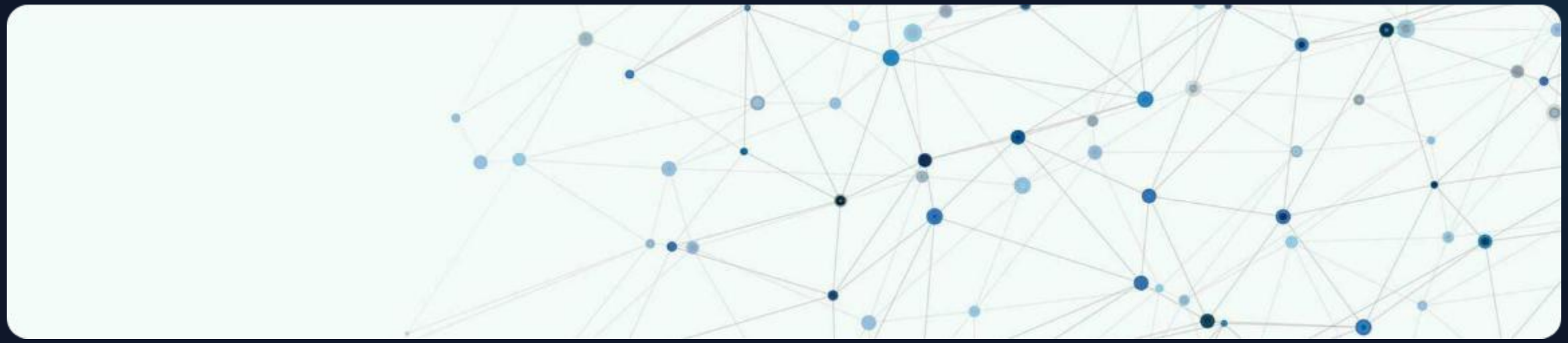


Module 9: LangChain Orchestration

Expression Language (LCEL) & Operational Multi-Step Chains

01. Introduction to LCEL Syntax

A declarative way to compose chains of components.

The Philosophy of LCEL

Composable

Build complex logic by stacking basic building blocks like Lego. Components share a unified interface.

Efficient

Native support for **async**, **streaming**, and **parallel execution** out of the box.

Declarative

Describe *what* the chain should do rather than *how* to program it step-by-step.

Prompt Templates & Schemas

Schema Icon

Structured Input

- ChatPromptTemplate defines the skeleton of the interaction.
- Uses placeholders like {input} to receive runtime variables.
- Decouples the prompt logic from the execution environment.

Core Operational Syntax

The Pipe Operator (|)

Inspired by Unix, the | operator pipes the output of one component directly into the next as an input.

```
chain = prompt | llm | parser
```

The Invoke Method

Executing the chain with a single input dictionary. It handles variable injection and parsing automatically.

```
chain.invoke({"topic": "AI"})
```

The LLM Layer

- **Model Flexibility:** Swap ChatOpenAI for ChatAnthropic without changing the chain structure.
- **Hyperparameter Tuning:** Configure temperature, max_tokens, and model_name directly at instantiation.
- **Message Handling:** The LLM accepts formatted prompt values and returns AIMessage objects.

