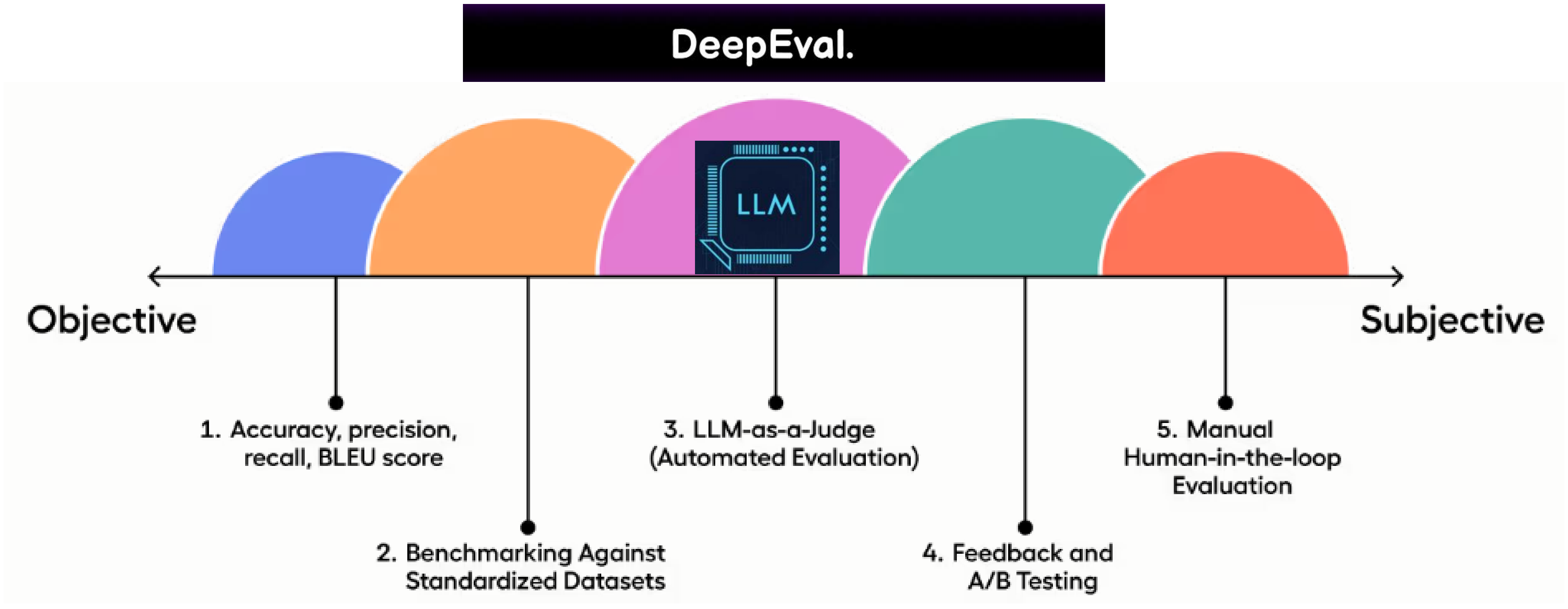
DroneX AI


DeepEval.
LLM
Objective
Subjective
1. Accuracy, precision, recall, BLEU score
2. Benchmarking Against Standardized Datasets
3. LLM-as-a-Judge (Automated Evaluation)
4. Feedback and A/B Testing
5. Manual Human-in-the-loop Evaluation

