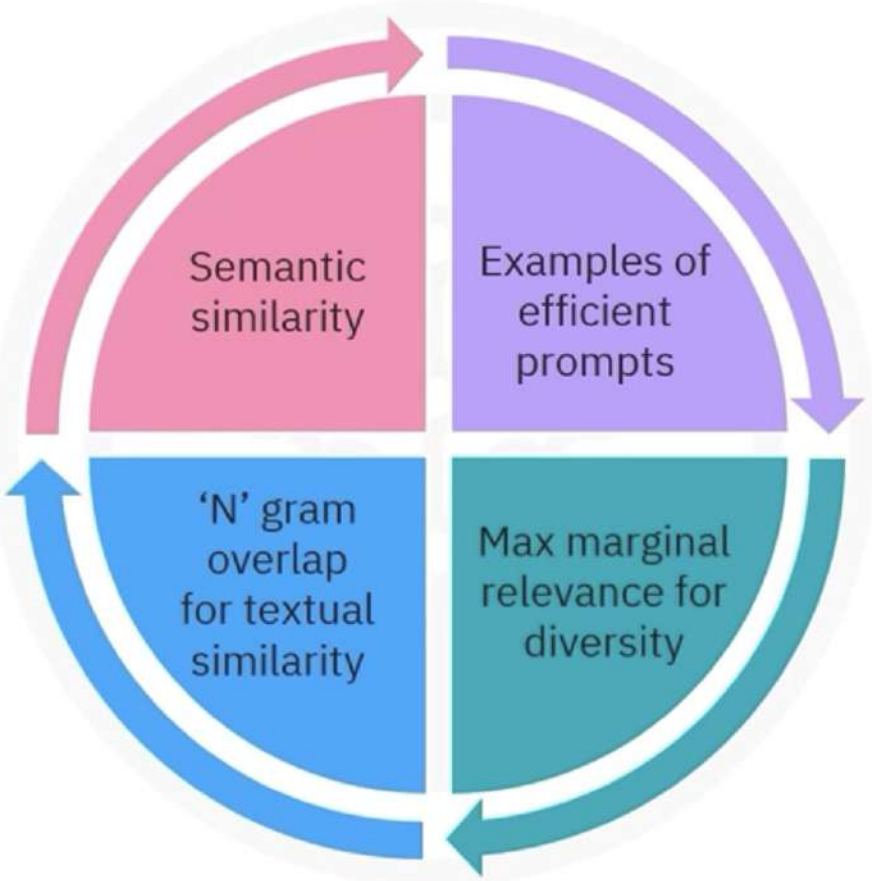
Handles example section process efficiently

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
```

```
examples = [
    {"input": "pirate", "output": "ship"},
    {"input": "pilot", "output": "plane"},
    {"input": "driver", "output": "car"},
    {"input": "tree", "output": "ground"},
    {"input": "bird", "output": "nest"},
]

example_prompt = PromptTemplate(
    input_variables=["input", "output"],
    template="Input: {input}\nOutput:
    {output}",
)
```

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"],
)
```

Prompt templates: Example selector

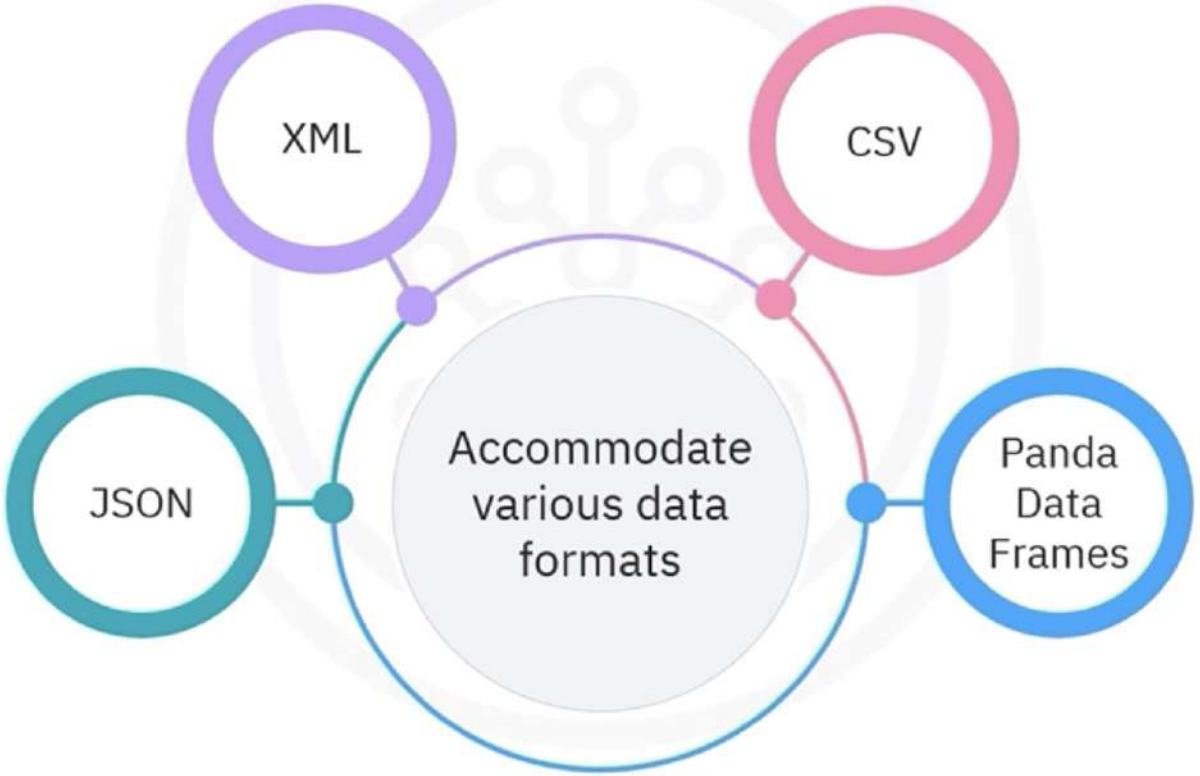
```
From langchain.prompts import PromptTemplate, FewShotPromptTemplate

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



Output parsers

```
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