CHAINS

MUKESH KUMAR

Introduction to Chains

- A chain in LangChain is a sequence of modular components (prompts, models, parsers, etc.)
- Helps manage multi-step reasoning or data transformation workflows
- Chains allow you to build complex logic from simple building blocks

Why Use Chains?

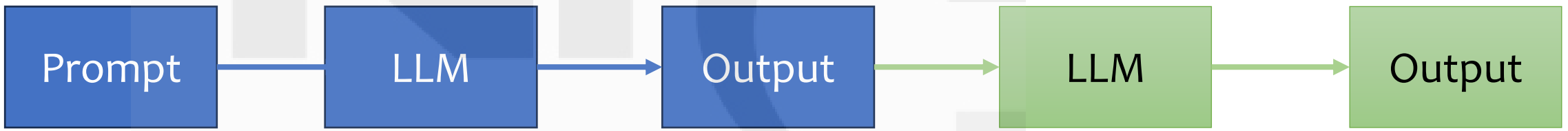
- **Benefits:**
- Compose multiple components (prompts, models, tools)
- Control flow and logic in LLM pipelines
- Enable modular design for complex tasks
- **Challenges Solved:**
- Repetitive prompting
- Intermediate result handling
- Complex workflows with conditionals

Overview of Chain Types

- **Simple Chain -** Linear execution, no mapping
- **Sequential Chain** – Chained with input/output mappings
- **Conditional Chain** – Executes based on conditions
- **Parallel Chain** – Executes multiple chains in parallel

Simple Chains

Sequential Chains



Sequential Chains Example

- First, generate a detailed product description based on a product name,

- then create a short FAQ section from that description to assist customer support or ecommerce listings.

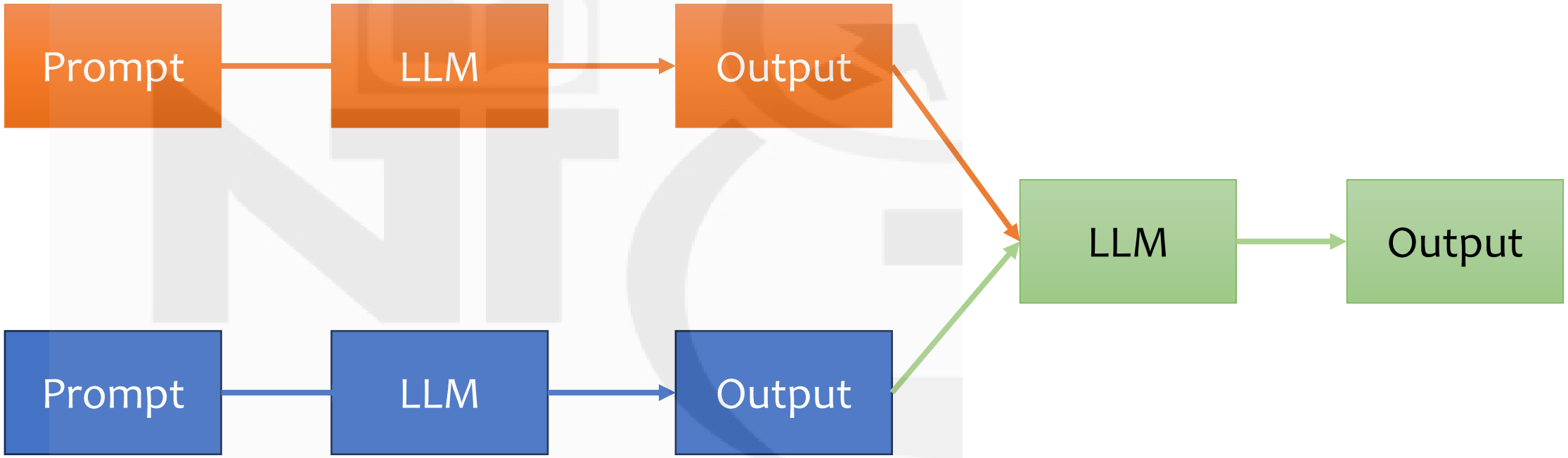
```
# Step 1: Generate product description
description_prompt = PromptTemplate(
    template="Write a comprehensive product description for the product named: {product_name}",
    input_variables=["product_name"]
)

# Step 2: Generate FAQ from product description
faq_prompt = PromptTemplate(
    template="Based on the following product description, generate 4 frequently asked questions and answers:\n\n{description}",
    input_variables=["description"]
)
```