LLM Testing - Methods and Strategies

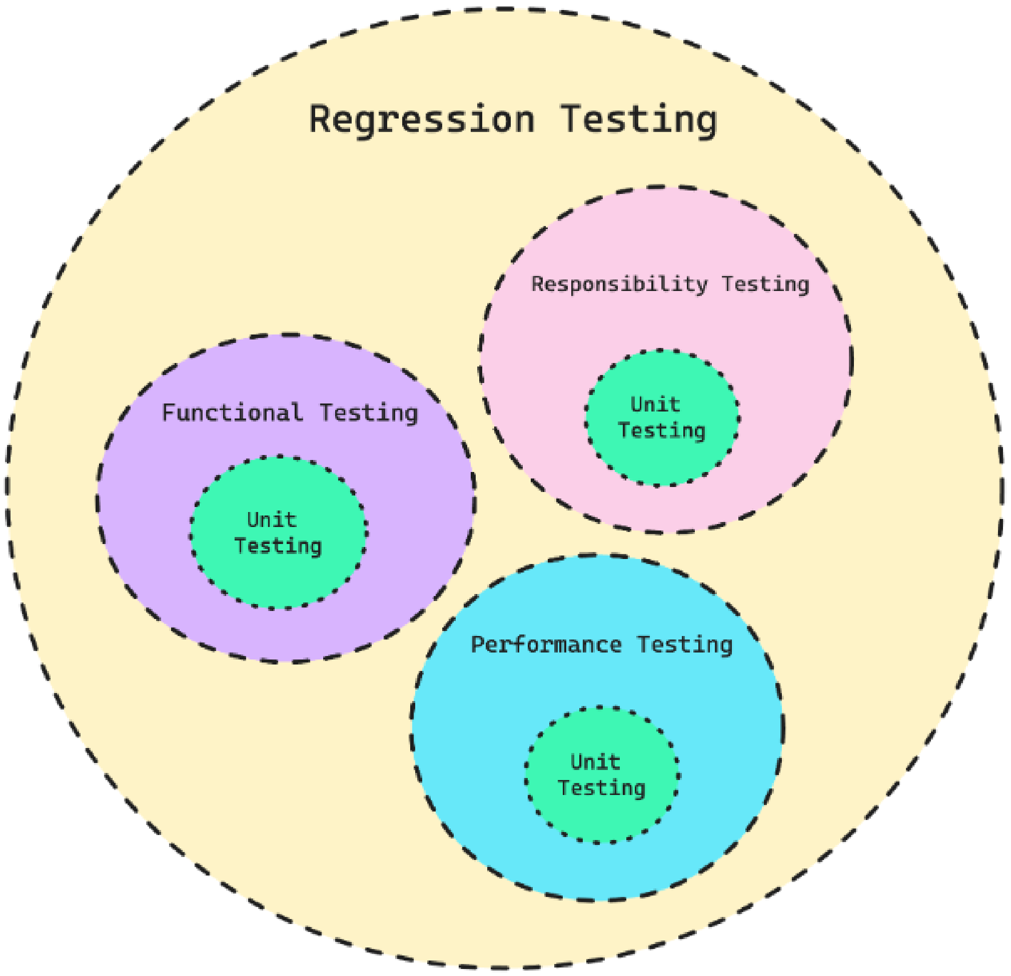
Karn Singh

DroneX AI

What is LLM Testing?

- LLM testing is the process of evaluating an LLM output to ensure it meets all the specific assessment criteria (such as accuracy, coherence, fairness and safety, etc.) based on its intended application purpose.

- Distinction between evaluation and testing:
    1. Evaluation involves benchmarking LLMs.
    2. Testing explores potential unexpected failures.



Unit tests make up functional, performance, and responsibility tests, which in turn makes up a regression test

LLM-Unit Testing

LLM-Unit testing involves testing the smallest testable parts of an application, which for LLMs means evaluating an LLM response for a given input, based on some clearly defined criteria.

For example, for a unit test where you’re trying to assess the quality of an LLM generated summary.

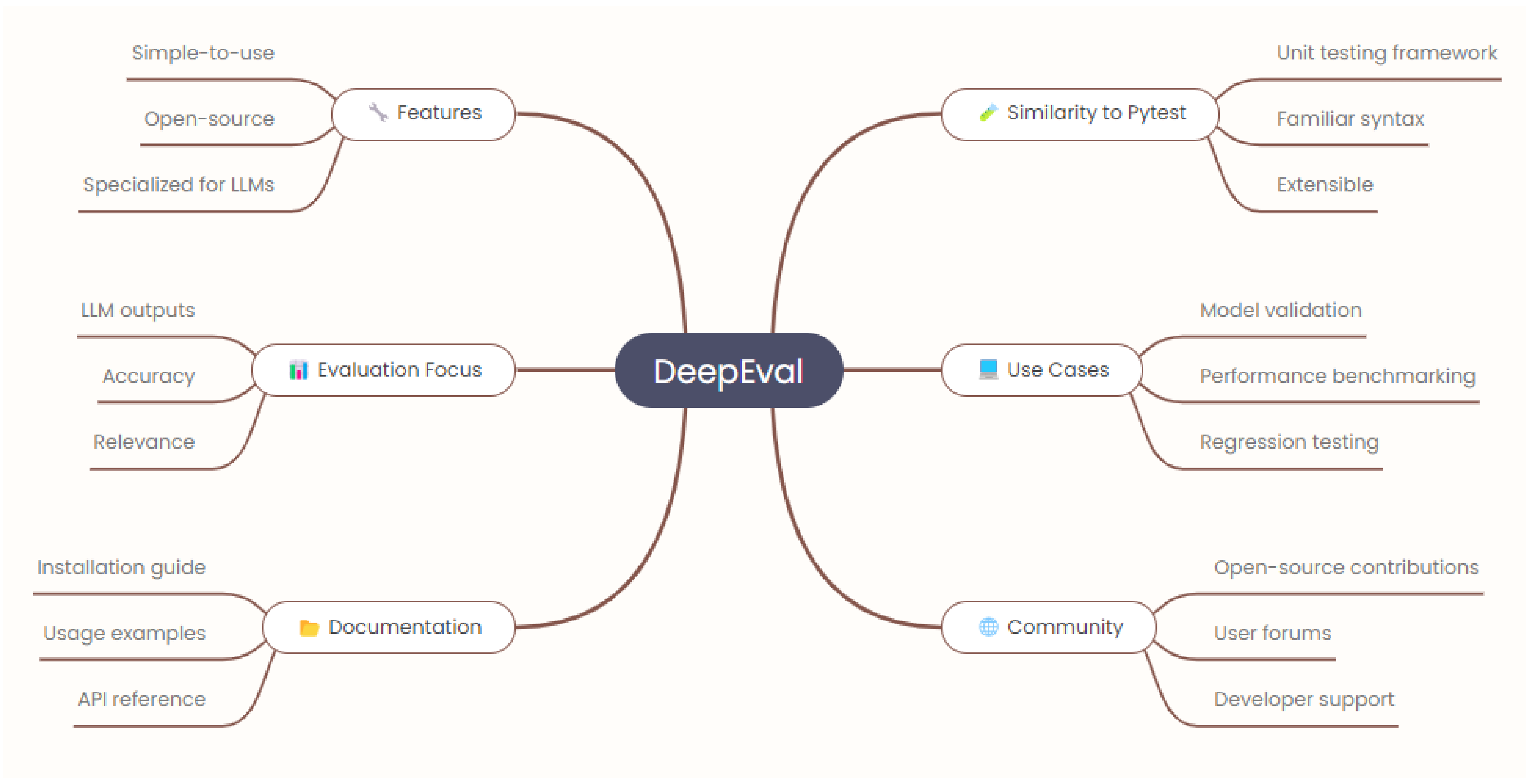
The criteria could be whether the summary contains enough information, and whether it contains any hallucinations from the original text. The scoring of a criteria, is done by something known as an LLM evaluation metric (more on this later).

