Evaluation

# Introduction

How to use and build evaluation metrics in your testing pipelines

We assume you are familiar with the [data primitives](https://docs.judgmentlabs.ai/documentation/evaluation/example) used in judgeval as well as the [agent modules](https://docs.judgmentlabs.ai/documentation/concepts/agents) you may be evaluating. If you are not familiar with these concepts, we recommend you understand them before continuing.

## [Quickstart](https://docs.judgmentlabs.ai/documentation/evaluation/introduction#quickstart)

evals.py

from judgeval import JudgmentClientfrom judgeval.data import Examplefrom judgeval.scorers import FaithfulnessScorerclient = JudgmentClient()agent = ... # your agenttask = "What if these shoes don't fit?"example = Example( input=task, actual\_output=agent.run(task), # e.g. "We offer a 30-day full refund at no extra cost." retrieval\_context=["All customers are eligible for a 30 day full refund at no extra cost."],)scorer = FaithfulnessScorer(threshold=0.5)results = client.run\_evaluation( examples=[example], scorers=[scorer], model="gpt-4.1",)print(results)

Evals in judgeval consist of three components:

[Example](https://docs.judgmentlabs.ai/documentation/evaluation/example) objects contain the fields involved in the eval. You can group Example objects into [Dataset](https://docs.judgmentlabs.ai/documentation/evaluation/dataset) objects for scaled testing/evals.

[Scorer](https://docs.judgmentlabs.ai/documentation/evaluation/scorers/introduction) objects encode the methodology for scoring the Example objects based on an evaluation criteria.

When using LLM-as-a-judge evals, [Judge](https://docs.judgmentlabs.ai/documentation/evaluation/judges) objects can be used to load specific LLMs to use as the judge model.

# Example

An Example is a basic unit of data in judgeval that allows you to run evaluation scorers on your agents.

In general, an Example corresponds to a single [span in an agent trace](https://docs.judgmentlabs.ai/documentation/concepts/tracing#spans).

In the context of unit testing, an Example corresponds to a single test case.

An Example can be composed of a mixture of the following fields:

| Field | Type | Description |
| --- | --- | --- |
| input | Optional[Union[str, Dict[str, Any]]] | Sample input to your agent/task |
| actual\_output | Optional[Union[str, List[str]]] | What your agent outputs based on the input |
| expected\_output | Optional[Union[str, List[str]]] | The ideal output of your agent |
| retrieval\_context | Optional[List[str]] | Context retrieved from a vector database |
| context | Optional[List[str]] | Ground truth information supplied to the agent |
| tools\_called | Optional[List[str]] | Tools that your agent actually invoked |
| expected\_tools | Optional[List[str]] | Tools you expect your agent to use |
| additional\_metadata | Optional[Dict[str, Any]] | Extra information attached to the example |

Here's a sample of creating an Example:

StandardCustom

example.py

from judgeval.data import Exampleexample = Example( input="Who founded Microsoft?", actual\_output="Bill Gates and Paul Allen.", expected\_output="Bill Gates and Paul Allen founded Microsoft in New Mexico in 1975.", retrieval\_context=["Bill Gates co-founded Microsoft with Paul Allen in 1975."], context=["Bill Gates and Paul Allen are the founders of Microsoft."], tools\_called=["research\_person", "research\_company"], expected\_tools=["research\_person", "research\_company"], additional\_metadata={"research\_source": "Wikipedia"})

It is often most useful to create an Example from the direct outputs of your agent system:

from judgeval.data import Exampleinput\_q = "Who founded Microsoft?"example = Example( input=input\_q, actual\_output=agent.research(input\_q),)

This enables you to use the Example as a test case for your agent system in your CI pipeline.

## [Example Fields](https://docs.judgmentlabs.ai/documentation/evaluation/example#example-fields)

### Input

Optional[Union[str, Dict[str, Any]]]

The input field represents a sample input to your agent/task.

example.py

from judgeval.data import Exampleexample = Example(input="Is sparkling water healthy?")

You can also pass a dictionary if your agent takes multiple args or a JSON object:

example.py

from judgeval.data import Exampleexample = Example( input={ "question": "Is sparkling water healthy?", "tool\_preferences": ["research\_person", "research\_company"], # suggested tools to agent "user\_id": "1234567890" # use for personalization })

### Actual Output

Optional[Union[str, List[str]]]

The actual\_output field represents what your agent outputs based on the input. This is the actual output of your agent system created either at evaluation time or with saved answers.

example.py

from judgeval.data import Exampleexample = Example( input="Is sparkling water healthy?", actual\_output="Sparkling water is neither healthy nor unhealthy.")