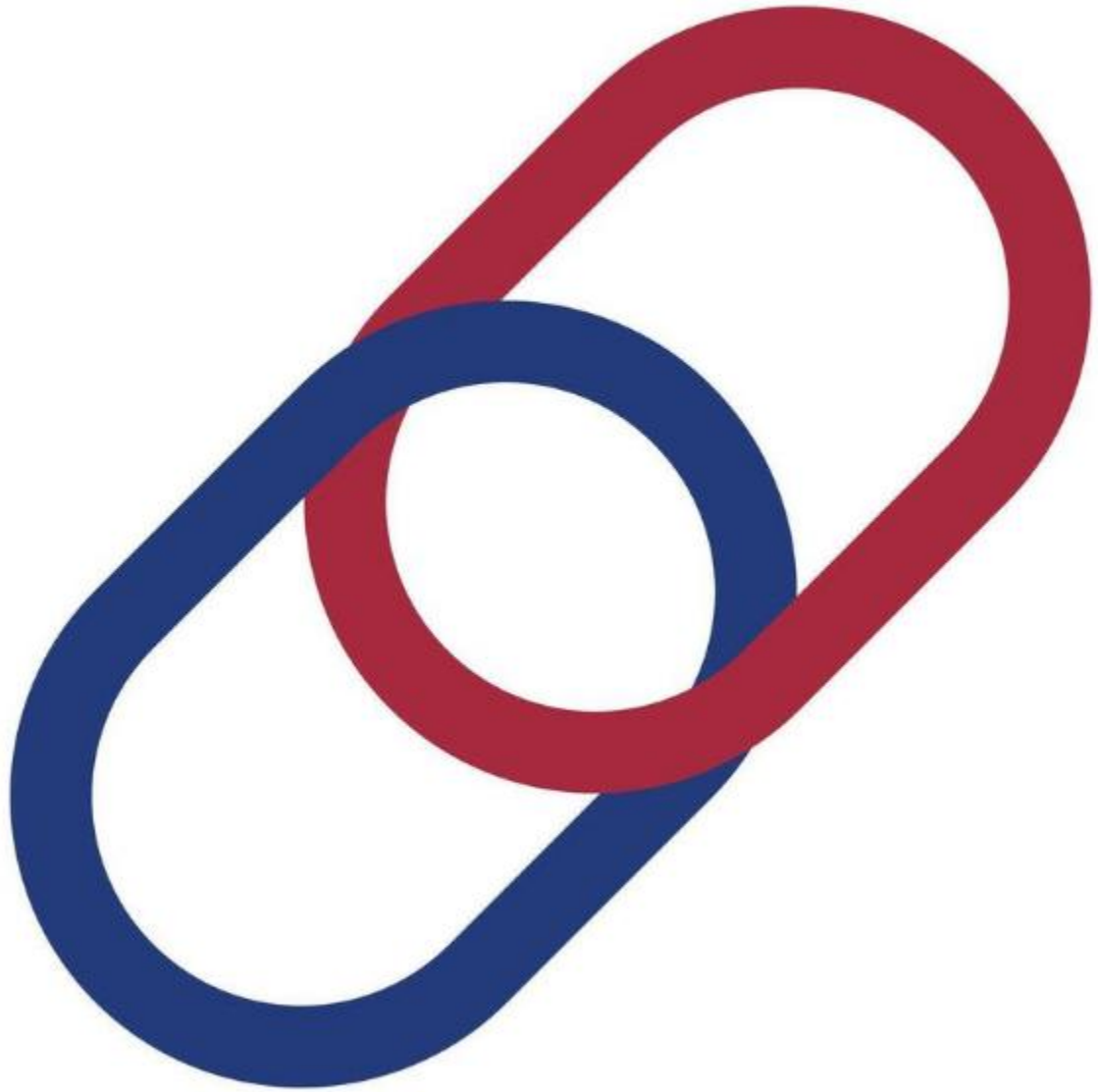
Parsing Response Messages

- **StrOutputParser:** The most common parser for simple text extraction.
- **Object to String:** Strips metadata (like token counts) and returns the core response content.
- **Downstream Ready:** Essential for feeding LLM output into Python scripts or databases.

Multi-Step Chains

Complex Workflows

Operational chains can be nested or linked sequentially. The output of Chain A becomes the context for Chain B. This allows for "Reasoning -> Acting" patterns or automated fact-checking pipelines within a single deployment.



Variable Orchestration

100%

Dynamic Injection

Feeding Runtime Variables

LangChain uses runtime dictionaries to populate prompt templates. This enables bulk processing of data scripts where each row triggers a unique LLM call.

Variable mapping ensures that {variable_name} in the template matches the dictionary key in .invoke().

02. Implementation
Case Study

Building the contextual text generation pipeline.

The Deliverable Pipeline

Data Input

Python script iterates through a dataset (CSV/JSON/SQL).

Chain Execution

chain.invoke() handles the logic for every row.

Automated Output

Contextual text is generated, parsed, and saved back to the script.

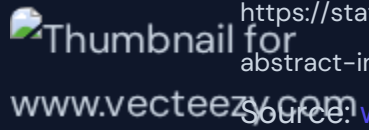
```
chain = prompt | llm | StrOutputParser()
result = chain.invoke({"input": data_row})
```

Questions?

Mastering LangChain Expression Language

Module 9: Advanced Orchestration Completed

Image Sources

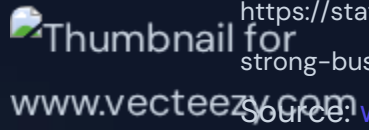


https://static.vecteezy.com/system/resources/previews/025/872/100/non_2x/illustration-molecule-and-internet-connect-technology-on-dark-blue-background-abstract-internet-network-connection-design-for-web-site-digital-data-communication-science-futuristic-concept-vector.jpg

Source: www.vecteezy.com

https://moudjadj.com/wp-content/uploads/2026/04/ai-prompt-expert-web-development.webp

Source: moudjadj.com



https://static.vecteezy.com/system/resources/previews/071/274/200/non_2x/simple-icon-of-two-interlocked-chain-links-a-symbol-for-connection-partnership-and-a-strong-business-relationship-vector.jpg

Source: www.vecteezy.com