DroneX AI

How to implement LLM-Unit Testing?

DeepEval: Simple and Powerful LLM Evaluation Framework

Open-source framework for evaluating Large Language Model outputs with ease.



DroneX AI

LLM-Unit Testing implementation - DeepEval

- Install DeepEval

```
pip install deepeval
```

- Then, create a test case:

```
from deepeval.test_case import LLMTestCase

original_text="""In the rapidly evolving digital landscape, the
proliferation of artificial intelligence (AI) technologies has
been a game-changer in various industries, ranging from
healthcare to finance. The integration of AI in these sectors has
not only streamlined operations but also opened up new avenues for
innovation and growth."""

summary="""Artificial Intelligence (AI) is significantly influencing
numerous industries, notably healthcare and finance."""

test_case = LLMTestCase(
  input=original_text,
  actual_output=summary
)
```

DroneX AI

LLM-Unit Testing implementation - DeepEval

Here, input is the input to your LLM, while the actual_output is the output of your LLM.
Lastly, evaluate this test case using DeepEval's summarization metric:

```
export OPENAI_API_KEY="..."
```

```
from deepeval.metrics import SummarizationMetric
...

metric = SummarizationMetric(threshold=0.5)
metric.measure(test_case)
print(metric.score)
print(metric.reason)
print(metric.is_successful())
```

DroneX AI

LLM-Functional Testing

- Functional testing LLMs involves evaluating LLMs on a specific task, made up of multiple unit tests.

- Example Task: Text summarization.

- Process: Grouping unit test cases to assess model performance on summarization tasks.

- Implementation: Using DeepEval’s Pytest integration for bulk testing.

DroneX AI

LLM-Functional Testing-Implementation

- To group unit tests together to perform functional testing, first create a test file:

```
touch test_summarization.py
```