Sequential Chains Example

Print the chain

```
chain.get_graph().print_ascii()
```

Parallel Chains

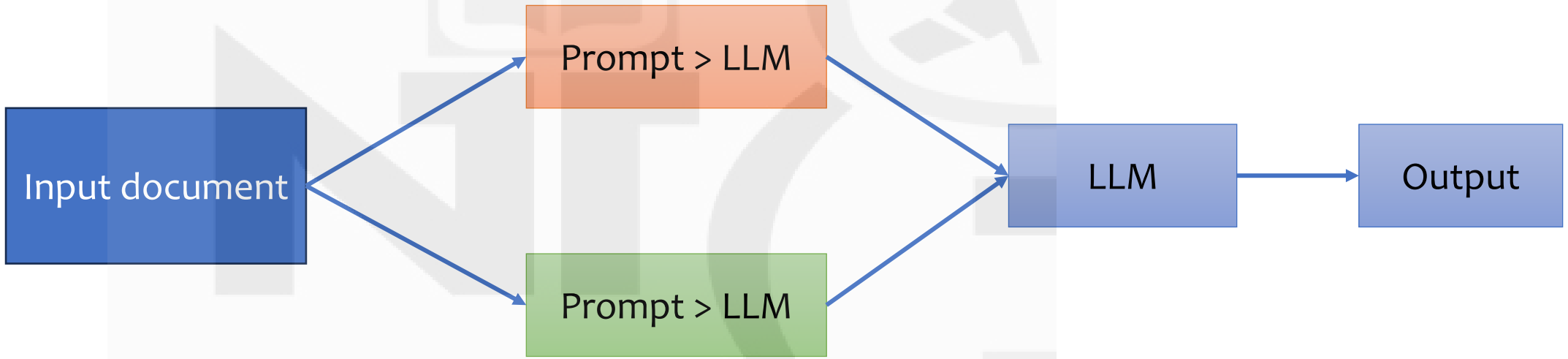


Parallel Chains Example

This example demonstrates a pipeline that takes a block of technical or educational text and automatically generates two key outputs in parallel:

- A tweet-sized summary for quick sharing on social media.
- A list of keywords that can be used for SEO, tagging, or categorization.
- The results are then merged into a well-formatted content block suitable for blog posts, newsletters, or documentation.
- This setup helps automate content repurposing, making it easier for creators, marketers, or educators to generate high-quality promotional and metadata content instantly.

Parallel Chains Example



Parallel Chains Example

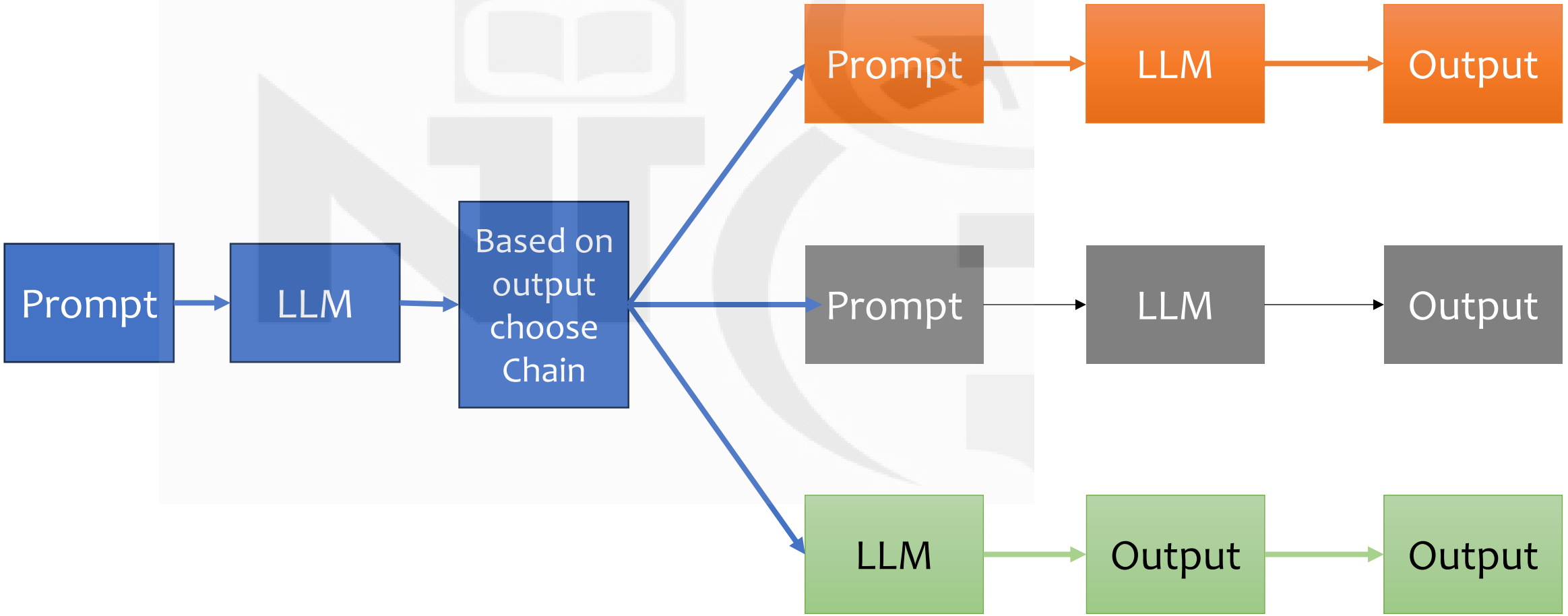
- To run parallel chains, we use Runnable parallel
- It takes a dictionary of chains.

```
from langchain.schema.runnable import RunnableParallel
```

Creating parallel Chains in langchain

```
# Run keyword and summary generation in parallel
parallel = RunnableParallel({
    'keywords': keyword_prompt | claude_model | parser,
    'summary': tweet_prompt | openai_model | parser
})

# Combine outputs
final_chain = final_prompt | openai_model | parser

# Compose the full chain
pipeline = parallel | final_chain
```

Conditional Chain



Conditional Chain Example

- This example automatically analyzes incoming customer support tickets to determine whether the issue is urgent or normal.
- Based on the urgency, it generates an appropriate response:
  - either quick and reassuring for urgent issues or
  - polite and informative for regular ones.

- It uses a conditional chain that first classifies the urgency using a language model
- and then branches the response generation logic accordingly.
- This kind of automation can significantly reduce response time and improve customer satisfaction.

Creating conditional branches

- Each conditional chain is provided as a tuple in runnablebranch
- And at the end if none of the conditions match the default chain is executed
- It more like if elif … else