



# Evaluation Quick Start

Evaluations are a quantitative way to measure performance of LLM applications, which is important because LLMs don't always behave predictably — small changes in prompts, models, or inputs can significantly impact results. Evaluations provide a structured way to identify failures, compare changes across different versions of your application, and build more reliable AI applications.

Evaluations are made up of three components:

1. A [dataset](#) with test inputs and optionally expected outputs.
2. A [target function](#) that defines what you're evaluating. For example, this may be one LLM call that includes the new prompt you are testing, a part of your application or your end to end application.
3. [Evaluators](#) that score your target function's outputs.

This quick start guides you through running a simple evaluation to test the correctness of LLM responses with the LangSmith SDK or UI.

**SDK**   **UI**

## TIP

This quickstart uses prebuilt LLM-as-judge evaluators from the open-source [openevals](#) package. OpenEvals includes a set of commonly used evaluators and is a great starting point if you're new to evaluations. If you want greater flexibility in how you evaluate your apps, you can also [define completely custom evaluators using your own code](#).

## 1. Install Dependencies

**Python**   **TypeScript**

```
npm install langsmith openevals openai
```

**! INFO**

If you are using `yarn` as your package manager, you will also need to manually install `@langchain/core` as a peer dependency of `openevals`. This is not required for LangSmith evals in general - you may define evaluators [using arbitrary custom code](#).

## 2. Create a LangSmith API key

To create an API key, head to the [Settings page](#). Then click **Create API Key**.

## 3. Set up your environment

Because this quickstart uses OpenAI models, you'll need to set the `OPENAI_API_KEY` environment variable as well as the required LangSmith ones:

### Shell

```
export LANGSMITH_TRACING=true
export LANGSMITH_API_KEY="<your-langchain-api-key>"

# This example uses OpenAI, but you can use other LLM providers if desired
export OPENAI_API_KEY="<your-openai-api-key>"
```

## 4. Create a dataset

Next, define example input and reference output pairs that you'll use to evaluate your app:

### Python TypeScript

```
import { Client } from "langsmith";

const client = new Client();

// Programmatically create a dataset in LangSmith
// For other dataset creation methods, see:
//
// https://docs.smith.langchain.com/evaluation/how_to_guides/manage_datasets_program
//
// https://docs.smith.langchain.com/evaluation/how_to_guides/manage_datasets_in_appl
const dataset = await client.createDataset("Sample dataset", {
```

```
description: "A sample dataset in LangSmith.",
});

// Create inputs and reference outputs
const examples = [
  {
    inputs: { question: "Which country is Mount Kilimanjaro located in?" },
    outputs: { answer: "Mount Kilimanjaro is located in Tanzania." },
    dataset_id: dataset.id,
  },
  {
    inputs: { question: "What is Earth's lowest point?" },
    outputs: { answer: "Earth's lowest point is The Dead Sea." },
    dataset_id: dataset.id,
  },
];

// Add examples to the dataset
await client.createExamples(examples);
```

## 5. Define what you're evaluating

Now, define `target function` that contains what you're evaluating. For example, this may be one LLM call that includes the new prompt you are testing, a part of your application or your end to end application.

### Python TypeScript

```
import { wrapOpenAI } from "langsmith/wrappers";
import OpenAI from "openai";

const openai = wrapOpenAI(new OpenAI());

// Define the application logic you want to evaluate inside a target function
// The SDK will automatically send the inputs from the dataset to your target function
async function target(inputs: {
  question: string;
}): Promise<{ answer: string }> {
  const response = await openai.chat.completions.create({
    model: "gpt-4o-mini",
    messages: [
```

```

    { role: "system", content: "Answer the following question accurately" },
    { role: "user", content: inputs.question },
  ],
});
return { answer: response.choices[0].message.content?.trim() || "" };
}

```

## 6. Define evaluator

Import a prebuilt prompt from `openevals` and create an evaluator. `outputs` are the result of your target function. `reference_outputs` / `referenceOutputs` are from the example pairs you defined in [step 4](#) above.

### ! INFO

`CORRECTNESS_PROMPT` is just an f-string with variables for `"inputs"`, `"outputs"`, and `"reference_outputs"`. See [here](#) for more information on customizing OpenEvals prompts.