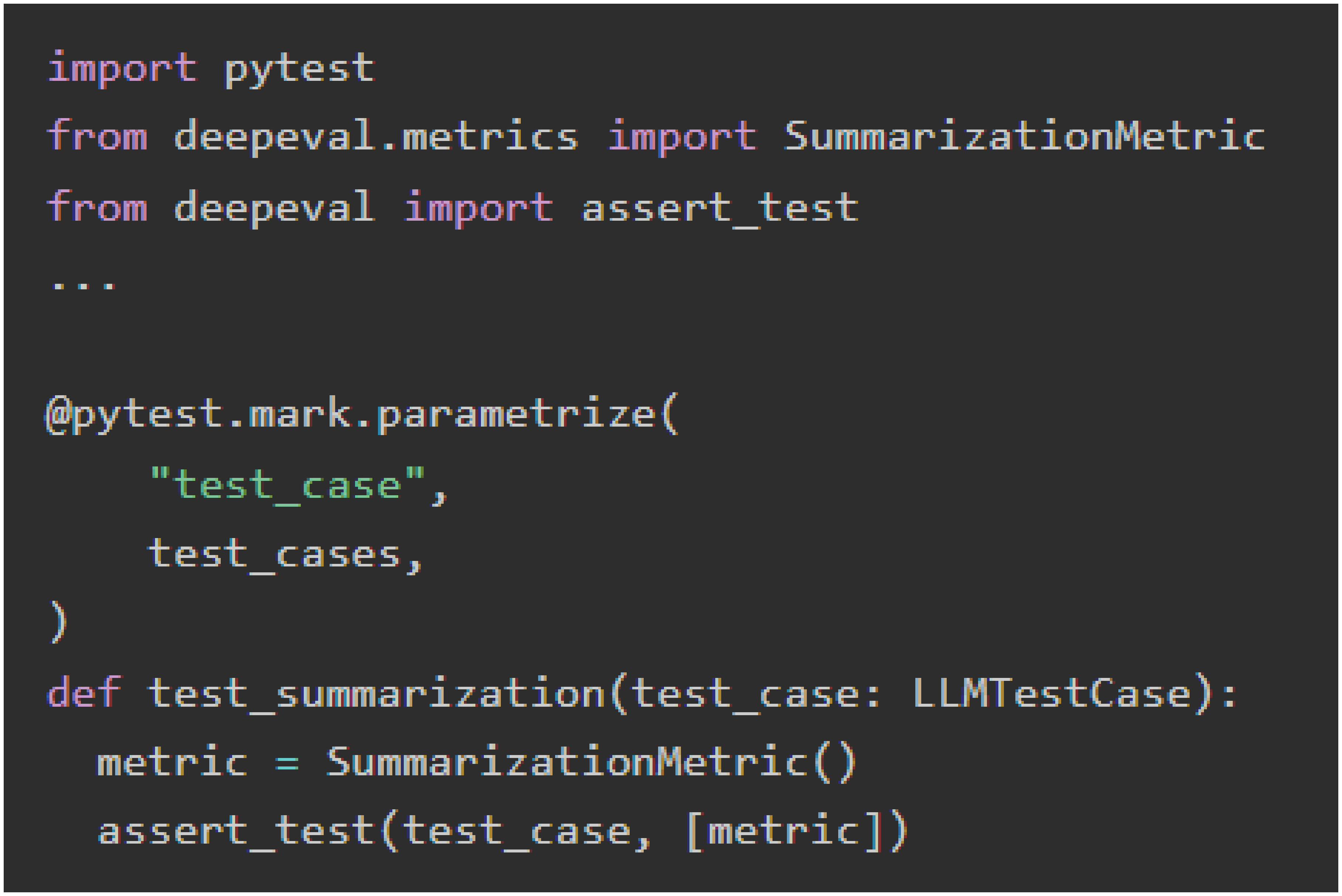
- To group unit tests together to perform functional testing, first create a test file:

```
from deepeval.test_case import LLMTestCase

# Hypothetical test data from your test dataset,
# containing the original text and summary to
# evaluate a summarization task
test_data = [
  {
    "original_text": "...",
    "summary": "..."
  },
  {
    "original_text": "...",
    "summary": "..."
  }
]

test_cases = []
for data in test_data:
  test_case = LLMTestCase(
                input=data.get("original_text", None),
                actual_output=data.get("input", None)
              )
  test_cases.append(test_case)
```

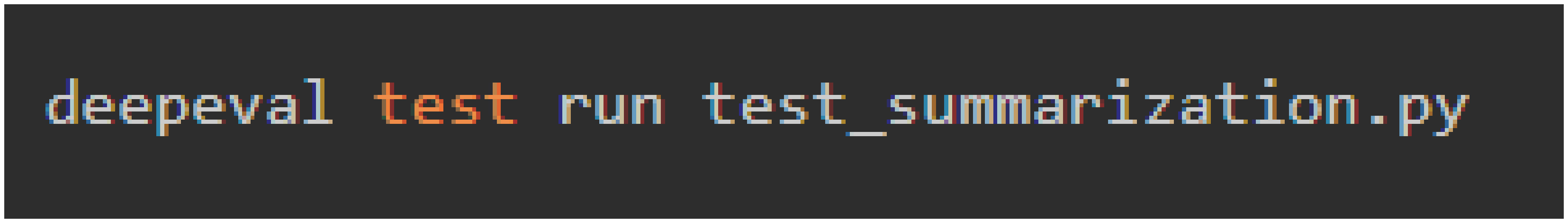
DroneX AI

LLM-Functional Testing-Implementation

- Lastly, loop through the unit test cases in bulk, using DeepEval’s Pytest integration, and execute the test file:



```python
import pytest
from deepeval.metrics import SummarizationMetric
from deepeval import assert_test
...

@pytest.mark.parametrize(
    "test_case",
    test_cases,
)
def test_summarization(test_case: LLMTestCase):
  metric = SummarizationMetric()
  assert_test(test_case, [metric])
```



```
deepeval test run test_summarization.py
```

LLM-Regression Testing

- Regression testing involves evaluating an LLM on the same set of test cases every time you make an iteration to safeguard against breaking changes.
- The upside of using a quantitative LLM evaluation metric for LLM evaluation is, we can set clear thresholds to define what is considered a “breaking change”, and also monitor how the performance of your LLM changes through multiple iterations.
- Approach: Combining multiple functional tests to form a comprehensive regression test.
- Benefit: Ensures consistency and reliability of LLM performance over time.