### Expected Output

Optional[Union[str, List[str]]]

The expected\_output field represents the ideal output of your agent.

example.py

from judgeval.data import Exampleexample = Example( input="Is sparkling water healthy?", actual\_output="Sparkling water is neither healthy nor unhealthy.", expected\_output="Sparkling water is neither healthy nor unhealthy.")

### Retrieval Context

Optional[List[str]]

The retrieval\_context field represents the context that is actually retrieved from a vector database.

example.py

from judgeval.data import Exampleexample = Example( input="Is sparkling water healthy?", actual\_output="Sparkling water is neither healthy nor unhealthy.", expected\_output="Sparkling water is neither healthy nor unhealthy.", retrieval\_context=["Sparkling water is carbonated and has no calories."])

### Context

Optional[List[str]]

The context field represents information that is supplied to the agent system as ground truth.

For instance, context could be a list of facts that the agent is aware of. However, context should not be confused with retrieval\_context.

In RAG systems, contextual information is retrieved from a vector database and is represented in judgeval by retrieval\_context, not context.

If you're building a RAG system, you'll want to use retrieval\_context.

example.py

# Sample app implementationimport medical\_chatbotfrom judgeval.data import Examplequestion = "Is sparkling water healthy?"example = Example( input=question, actual\_output=medical\_chatbot.chat(question), expected\_output="Sparkling water is neither healthy nor unhealthy.", retrieval\_context=["Sparkling water is carbonated and has no calories."], context=["Sparkling water is a type of water that is carbonated."])

### Tools Called

Optional[List[str]]

The tools\_called field represents the list of tools that your agent actually invoked during the process of generating the output. This is useful for evaluating tool-using agents or agents that interact with external APIs.

example.py

from judgeval.data import Exampleexample = Example( input="What is the weather in Paris?", actual\_output="The weather in Paris is sunny and 25°C.", tools\_called=["weather\_api"])

### Expected Tools

Optional[List[str]]

The expected\_tools field represents the list of tools that you expect your agent to use to answer the input. This can be used to compare with tools\_called to evaluate whether the agent used the correct tools.

example.py

from judgeval.data import Exampleexample = Example( input="What is the weather in Paris?", actual\_output="The weather in Paris is sunny and 25°C.", tools\_called=["weather\_api"], expected\_tools=["weather\_api"])

### Additional Metadata

Optional[Dict[str, Any]]

The additional\_metadata field allows you to attach any extra information to the example. This can be useful for storing custom tags, sources, or any other data relevant to your evaluation or analysis.

example.py

from judgeval.data import Exampleexample = Example( input="What is the weather in Paris?", actual\_output="The weather in Paris is sunny and 25°C.", tools\_called=["weather\_api"], expected\_tools=["weather\_api"], additional\_metadata={"source": "OpenWeatherMap", "confidence": 0.95})

# Dataset

If you haven't read about [examples](https://docs.judgmentlabs.ai/documentation/evaluation/example), you should reference the docs page before continuing.

#### [Learn about Examples](https://docs.judgmentlabs.ai/documentation/evaluation/example)

[The data primitives of evals in judgeval](https://docs.judgmentlabs.ai/documentation/evaluation/example)

In most scenarios, you will have multiple [examples](https://docs.judgmentlabs.ai/documentation/evaluation/example) that you want to evaluate together. The EvalDataset class is used to manage collections of examples. Datasets in judgeval allow you to scale evaluations and save, load, and synchronize datasets with the Judgment platform.

## [Creating a Dataset](https://docs.judgmentlabs.ai/documentation/evaluation/dataset#creating-a-dataset)

Datasets can be created by passing a list of examples to the EvalDataset constructor.

dataset.py

from judgeval.data import Examplefrom judgeval.data.datasets import EvalDatasetexamples = [ Example(input="...", actual\_output="..."), Example(input="...", actual\_output="..."), ...]dataset = EvalDataset( examples=examples)

You can also add Examples to an existing EvalDataset.

dataset.add\_example(Example(input="Question 3?", actual\_output="Answer 3."))

## [Saving/Loading Datasets](https://docs.judgmentlabs.ai/documentation/evaluation/dataset#savingloading-datasets)

You can save and load EvalDataset objects locally and interact with the Judgment Platform.

