LangChain Runnables

Understanding Task-Specific and Primitive Runnables in LangChain

Overview of LangChain Runnables

Runnables are the fundamental abstraction in LangChain for representing any callable computation block that can be composed, executed, and reused in a structured way.

Each Runnable defines a standardized interface with methods such as:

- `invoke(input)` — for synchronous execution
- `ainvoke(input)` — for asynchronous execution
- `batch(inputs)` — for batch processing
- `stream(input)` — for streaming outputs

Runnables enable the creation of modular, composable data-processing or reasoning pipelines.

Two main categories of Runnables:

1. **Task-Specific Runnables**
2. **Runnable Primitives**

Task-Specific Runnables

These are higher-level Runnables tailored to specific operations in the LangChain ecosystem such as LLM calls, prompt formatting, output parsing, and embeddings.

Examples of Task-Specific Runnables

- **ChatModelRunnable:** Wraps an LLM or chat model (e.g., OpenAI, Gemini) for inference.
- **PromptTemplate:** A Runnable that formats text with placeholders for dynamic inputs.
- **OutputParser:** Converts raw LLM output into structured formats (string, JSON, or Pydantic models).
- **Retriever:** Fetches relevant documents or data from a knowledge base.

- **Embeddings:** Encodes text into vector space for similarity search.

Each of these implements the `Runnable` interface, allowing them to be seamlessly composed into larger pipelines.

Example

```
prompt = PromptTemplate(
    template="Tell me a joke about {topic}",
    input_variables=["topic"]
)

model = ChatGoogleGenerativeAI(model="gemini-2.5-flash")
parser = StrOutputParser()

chain = prompt | model | parser
result = chain.invoke({"topic": "cats"})
```

Runnable Primitives

Runnable Primitives are low-level components used to build complex workflows by controlling the flow of data between Runnables.

1. RunnableSequence

Purpose: Execute multiple Runnables sequentially, passing the output of one as the input to the next.

- **Old Syntax (deprecated):** `RunnableSequence([r1, r2, r3])`
- **Modern Syntax:** Use the | operator: `r1 | r2 | r3`

Example

```
chain = prompt | model | parser
result = chain.invoke({"topic": "AI"})
```

2. RunnableParallel

Purpose: Run multiple branches of computation in parallel, returning a dictionary of results.

Example

```
parallel_chain = RunnableParallel({
    "tweet": prompt1 | model | parser,
    "linkedin": prompt2 | model | parser
})
result = parallel_chain.invoke({"topic": "AI"})
```

3. RunnableBranch

Purpose: Execute conditional logic (like an if-else). It evaluates predicates and routes input to the first branch whose condition is true.

Example

```
branch = RunnableBranch(
    (lambda x: "joke" in x, joke_chain),
    (lambda x: "news" in x, news_chain),
    default_chain
)
```

4. RunnablePassthrough

Purpose: Returns the input as output. Useful for testing or combining with parallel pipelines.

Example

```
from langchain_core.runnables import RunnablePassthrough

chain = RunnablePassthrough()
result = chain.invoke("Hello")
# Output: "Hello"
```

5. RunnableLambda

Purpose: Allows embedding arbitrary Python logic in the chain.

Example

```
from langchain_core.runnables import RunnableLambda

lambda_chain = RunnableLambda(lambda x: x.upper())
result = lambda_chain.invoke("hello")
# Output: "HELLO"
```

6. RunnableMap

Purpose: Applies a chain to each element of a list (like Python's map function).

Example

```
chain = prompt | model | parser
mapper = RunnableMap(chain)
results = mapper.invoke(["AI", "cats", "space"])
```

Comparison Table

Runnable Type	**Purpose**	**Example Use Case**
RunnableSequence	Sequential execution	Prompt → Model → Parser
RunnableParallel	Run multiple tasks at once	Generate Tweet & LinkedIn posts
RunnableBranch	Conditional routing	If input has keyword "joke", use joke chain
RunnablePassthrough	Identity function	Forward input unchanged
RunnableLambda	Embed custom logic	Format or preprocess data inline
RunnableMap	Apply chain to list	Run model on multiple inputs

Conclusion

LangChain Runnables provide a powerful way to compose LLM pipelines using modular and reusable building blocks. Understanding both task-specific and primitive Runnables allows developers to design dynamic, parallel, and conditional AI workflows with ease.

Recommended Learning Path:

1. Start with simple `RunnableSequence` compositions.

2. Explore `RunnableParallel` for multi-output workflows.

3. Use `RunnableBranch` and `RunnableLambda` for intelligent control flow.