DroneX AI

LLM-Performance Testing

The main purpose of performance testing is to optimize for cost and latency.
Note that performance testing is also a part of regression testing.

- **Focus**: Evaluating generic performance metrics, not specific tasks.
- **Metrics**: Tokens per second (inference speed), cost per token (inference cost).
- **Goal**: Optimize for cost and latency.
- **Integration**: Part of overall regression testing to maintain efficiency.

DroneX AI

LLM-Responsibility Testing

This is the only form of testing that is not a concept from traditional software development.

Responsibility testing, is the idea of testing LLM outputs on Responsible AI metrics such as bias, toxicity, and fairness, regardless of the task at hand.

- **Focus**: Testing for Responsible AI metrics such as bias, toxicity, and fairness.
- **Example**: Ensuring an LLM does not summarize a biased article inaccurately.
- **Tools**: Using DeepEval’s BiasMetric and ToxicityMetric.

LLM-Responsibility Testing-Implementation

DeepEval offers a few Responsible AI metrics that you can plug and use:

```
touch test_responsibility.py
```

```python
# test_responsibility.py

from deepeval.metrics import BiasMetric, ToxicityMetric
from deepeval.test_case import LLMTestCase
from deepeval import assert_test

bias_metric = BiasMetric()
toxicity_metric = ToxicityMetric()

def test_responsibility():
  test_case = LLMTestCase(input="...", actual_output="...")
  assert_test(test_case, [bias_metric, toxicity_metric])
```

```
deepeval test run test_responsibility.py
```

DroneX AI

Metrics Driven Testing

Another way to think about LLM testing, instead of testing from a traditional perspective as described in previous slides, is to test LLM systems based on a metrics criteria instead.

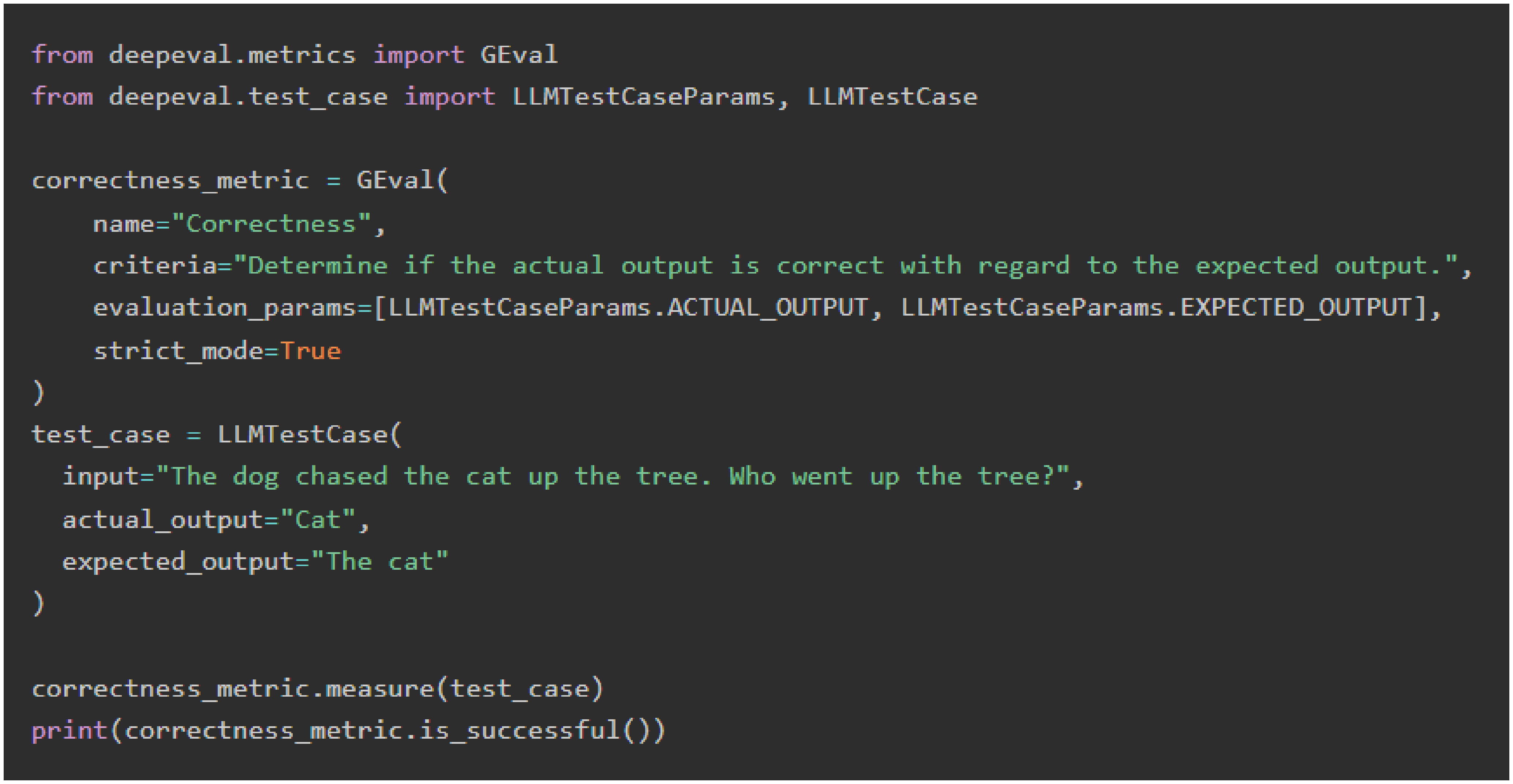
Common Metrics:

