

Cheat Sheet: LangChain Expression Language (LCEL)

Estimated time: 5 minutes

LangChain Expression Language (LCEL) is a declarative method for assembling chains from modular components within the LangChain framework. Rather than prescribing step-by-step instructions, LCEL allows you to specify the outcome, enabling LangChain to optimize its code.

This cheat sheet provides an overview of LCEL's key capabilities, patterns, and orchestration strategies.

Now let's explore some LCEL capabilities and their related benefits.

| Capability | Benefits |
|--|---|
| Run optimized parallel execution | Reduces latency and increases overall application performance by running components concurrently. |
| Use guaranteed async support | Enables smooth, non-blocking workflows that improve responsiveness and throughput in complex chains. |
| Stream outputs incrementally | Delivers immediate feedback to users, lowers perceived latency, and monitors progress through each stage of the pipeline. |
| Trace all steps automatically with LangSmith | Provides visibility into your chain's behavior to quickly debug, monitor performance, and improve reliability. |
| Call all chains through a shared LCEL API | Simplifies integration and enables consistent behavior across workflows. |
| Deploy chains with LangServe | Accelerates the transition from development to production with minimal overhead. |
| Uses concise and expressive syntax | Eases connectivity among components for building robust data pipelines. |

Runnables

Each component conforms to the LCEL standardized Runnable interface.

Runnables represent any component that can transform input into output. The Runnable interface provides a standard set of methods (like `invoke()`, `batch()`, `stream()`) that all components implement, making them interoperable. The following table lists some runnable types, provides a brief description of the runnable type, and lists an example use case.

| Runnable type | Description | Example use case |
|-------------------------------|--|--|
| <code>ChatModel</code> | Interfaces with LLM APIs for chat | Generate conversational responses |
| <code>PromptTemplate</code> | Creates formatted prompts from variables | Prepare structured inputs for LLMs |
| <code>OutputParser</code> | Converts raw outputs to structured data | Extract structured data from LLM responses |
| <code>RunnableLambda</code> | Wraps custom Python functions | Implement custom business logic |
| <code>RunnableSequence</code> | Chains multiple Runnables together | Create multi-step processing pipelines |

Next, learn about using runnable chains.

Runnable chains: Use cases, components, and workflows.

You can build runnable chains by connecting components from LangChain's standardized Runnable interface. Each component passes its output directly to the next component, creating a workflow.

Next, review some runnable chain use cases, their components, and their workflow summaries.

| Use cases | Components | Workflow summary |
|--------------------------------------|---|--|
| Simple question answering | <code>PromptTemplate</code> → <code>ChatModel</code> → <code>StrOutputParser</code> | Format a question, send to LLM, return text response |
| Retrieval augmented generation (RAG) | <code>Retriever</code> → <code>PromptTemplate</code> → <code>ChatModel</code> → <code>OutputParser</code> | Find relevant documents, combine with prompt, generate response |
| Function calling | <code>PromptTemplate</code> → <code>ChatModel</code> → <code>Tool</code> | Format prompt, generate function call, parse function parameters, execute function |
| Structured output | <code>PromptTemplate</code> → <code>ChatModel</code> → <code>JsonOutputParser</code> | Format prompt, generate response, parse to JSON object |

LCEL function reference

LCEL provides a rich set of functions to invoke, compose, and configure Runnables. This reference section categorizes LCEL functions into basic operations, composition techniques, data manipulation, advanced patterns, configuration tools, and streaming/batching support. Each function enhances how you structure and control your chains. It's important to note that functions that begin with the letter `a` are asynchronous functions.

Basic operations

| Function | Description | Usage |
|----------|-------------|-------|
|----------|-------------|-------|

| | | |
|--|---|---|
| <code>invoke()</code> / <code>ainvoke()</code> | Execute a Runnable with a single input | Process one input and get one output |
| <code>batch()</code> / <code>abatch()</code> | Process multiple inputs efficiently in parallel | Run the same operation on multiple inputs at once |
| <code>stream()</code> / <code>astream()</code> | Return incremental results as they're generated | Show partial responses as they're created |

Composition patterns

| Function | Description | Usage |
|--|--|--|
| Pipe operator <code> </code> or <code>.pipe()</code> | Create a sequence of Runnables | Chain components where output of one becomes input to the next |
| <code>RunnableParallel</code> | Execute multiple Runnables with the same input, concurrently | Process the same input in different ways simultaneously |
| <code>RunnableLambda</code> | Convert Python functions into Runnables | Add custom logic within a chain |

Data manipulation patterns

| Function | Description | Usage |
|---|---|--|
| <code>RunnablePassthrough.assign()</code> | Add new fields to the input dictionary | Augment input with additional data while preserving the original input |
| <code>RunnablePassthrough()</code> | Return input unchanged | Include the original input as part of the output |
| <code>.pick()</code> | Select specific keys from the dictionary output | Filter output to only needed fields |

Advanced patterns

| Function | Description | Usage |
|--------------------------------|--|--|
| <code>.bind()</code> | Set default values for parameters | Fix certain parameters while leaving others configurable |
| <code>.with_fallbacks()</code> | Try alternative Runnables if the primary fails | Handle errors by providing backup components |
| <code>.with_retry()</code> | Add automatic retry capability | Retry operations on failure For example: Network issues |

Configuration

| Function | Description | Usage |
|-------------------------------|--|---|
| <code>config</code> parameter | Control runtime execution | Pass to <code>invoke()</code> , <code>batch()</code> , <code>stream()</code> methods with settings like concurrency limits and tracing parameters |
| <code>.with_config()</code> | Create Runnable with default configuration | Apply the same configuration to all invocations automatically |

Streaming and batching

| Function | Description | Usage |
|--|---|--|
| <code>astream_events()</code> | Detailed stream of execution events | Monitor the entire execution process, including intermediate steps |
| <code>batch_as_completed()</code> / <code>abatch_as_completed()</code> | Process inputs in parallel, return as completed | Start processing results as soon as they're available |

Next, explore some recommended approaches to orchestration.

Recommended approaches to orchestration

When building LLM applications, you can choose the orchestration method based on the complexity of your use case. The following table provides some guidance for when to use a direct call, LCEL, or LangGraph.

| Use case | When to use | Recommended tool |
|------------------------------|---|------------------|
| Single LLM call | <ul style="list-style-type: none">You need to generate text from a promptThe overhead of setting up a chain is not justified | LLM directly |
| Simple chains | <ul style="list-style-type: none">You need a straightforward pipeline of components. For example: A prompt + LLM + parser + basic retrievalYour application can benefit from parallelized code and streamingYour application has a clear linear flow with minimal branching | LCEL |
| Complex logic with branching | <ul style="list-style-type: none">Your application requires complex state managementYour application requires conditional flows, loops, or cyclesYou are implementing multi-agent systems that interact with each other | LangGraph |

Author(s)

IBM Skills Network Team



Skills Network