Remote access: Judgment Platform

Local Formats: JSON, CSV, YAML

The save\_as methods and add\_from\_X methods are available for all three local formats. See the relevant code snippet below for a guide on how to save/load from each format!

### From Judgment Platform

You can push your local EvalDataset to the Judgment platform or pull an existing one.

dataset.py

from judgeval import JudgmentClientfrom judgeval.data.datasets import EvalDatasetclient = JudgmentClient()client.push\_dataset(alias="my\_dataset", dataset=dataset, project\_name="my\_project")pulled\_dataset = client.pull\_dataset(alias="my\_dataset", project\_name="my\_project")

### From JSON

### From CSV

### From YAML

## [Evaluate On Your Dataset / Examples](https://docs.judgmentlabs.ai/documentation/evaluation/dataset#evaluate-on-your-dataset--examples)

You can use the JudgmentClient to evaluate a collection of Examples using scorers.

Want to learn more about running evaluations?

Check out our [evaluation scoring](https://docs.judgmentlabs.ai/documentation/evaluation/introduction) section for a deep dive.

evaluate\_dataset.py

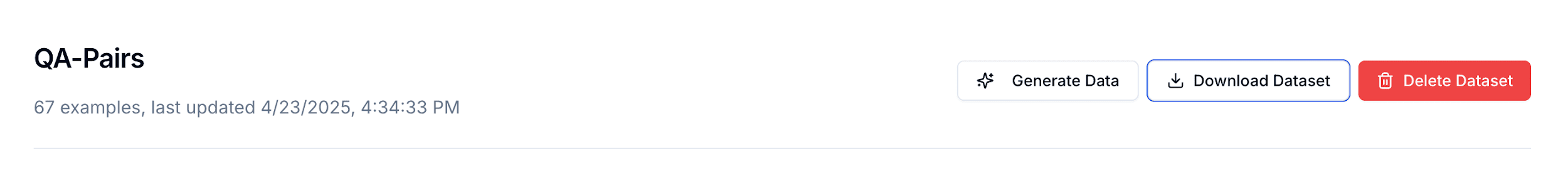
from judgeval import JudgmentClientfrom judgeval.scorers import FaithfulnessScorerclient = JudgmentClient()res = client.run\_evaluation( examples=dataset.examples, scorers=[FaithfulnessScorer(threshold=0.9)], model="gpt-4.1",)

## [Exporting Datasets](https://docs.judgmentlabs.ai/documentation/evaluation/dataset#exporting-datasets)

You can export your datasets from the Judgment Platform UI for backup purposes or sharing with team members.

### [Export from Platform UI](https://docs.judgmentlabs.ai/documentation/evaluation/dataset#export-from-platform-ui)

1. Navigate to your project in the [Judgment Platform](https://app.judgmentlabs.ai/)
2. Select the dataset you want to export
3. Click the "Download Dataset" button in the top right
4. The dataset will be downloaded as a JSON file



The exported JSON file contains the complete dataset information, including metadata and examples:

{ "dataset\_id": "f852eeee-87fa-4430-9571-5784e693326e", "organization\_id": "0fbb0aa8-a7b3-4108-b92a-cc6c6800d825", "dataset\_alias": "QA-Pairs", "comments": null, "source\_file": null, "created\_at": "2025-04-23T22:38:11.709763+00:00", "examples": [ { "example\_id": "119ee1f6-1046-41bc-bb89-d9fc704829dd", "input": "How can I start meditating?", "actual\_output": null, "expected\_output": "Meditation is a wonderful way to relax and focus...", "context": null, "retrieval\_context": null, "additional\_metadata": { "synthetic": true }, "tools\_called": null, "expected\_tools": null, "name": null, "created\_at": "2025-04-23T23:34:33.117479+00:00", "dataset\_id": "f852eeee-87fa-4430-9571-5784e693326e", "eval\_results\_id": null, }, // more examples... ]}

# Unit Testing

Use evals as unit tests in your CI pipelines

judgeval enables you to unit test your agent against predefined tasks/inputs, with built-in support for common unit testing frameworks like [pytest](https://docs.pytest.org/en/stable/).

## [Quickstart](https://docs.judgmentlabs.ai/documentation/evaluation/unit-testing#quickstart)

You can formulate evals as unit tests by checking if [scorers](https://docs.judgmentlabs.ai/documentation/evaluation/scorers/introduction) exceed or fall below threshold values on a set of [examples](https://docs.judgmentlabs.ai/documentation/evaluation/example) (test cases).