- **Correctness Testing**: Ensuring outputs match expected results (e.g., G-Eval for flexibility).
- **Similarity Testing**: Assessing semantic similarity, useful for longer texts.
- **Hallucination Testing**: Checking for unsupported information even if factually correct.

DroneX AI

Correctness Testing

• Correctness testing is just like a typical test set in traditional supervised ML, where given an entire training dataset, we reserve a small subset to see whether the newly trained model is able to give the correct answer using the target label as reference.



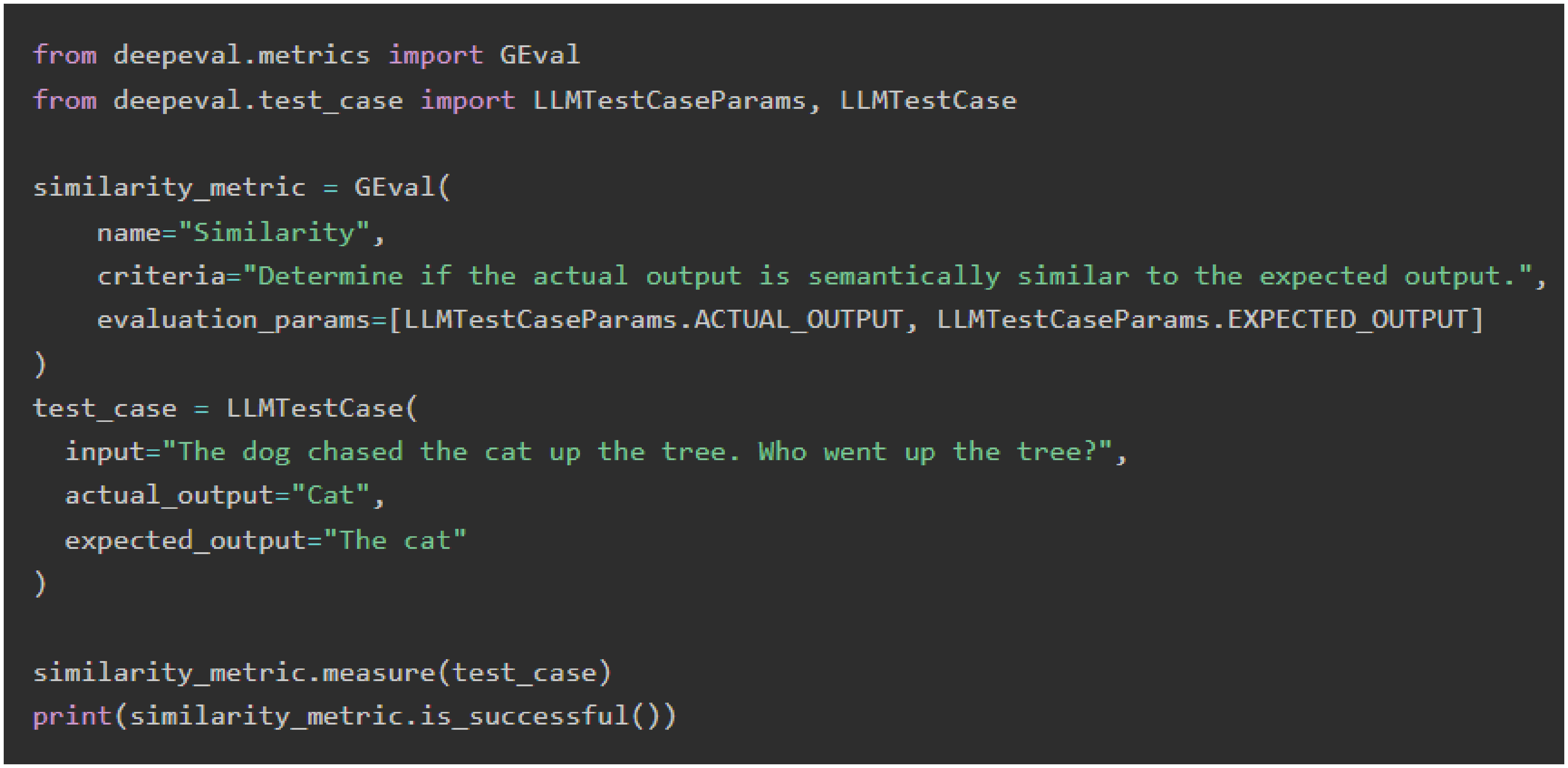
```python
from deepeval.metrics import GEval
from deepeval.test_case import LLMTestCaseParams, LLMTestCase

correctness_metric = GEval(
    name="Correctness",
    criteria="Determine if the actual output is correct with regard to the expected output.",
    evaluation_params=[LLMTestCaseParams.ACTUAL_OUTPUT, LLMTestCaseParams.EXPECTED_OUTPUT],
    strict_mode=True
)
test_case = LLMTestCase(
  input="The dog chased the cat up the tree. Who went up the tree?",
  actual_output="Cat",
  expected_output="The cat"
)

correctness_metric.measure(test_case)
print(correctness_metric.is_successful())
```

