Generative AI in Cybersecurity

Module 2A: LLMs, Prompt templates, Langchain

Henning Thomsen

hth@ucn.dk

5. May 2025

Pba IT-security @ UCN

Agenda

- Prompting and prompt templates
- Langchain
- Retrieval Augmented Generation (RAG)
  - Afternoon lecture

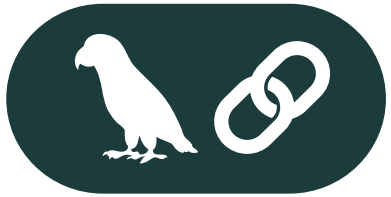
LangChain

Python module for application developent using LLMs and Agents

LangChain

- Framework for Building LLM-Powered Applications
  - https://python.langchain.com
  - Also for JavaScript
- Call LLMs using Python
- Purpose: Use LLMs in Python applications

LangChain libraries

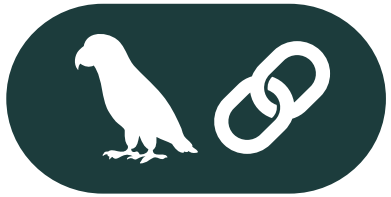


- langchain-core
- langchain-openai, langchain-groq
- langchain
- langchain-community
- Langgraph (next lecture)

LangChain

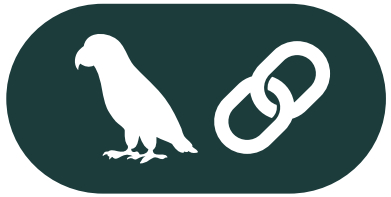
```python
# Define your prompt here, as per exercise text
prompt = """
Explain in max three sentences why one should study Generative AI in Cybersecurity.
"""

# Run OpenAI model
llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)
response = llm.invoke(prompt)
print(response.content)

# Run model via Groq
llm = ChatGroq(model="llama3-8b-8192", temperature=0)
response = llm.invoke(prompt)
print(response.content)
```

LLM constructor



```
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(
    model="gpt-4o",
    temperature=0,
    max_tokens=None,
    timeout=None,
    max_retries=2,
    # api_key="...“
    # base_url="...",
    # organization="...",
    # other params...
)
```

LLM invocation



```
messages = [
    (
        "system",
        "You are a helpful assistant that translates English to French.
             Translate the user sentence.",
    ),
    ("human", "I love programming."),
]
ai_msg = llm.invoke(messages)
ai_msg
```

LLM response (object)

```
AIMessage(content="J'adore la programmation.",
additional_kwargs={'refusal': None}, response_metadata={'token_usage':
{'completion_tokens': 5, 'prompt_tokens': 31, 'total_tokens': 36},
'model_name': 'gpt-4o-2024-05-13', 'system_fingerprint': 'fp_3aa7262c27',
'finish_reason': 'stop', 'logprobs': None}, id='run-63219b22-03e3-4561-
8cc4-78b7c7c3a3ca-0', usage_metadata={'input_tokens': 31, 'output_tokens':
5, 'total_tokens': 36})
```

LLM response (object)

```
AIMessage(content="J'adore la programmation.", additional_kwargs={'refusal':
None}, response_metadata={'token_usage': {'completion_tokens': 5,
'prompt_tokens': 31, 'total_tokens': 36}, 'model_name': 'gpt-4o-2024-05-13',
'system_fingerprint': 'fp_3aa7262c27', 'finish_reason': 'stop', 'logprobs':
None}, id='run-63219b22-03e3-4561-8cc4-78b7c7c3a3ca-0',
usage_metadata={'input_tokens': 31, 'output_tokens': 5, 'total_tokens': 36})
```

LLM response (object)

```
AIMessage(content="J'adore la programmation.", additional_kwargs={'refusal':
None}, response_metadata={'token_usage': {'completion_tokens': 5,
'prompt_tokens': 31, 'total_tokens': 36}, 'model_name': 'gpt-4o-2024-05-13',
'system_fingerprint': 'fp_3aa7262c27', 'finish_reason': 'stop', 'logprobs':
None}, id='run-63219b22-03e3-4561-8cc4-78b7c7c3a3ca-0',
usage_metadata={'input_tokens': 31, 'output_tokens': 5, 'total_tokens': 36})
```

```
print(ai_msg.content)
```

```
J'adore la programmation.
```

Prompt Templates

- Define prompts as templates
  - Go from static prompts to prompts with variables

```
from langchain_core.prompts import ChatPromptTemplate

system_template = (
    "You are a cybersecurity assistant specializing in {specialty}. "
    "Analyze the following input for threats related to {threat_type} and
provide detailed recommendations."
)

prompt_template = ChatPromptTemplate.from_messages(
    [("system", system_template), ("user", "{log_data}")]
)
```

Prompt Templates

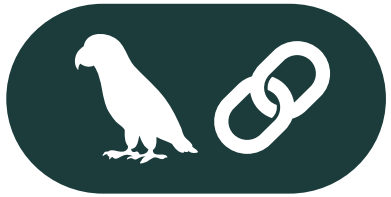
- Variable assignment

```
formatted_prompt = prompt_template.format(
    specialty="cloud infrastructure security",
    threat_type="unauthorized access and privilege escalation",
    log_data="User 'admin' logged in from unknown IP 203.0.113.5 at 03:14 AM..."
)
```

Prompt Templates

- Let’s look at the complete example
  - 02_prompt_templates.py

Spam Classification



- Giving the LLM examples via the prompt (Few-shot examples)

- Workflow
  - Load API-keys
  - Define LLM and its parameters
  - Define prompt template
  - Format the prompt template
  - Invoke LLM

