LangChain Learning Notes

Understanding the Motivation Behind LangChain and Runnables

Introduction

LangChain is an open-source framework developed to help engineers build powerful applications powered by **Large Language Models (LLMs)**. Its primary goal is to provide an abstraction layer that allows developers to integrate and compose various LLMs seamlessly, without worrying about model-specific syntax or API differences.

Evolution of LangChain

At the beginning, the LangChain community aimed to simplify working with multiple LLM APIs such as **OpenAI, Anthropic, and Gemini**. Each LLM had its own API structure and syntax, which made direct integration difficult and inconsistent.

To solve this, LangChain introduced a modular pipeline system made up of different components:

- **Document Loaders** – to read and load raw data or documents.
- **Text Splitters & Parsers** – to process and structure the data.
- **Prompt Templates** – to dynamically format user prompts.
- **Vector Databases (Vector DBs)** – to store and retrieve text embeddings efficiently.

While this component-based architecture helped developers create end-to-end pipelines, many common tasks (like sending prompts or retrieving relevant documents) were repetitive across LLM apps.

Motivation Behind Chains

To automate these recurring patterns, LangChain introduced the concept of **Chains**. Chains connected components such as prompt templates, models, and retrievers into a single, reusable flow. Examples include:

- Prompt creation and LLM response generation
- Retrieval-Augmented Generation (RAG) pipelines
- Sequential or parallel model execution

Although Chains improved productivity, they introduced new challenges:

1. **Too Many Chain Types:** Each task or use case led to the creation of a new chain, increasing the codebase and maintenance overhead.
2. **Different Syntax Across Components:** For example:
   - LLMs use `predict()`
   - Prompt templates use `format()`
   - Retrievers use `get_relevant_documents()`

   This inconsistency made it harder to combine components.
3. **Custom Wrappers Required:** Developers had to write additional code to connect incompatible components.
4. **Learning Curve Increased:** Understanding and debugging numerous chain types became increasingly difficult.

These issues highlighted the need for a more unified and modular approach.

Introduction of the Runnable Interface

To address these limitations, the LangChain team introduced the **Runnable Interface**. Runnables provide a standardized and composable way to connect any component — whether it's a prompt, model, retriever, or parser — under a single, consistent interface.

Key Advantages of Runnables

- **Unified Syntax:** All components now share the same input-output method structure.
- **Composable Workflows:** Developers can easily chain multiple Runnables together.
- **Cross-Compatibility:** No more need for custom wrappers between different components.
- **Reduced Boilerplate:** Common logic (like calling, streaming, or batching) is handled automatically.
- **Simplified Debugging:** Clear, consistent execution flow makes testing and maintenance easier.

Runnable Methods and Functionality (LangChain v0.1+)

In LangChain v0.1 and above, most Chains and Tools have been re-implemented internally using the **Runnable Interface**. This interface introduces three core methods that unify how all components are executed:

- **invoke():** Executes a single input synchronously and returns the result.
- **batch():** Processes multiple inputs in parallel for higher throughput.
- **stream():** Streams partial outputs progressively, allowing real-time response handling.

Because all components follow the same pattern, **Runnables are fully compatible with each other**, making it simple to combine any LLM, retriever, or parser into one coherent workflow.

Conclusion

The transition from traditional Chains to **Runnables** marks a major evolution in LangChain's design philosophy. It unifies syntax across all components, simplifies development, enhances composability, and reduces maintenance complexity.

By using the Runnable Interface, developers can now focus on building logic and features instead of managing individual model behaviors. Overall, Runnables form the foundation for the next generation of modular, efficient, and scalable LLM applications.