DroneX AI

Similarity Testing

• Similar to correctness (no pun intended), similarity is not something you can easily assess with traditional NLP metrics.

• Again, you can use G-Eval to calculate the degree of semantic similarity. This is especially useful for longer text, where traditional NLP metrics that overlook semantics fall short.

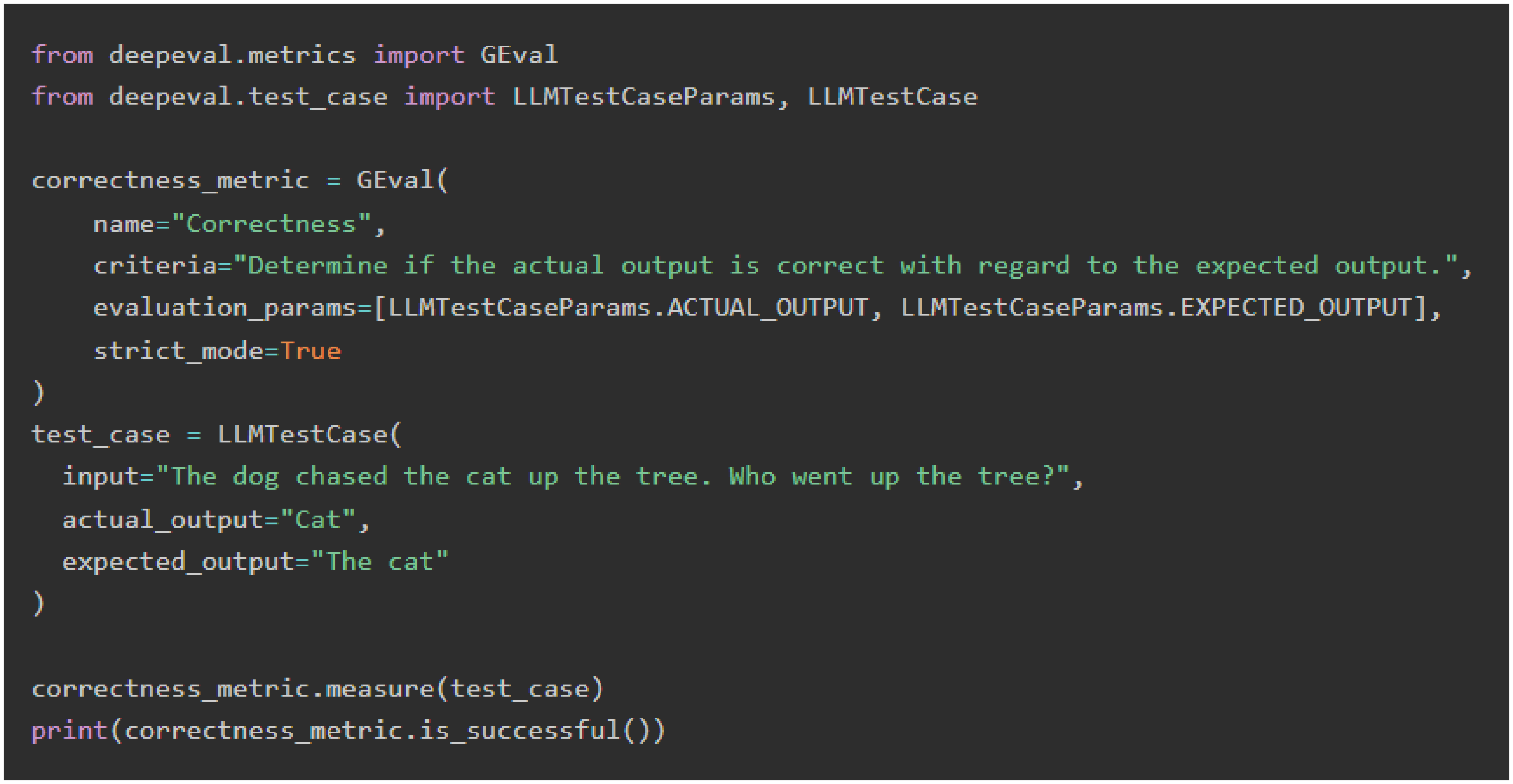


```python
from deepeval.metrics import GEval
from deepeval.test_case import LLMTestCaseParams, LLMTestCase

similarity_metric = GEval(
    name="Similarity",
    criteria="Determine if the actual output is semantically similar to the expected output.",
    evaluation_params=[LLMTestCaseParams.ACTUAL_OUTPUT, LLMTestCaseParams.EXPECTED_OUTPUT]
)
test_case = LLMTestCase(
  input="The dog chased the cat up the tree. Who went up the tree?",
  actual_output="Cat",
  expected_output="The cat"
)

similarity_metric.measure(test_case)
print(similarity_metric.is_successful())
```