- This is a one-shot operation

Prompt Templates

- Where is the variable?

```
prompt_template = """
Classify the given email message as either spam or legitimate.

Examples are given below:

Message: "Hi Alex, just confirming our meeting tomorrow at 10 AM-let me know if
anything changes."
Classification: Legitimate

Message: "Your account has been compromised-click here immediately to verify your
identity and avoid suspension!"
Classification: Spam

Message: {message}
Classification:
"""
```

Prompt Templates

- Where is the variable?

```
prompt_template = """
Classify the given email message as either spam or legitimate.

Examples are given below:

Message: "Hi Alex, just confirming our meeting tomorrow at 10 AM-let me know if
anything changes."
Classification: Legitimate

Message: "Your account has been compromised-click here immediately to verify your
identity and avoid suspension!"
Classification: Spam

Message: {message}
Classification:
"""
```

```
spam_classification_prompt_template = PromptTemplate(
    input_variables = ["message"],
    template = prompt_template
)
```

Prompt Templates

- The message we want to classify

```
prompt_template = """
Classify the given email message as either spam or legitimate.

Examples are given below:

Message: "Hi Alex, just confirm
changes."
Classification: Legitimate

Message: "Your account has beer
avoid suspension!"
Classification: Spam

Message: {message}
Classification:
"""
```

```
spam_classification_prompt_template = PromptTemplate(
    input_variables = ["message"],
    template = prompt_template
)

spam_classification_prompt =
spam_classification_prompt_template.format(message="Hey.
Nice to see you! Best regards, Rick.")
```

Prompt Templates

- Invocation

```
prompt_template = """
Classify the given email message as either spam or legitimate.

Examples are given below:

Message: "Hi Alex, just confirm
changes."
Classification: Legitimate

Message: "Your account has beer
avoid suspension!"
Classification: Spam

Message: {message}
Classification:
"""
```

```
spam_classification_prompt_template = PromptTemplate(
    input_variables = ["message"],
    template = prompt_template
)

spam_classification_prompt =
spam_classification_prompt_template.format(message="Hey.
Nice to see you! Best regards, Rick.")

result = llm.invoke(spam_classification_prompt)
```

Chat Prompt Templates

- Instead of a plain string, structure the chat session using tuples.

```
prompt_template = """
Classify the given email message as either spam or legitimate.

Examples are given below:

Message: "Hi Alex, just confirming our meeting tomorrow at 10 AM–let me know if
anything changes."
Classification: Legitimate

Message: "Your account has been compromised–click here immediately to verify your
identity and avoid suspension!"
Classification: Spam

Message: {message}
Classification:
"""
```

Chat Prompt Templates

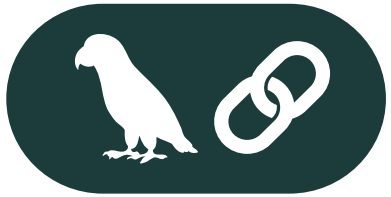
- Instead of a plain string, structure the chat session using tuples.

```
chat_prompt_template = ChatPromptTemplate.from_messages([
    ("system",
     "Classify the given email message as either spam or legitimate."),
    ("human",
     "Message: Hi Alex, just confirming our meeting tomorrow at 10 AM-let me know if
        anything changes."),
    ("ai",
     "Legitimate"),
    ("human",
     "Message: Your account has been compromised-click here immediately to verify
        your identity and avoid suspension!"),
    ("ai",
     "Spam"),
    ("human",
     "Message: {message}")
])
```

Chat Prompt Templates

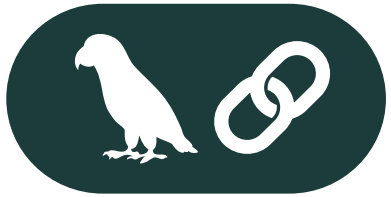
- Why would we want to use ChatPromptTemplate?
  - Better structure and role-tagging
  - Use with memory (relevant for (interactive) chat sessions)
  - Can it reduce risk of prompt injection attacks?

LangChain Expression Language



- Composable Syntax for **chaining** LLM calls, input, parsers etc.
  - Output of one call is input of subsequent call

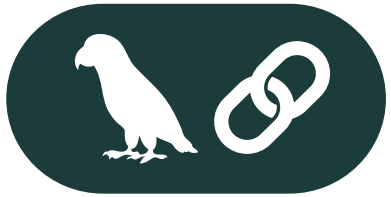
LangChain Expression Language



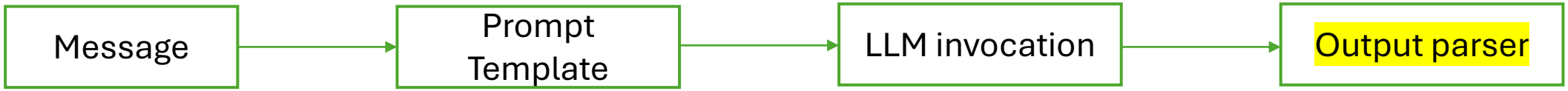
- Composable Syntax for **chaining** LLM calls, input, parsers etc.
  - Output of one call is input of subsequent call


```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate

llm = ChatOpenAI(model="gpt-4o-mini", temperature=0.1)

prompt = ChatPromptTemplate.from_template("What is the capital in {country}?")

chain = prompt | llm

chain.invoke({"country": "Denmark"})
```

LangChain Expression Language



- Output parsing using StrOutputParser




```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers.string import StrOutputParser

llm = ChatOpenAI(model="gpt-4o-mini", temperature=0.1)

prompt = ChatPromptTemplate.from_template("What is the capital in {country}?")

chain = prompt | llm | StrOutputParser()

chain.invoke({"country": "Denmark"})
```