The client.assert\_test() runs evaluations as unit tests, raising an exception if the score falls below the defined threshold.

unit\_test.py

from judgeval import JudgmentClientfrom judgeval.data import Examplefrom judgeval.scorers import FaithfulnessScorerclient = JudgmentClient()agent = ... # your agenttask = "What if these shoes don't fit?"example = Example( input=task, actual\_output=agent.run(task), # e.g. "We offer a 30-day full refund at no extra cost." retrieval\_context=["All customers are eligible for a 44 day full refund at no extra cost."],)scorer = FaithfulnessScorer(threshold=0.5)client.assert\_test( examples=[example], scorers=[scorer], model="gpt-4.1", project\_name="my-first-test")

Let's break down how assert\_test() works:

assert\_test() will take each example from examples and pass the input to function as kwargs (in this case, the function is generate\_itinerary) to generate an output. Then, the scorers will be applied to the output and if the score falls below the defined threshold, the test will fail with an exception raised.

If an example fails, the test will report the failure like this:

============================================================================

⚠️ TEST RESULTS: 0/1 passed (1 failed)

============================================================================

✗ Test 1: FAILED

Scorer: Faithfulness

Score: 0.0

Reason: The score is 0.00 because the output repeatedly misrepresented the refund policy, incorrectly stating it as 30 days instead of the 44 days specified in the retrieval context. None of the claims about a 30-day full refund are supported by the actual retrieval context.

----------------------------------------

=============================================================================

Unit tests are treated as evals and the results are saved to your projects on the [Judgment platform](https://app.judgmentlabs.ai/):



## [Pytest Integration](https://docs.judgmentlabs.ai/documentation/evaluation/unit-testing#pytest-integration)

judgeval integrates with pytest so you don't have to write any additional scaffolding for your agent unit tests. We'll reuse the code above and now expect a failure with pytest:

unit\_test.py

import pytest...with pytest.raises(AssertionError): client.assert\_test( examples=[example], scorers=[scorer], model="gpt-4.1", )

## [Test Suites From YAML](https://docs.judgmentlabs.ai/documentation/evaluation/unit-testing#test-suites-from-yaml)

If you have a large number of tests, you may find it convenient to store your test cases in a YAML file:

# tests.yamlexamples: - input: "What if these shoes don't fit?" expected\_output: "We offer a 30-day full refund at no extra cost." retrieval\_context: - "All customers are eligible for a 44 day full refund at no extra cost."

Then run the evaluation using the YAML file:

from judgeval.utils.file\_utils import get\_examples\_from\_yamlclient.assert\_test( examples=get\_examples\_from\_yaml("tests.yaml"), scorers=[FaithfulnessScorer(threshold=0.5)], model="gpt-4.1",)

Evaluation/Scorers

# Introduction

Scorers are used to evaluate agent systems based on specific criteria.

## [Overview](https://docs.judgmentlabs.ai/documentation/evaluation/scorers/introduction#overview)

#### [Want to see a new scorer?](https://github.com/judgmentlabs/judgeval/issues)

[We're always adding new scorers to judgeval. If you have a suggestion, please let us know by opening a GitHub issue!](https://github.com/judgmentlabs/judgeval/issues)

Scorers execute on [Examples](https://docs.judgmentlabs.ai/documentation/evaluation/example) and [Dataset](https://docs.judgmentlabs.ai/documentation/evaluation/dataset), producing a numerical score.

## [Categories of Scorers](https://docs.judgmentlabs.ai/documentation/evaluation/scorers/introduction#categories-of-scorers)

judgeval supports three implementations of scorers.

Default Scorers: plug-and-play scorers carefully crafted by our [research team](https://judgmentlabs.ai/research).

Custom Scorers: Powerful scorers that you can tailor to your own agent systems.

Classifier Scorers: A custom scorer that evaluates based on a prompt you provide.

### Default Scorers

Most built-in scorers in judgeval are implemented using LLM systems, leveraging prompt engineering to ensure evaluations are accurate and scalable. Each scorer provides transparent reasoning through detailed chains of thought, enabling validation of evaluation results.

In line with our [agent concepts](https://docs.judgmentlabs.ai/documentation/concepts/agents), default scorers are divided by the agent module they are designed to evaluate. This includes [planning](https://docs.judgmentlabs.ai/documentation/evaluation/scorers/planning), [tool calling](https://docs.judgmentlabs.ai/documentation/evaluation/scorers/tool_calling), [abilities](https://docs.judgmentlabs.ai/documentation/evaluation/scorers/abilities), and memory.