### Python TypeScript

```

import { createLLMASJudge, CORRECTNESS_PROMPT } from "openevals";

const correctnessEvaluator = async (params: {
  inputs: Record<string, unknown>;
  outputs: Record<string, unknown>;
  referenceOutputs?: Record<string, unknown>;
}) => {
  const evaluator = createLLMASJudge({
    prompt: CORRECTNESS_PROMPT,
    model: "openai:o3-mini",
    feedbackKey: "correctness",
  });
  const evaluatorResult = await evaluator({
    inputs: params.inputs,
    outputs: params.outputs,
    referenceOutputs: params.referenceOutputs,
  });
  return evaluatorResult;
};

```

## 7. Run and view results

Finally, run the experiment!

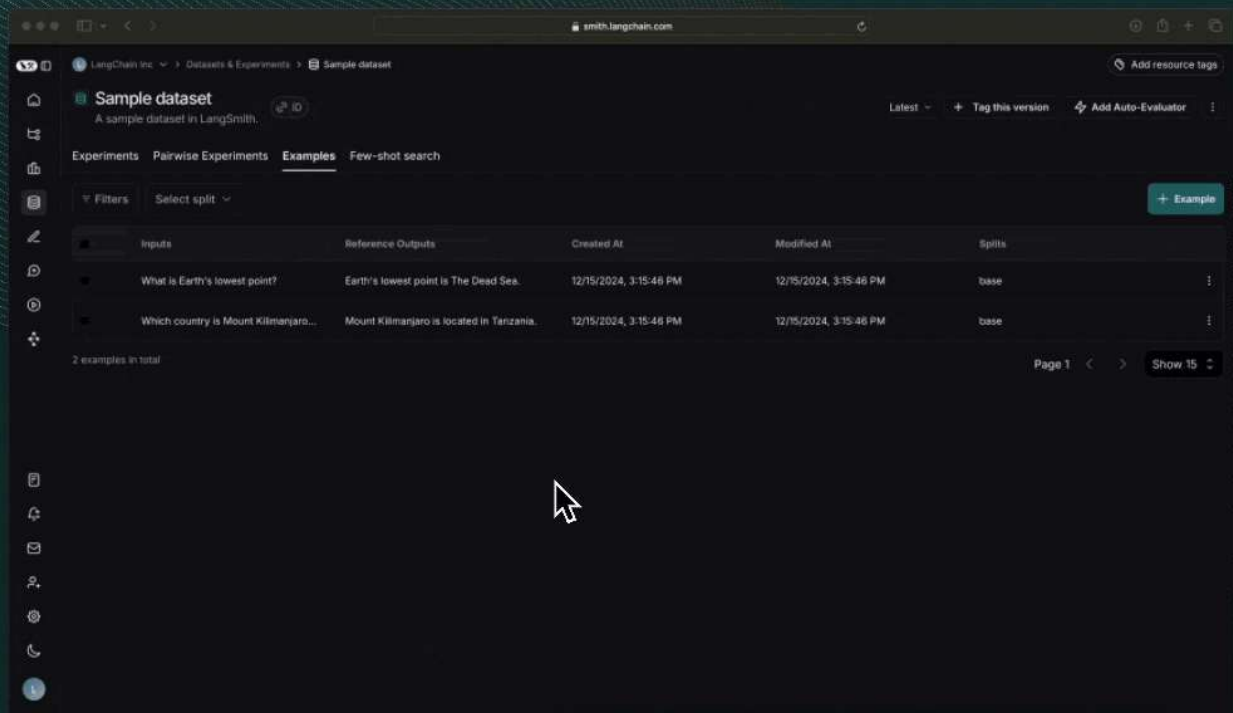
Python

TypeScript

```
import { evaluate } from "langsmith/evaluation";

// After running the evaluation, a link will be provided to view the results
// in langsmith
await evaluate(target, {
  data: "Sample dataset",
  evaluators: [
    correctnessEvaluator,
    // can add multiple evaluators here
  ],
  experimentPrefix: "first-eval-in-langsmith",
  maxConcurrency: 2,
});
```

Click the link printed out by your evaluation run to access the LangSmith Experiments UI, and explore the results of the experiment.



## Next steps



TIP

To learn more about running experiments in LangSmith, read the [evaluation conceptual guide](#).

- Check out the [OpenEvals README](#) to see all available prebuilt evaluators and how to customize them.
- Learn [how to define custom evaluators](#) that contain arbitrary code.
- See the [How-to guides](#) for answers to “How do I...?” format questions.
- For end-to-end walkthroughs see [Tutorials](#).
- For comprehensive descriptions of every class and function see the [API reference](#).

Or, if you prefer video tutorials, check out the [Datasets](#), [Evaluators](#), and [Experiments](#) videos from the Introduction to LangSmith Course.

Was this page helpful?



You can leave detailed feedback [on GitHub](#).