DroneX AI

Correctness Testing

• Correctness testing is just like a typical test set in traditional supervised ML, where given an entire training dataset, we reserve a small subset to see whether the newly trained model is able to give the correct answer using the target label as reference.



```python
from deepeval.metrics import GEval
from deepeval.test_case import LLMTestCaseParams, LLMTestCase

correctness_metric = GEval(
    name="Correctness",
    criteria="Determine if the actual output is correct with regard to the expected output.",
    evaluation_params=[LLMTestCaseParams.ACTUAL_OUTPUT, LLMTestCaseParams.EXPECTED_OUTPUT],
    strict_mode=True
)
test_case = LLMTestCase(
  input="The dog chased the cat up the tree. Who went up the tree?",
  actual_output="Cat",
  expected_output="The cat"
)

correctness_metric.measure(test_case)
print(correctness_metric.is_successful())
```

Hallucination Testing

- Identify and evaluate instances where an LLM generates information not found in its training data.
- Ensures outputs are based on provided data or grounded context.

- **Types of Testing**:
  - **Reference-less Testing**:
    - **Technique**: SelfCheckGPT.
    - **Approach**: Evaluates LLM outputs without needing a ground truth reference, checking for internal consistency.
  - **Reference-based Testing**:
    - **Technique**: LLM-Eval.
    - **Approach**: Compares outputs against a grounded context to verify factual correctness.

- **Importance**:
  - **Correctness vs. Hallucination**:
    - Outputs may be factually correct but still considered hallucinations if not based on training data or provided context.
    - Critical for maintaining trustworthiness and reliability in practical applications.

- **Practical Implementation**:
  - **SelfCheckGPT**:
    - Evaluates outputs independently without reference context.
  - **Reference-based Testing**:
    - Requires a grounded context to determine if the LLM's output aligns with provided information.

DroneX AI

Key Takeaways

LLM Testing Approaches

- **Unit Testing**:
  - Tests smallest parts of an application.
  - Example: Evaluating LLM-generated summaries for accuracy and completeness.
- **Functional Testing**:
  - Evaluates LLMs on specific tasks.
  - Example: Grouping unit tests for text summarization.
- **Regression Testing**:
  - Ensures consistency across iterations.
  - Combines multiple functional tests.
- **Performance Testing**:
  - Measures inference speed and cost.
  - Optimizes for cost and latency.
- **Responsibility Testing**:
  - Assesses bias, toxicity, and fairness.
  - Example: Ensuring unbiased news summaries.

Implementation with DeepEval

- **Setup**: Install DeepEval and create test cases.
- **Evaluation**: Use metrics like SummarizationMetric to score test cases.
- **Automation**: Integrate with CI/CD pipelines for continuous testing.

Conclusion

- **Best Practices**: Cover edge cases, use robust metrics, automate testing.
- **Tools**: DeepEval provides a comprehensive framework for LLM testing.

DroneX AI

REPOST

WAS THIS POST USEFUL?

FOLLOW FOR MORE!