### Custom Scorers

### Classifier Scorers

## [Running Scorers](https://docs.judgmentlabs.ai/documentation/evaluation/scorers/introduction#running-scorers)

All scorers in judgeval can be run uniformly through the JudgmentClient. All scorers are set to run in async mode by default in order to support parallelized evaluations for large datasets.

run\_scorer.py

from judgeval import JudgmentClientexample = ... # your choicescorer = ...client = JudgmentClient()results = client.run\_evaluation( examples=[example], scorers=[scorer], model="gpt-4.1",)

Evaluation/Scorers/Planning

# Execution Order

The ExecutionOrder scorer is a default scorer that checks whether your LLM agent traversed the correct path in your execution graph. This scorer is an algorithm-based scorer that does not rely on an LLM judge.

In practice, this allows you to assess the quality of an LLM agent's tool choice and applied use of tools as well as the order of node visits.

## [Required Fields](https://docs.judgmentlabs.ai/documentation/evaluation/scorers/planning/execution-order#required-fields)

To run the ExecutionOrder scorer, you must include the following fields in your [Example](https://docs.judgmentlabs.ai/documentation/evaluation/example):

* actual\_output
* expected\_output

## [Scorer Breakdown](https://docs.judgmentlabs.ai/documentation/evaluation/scorers/planning/execution-order#scorer-breakdown)

The execution order score is calculated in different ways depending on how you configure the scorer.

Set Match (Default)

Calculates the score based on the intersection of the actual and expected actions. The score is the size of the intersection divided by the total number of expected actions.

This configuration is useful when all you care about is that the correct tools were called in any order with other possible tools too.

set\_match.py

scorer = ExecutionOrderScorer(threshold=0.8)

Execution Order=Intersection of Actual and Expected ActionsTotal Number of Actions

Execution Order=

Total Number of Actions

Intersection of Actual and Expected Actions

​

Ordering Match

Uses the Longest Common Subsequence (LCS) to calculate the score. The score is the length of the LCS divided by the length of the expected output. If the LCS is the same as the expected output, the score is 1.0, otherwise it is 0.0.

This configuration is useful when you care about the ordering of the tools called but are fine with other tools being within the path.

ordering\_match.py

scorer = ExecutionOrderScorer(threshold=0.8, should\_consider\_ordering=True)

Exact Match

Checks that the actual output matches the expected output exactly. Returns a score of 1.0 if they match, otherwise 0.0.

This configuration is useful when you care about the exact ordering of the tools called.

exact\_match.py

scorer = ExecutionOrderScorer(threshold=0.8, should\_exact\_match=True)

## [Sample Implementation](https://docs.judgmentlabs.ai/documentation/evaluation/scorers/planning/execution-order#sample-implementation)

from judgeval import JudgmentClientfrom judgeval.data import Examplefrom judgeval.scorers import ToolCorrectnessScorerclient = JudgmentClient()example = Example( actual\_output=["GoogleSearch", "Perplexity"], expected\_output=["DBQuery", "GoogleSearch"],)# supply your own thresholdscorer = ExecutionOrderScorer(threshold=0.8)results = client.run\_evaluation( examples=[example], scorers=[scorer],)print(results)

Evaluation/Scorers/Tool Calling

# Tool Order

The Tool Order scorer is an agentic scorer that evaluates whether tools are called in the correct sequence and optionally with the correct parameters.

This is particularly useful for evaluating agent workflows where the order of tool calls matters for the overall success of the task.

## [Scorer Breakdown](https://docs.judgmentlabs.ai/documentation/evaluation/scorers/tools/tool-order#scorer-breakdown)

The Tool Order scorer offers two distinct scoring modes that can be configured to match your evaluation needs.

Ordering Match (Default)

Checks that the ordering of the tools called is the same as the expected ordering. Returns a score of 1.0 if they match, otherwise 0.0.

This score is useful when you care about the ordering of the tools called but are fine with other tools being within the path.

ordering\_match.py

scorer = ToolOrderScorer()

Exact Match

Checks that the ordering of the tools called is exactly the same as the expected ordering. Returns a score of 1.0 if they match, otherwise 0.0.

This score is useful when you care about the exact ordering of the tools called.

exact\_match.py

scorer = ToolOrderScorer(exact\_match=True)

Additionally, if tool parameters are included in the expected ordering, they will be checked against the actual parameters used. If parameters are not specified, only the tool order will be evaluated.

## [Example Agent Tool Structure](https://docs.judgmentlabs.ai/documentation/evaluation/scorers/tools/tool-order#example-agent-tool-structure)

Here's how to structure your agent and tools with the @judgment.observe decorator:

Python (Vanilla)LangGraph (Python)

vanilla\_agent.py

from judgeval.tracer import Tracerjudgment = Tracer(project\_name="my\_agent")class MyAgent: # sample agent, replace with your own @judgment.observe(span\_type="tool") def get\_attractions(self, destination: str) -> str: """Get attractions for a destination""" pass @judgment.observe(span\_type="tool") def get\_weather(self, destination: str, start\_date: str, end\_date: str) -> str: """Get weather forecast for a destination""" pass @judgment.observe(span\_type="function") def run\_agent(self, prompt: str) -> str: """Run the agent with the given prompt""" attractions = self.get\_attractions("Paris") # replace with your own tool calling logic weather = self.get\_weather("Paris", "2025-06-01", "2025-06-02") # replace with your own args return f"Attractions: {attractions}\nWeather: {weather}"

## [Sample Implementation](https://docs.judgmentlabs.ai/documentation/evaluation/scorers/tools/tool-order#sample-implementation)

Then you can use the ToolOrderScorer to evaluate the agent's tool selection/ordering:

Python (Vanilla)LangGraph (Python)

vanilla\_agent.py

from judgeval import JudgmentClientfrom judgeval.data import Examplefrom judgeval.scorers import ToolOrderScorerclient = JudgmentClient()# Define example with expected tool sequenceexample = Example( input={"prompt": "What's the attraction and weather in Paris for early June 2025 (1st - 2nd)?"}, expected\_tools=[ { "tool\_name": "get\_attractions", "parameters": {"destination": "Paris"} }, { "tool\_name": "get\_weather", "parameters": {"destination": "Paris", "start\_date": "2025-06-01", "end\_date": "2025-06-02"} } ])agent = MyAgent()results = client.assert\_test( examples=[example], scorers=[ToolOrderScorer(exact\_match=True)], function=agent.run\_agent, tracer=judgment)

Try playing around with the inputs and expected outputs to simulate different tool calling situations, like when the agent fails.

You can also define your test cases in a YAML file:

# tests.yamlexamples: - input: prompt: "What's the attraction and weather in Paris for early June 2025 (1st - 2nd)?" expected\_tools: - tool\_name: "get\_attractions" parameters: destination: "Paris" - tool\_name: "get\_weather" parameters: destination: "Paris" start\_date: "2025-06-01" end\_date: "2025-06-02"

Then run the evaluation using the YAML file:

Python (Vanilla)LangGraph (Python)

from judgeval.utils.file\_utils import get\_examples\_from\_yamlclient.assert\_test( examples=get\_examples\_from\_yaml("tests.yaml"), scorers=[ToolOrderScorer(exact\_match=True)], function=agent.run\_agent, tracer=judgment)

Monitoring  
  
Performance Monitoring

# Online Evals

Run real-time evaluations on your agents in production.

## [Quickstart](https://docs.judgmentlabs.ai/documentation/performance/online-evals#quickstart)

Online evals are embedded as a method of the Tracer class. They can be attached to any trace and will be executed asynchronously, inducing no latency on your agent's response time.

my\_agent.py

from judgeval.common.tracer import Tracer, wrapfrom judgeval.scorers import AnswerRelevancyScorerfrom judgeval.data import Examplefrom openai import OpenAIclient = wrap(OpenAI())judgment = Tracer(project\_name="my\_project")@judgment.observe(span\_type="tool")def my\_tool(): return "Hello world!"@judgment.observe(span\_type="function")def main(): task\_input = my\_tool() res = client.chat.completions.create( model="gpt-4.1", messages=[{"role": "user", "content": f"{task\_input}"}] ).choices[0].message.content judgment.async\_evaluate( scorers=[AnswerRelevancyScorer(threshold=0.5)], example=Example( input=task\_input, actual\_output=res ), model="gpt-4.1" ) return resmain()

You should see the online eval results attached to the relevant trace span on the Judgment platform shortly after the trace is recorded.

Evals can take time to execute, so they may appear slightly delayed on the UI. Once the eval is complete, you should see it attached to your trace like this:

