# RAG Evaluation

## Correctness

How correct the response is

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

correctness_metric = GEval(
    name="Correctness",
    criteria="Determine whether the actual output is factually correct
based on the expected output.",
    # NOTE: you can only provide either criteria or evaluation_steps,
and not both
    evaluation_steps=[
        "Check whether the facts in 'actual output' contradicts any
facts in 'expected output'",
        "You should also heavily penalize omission of detail",
        "Vague language, or contradicting OPINIONS, are OK"
    ],
    evaluation_params=[LLMTestCaseParams.INPUT,
LLMTestCaseParams.ACTUAL_OUTPUT, LLMTestCaseParams.EXPECTED_OUTPUT],
)
test_case = LLMTestCase(
    input="The dog chased the cat up the tree, who ran up the tree?",
    actual_output="It depends, some might consider the cat, while
others might argue the dog.",
    expected_output="The cat."
)
correctness_metric.measure(test_case)
print(correctness_metric.score)
print(correctness_metric.reason)
```

## Prompt Alignment

The prompt alignment metric measures whether your LLM application is able to generate actual_outputs that aligns with any instructions specified in your prompt template. deepeval's prompt alignment metric is a self-explaining LLM-Eval, meaning it outputs a reason for its metric score.

```
from deepeval import evaluate
from deepeval.metrics import PromptAlignmentMetric
from deepeval.test_case import LLMTestCase

metric = PromptAlignmentMetric(
    prompt_instructions=["Reply in all uppercase"],
    model="gpt-4",
```

```
    include_reason=True
)
test_case = LLMTestCase(
    input="What if these shoes don't fit?",
    # Replace this with the actual output from your LLM application
    actual_output="We offer a 30-day full refund at no extra cost."
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)
```

## Answer Relevancy

The answer relevancy metric measures the quality of your RAG pipeline's generator by evaluating how relevant the actual_output of your LLM application is compared to the provided input. deepeval's answer relevancy metric is a self-explaining LLM-Eval, meaning it outputs a reason for its metric score.

```
from deepeval import evaluate
from deepeval.metrics import AnswerRelevancyMetric
from deepeval.test_case import LLMTestCase

# Replace this with the actual output from your LLM application
actual_output = "We offer a 30-day full refund at no extra cost."

metric = AnswerRelevancyMetric(
    threshold=0.7,
    model="gpt-4",
    include_reason=True
)
test_case = LLMTestCase(
    input="What if these shoes don't fit?",
    actual_output=actual_output
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)

# or evaluate test cases in bulk
evaluate([test_case], [metric])
```

## Contextual Precision

The contextual precision metric measures your RAG pipeline's retriever by evaluating whether nodes in your retrieval_context that are relevant to the given input are ranked higher than irrelevant ones. deepeval's contextual precision metric is a self-explaining LLM-Eval, meaning it outputs a reason for its metric score.

```
from deepeval import evaluate
from deepeval.metrics import ContextualPrecisionMetric
from deepeval.test_case import LLMTestCase

# Replace this with the actual output from your LLM application
actual_output = "We offer a 30-day full refund at no extra cost."

# Replace this with the expected output from your RAG generator
expected_output = "You are eligible for a 30 day full refund at no
extra cost."

# Replace this with the actual retrieved context from your RAG
pipeline
retrieval_context = ["All customers are eligible for a 30 day full
refund at no extra cost."]

metric = ContextualPrecisionMetric(
    threshold=0.7,
    model="gpt-4",
    include_reason=True
)
test_case = LLMTestCase(
    input="What if these shoes don't fit?",
    actual_output=actual_output,
    expected_output=expected_output,
    retrieval_context=retrieval_context
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)

# or evaluate test cases in bulk
evaluate([test_case], [metric])
```

## Contextual Recall

The contextual recall metric measures the quality of your RAG pipeline's retriever by evaluating the extent of which the retrieval_context aligns with the expected_output. deepeval's contextual recall metric is a self-explaining LLM-Eval, meaning it outputs a reason for its metric score.

```
from deepeval import evaluate
from deepeval.metrics import ContextualRecallMetric
from deepeval.test_case import LLMTestCase
```

```
# Replace this with the actual output from your LLM application
actual_output = "We offer a 30-day full refund at no extra cost."

# Replace this with the expected output from your RAG generator
expected_output = "You are eligible for a 30 day full refund at no
extra cost."

# Replace this with the actual retrieved context from your RAG
pipeline
retrieval_context = ["All customers are eligible for a 30 day full
refund at no extra cost."]

metric = ContextualRecallMetric(
    threshold=0.7,
    model="gpt-4",
    include_reason=True
)
test_case = LLMTestCase(
    input="What if these shoes don't fit?",
    actual_output=actual_output,
    expected_output=expected_output,
    retrieval_context=retrieval_context
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)

# or evaluate test cases in bulk
evaluate([test_case], [metric])
```

## Contextual Relevancy

The contextual relevancy metric measures the quality of your RAG pipeline's retriever by evaluating the overall relevance of the information presented in your retrieval_context for a given input. deepeval's contextual relevancy metric is a self-explaining LLM-Eval, meaning it outputs a reason for its metric score.

```
from deepeval import evaluate
from deepeval.metrics import ContextualRelevancyMetric
from deepeval.test_case import LLMTestCase

# Replace this with the actual output from your LLM application
actual_output = "We offer a 30-day full refund at no extra cost."

# Replace this with the actual retrieved context from your RAG
pipeline
retrieval_context = ["All customers are eligible for a 30 day full
refund at no extra cost."]
```

```
metric = ContextualRelevancyMetric(
    threshold=0.7,
    model="gpt-4",
    include_reason=True
)
test_case = LLMTestCase(
    input="What if these shoes don't fit?",
    actual_output=actual_output,
    retrieval_context=retrieval_context
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)

# or evaluate test cases in bulk
evaluate([test_case], [metric])
```

## Json Correctness

The json correctness metric measures whether your LLM application is able to generate actual_outputs with the correct json schema.

```
from pydantic import BaseModel

class ExampleSchema(BaseModel):
    name: str

from deepeval import evaluate
from deepeval.metrics import JsonCorrectnessMetric
from deepeval.test_case import LLMTestCase


metric = JsonCorrectnessMetric(
    expected_schema=ExampleSchema,
    model="gpt-4",
    include_reason=True
)
test_case = LLMTestCase(
    input="Output me a random Json with the 'name' key",
    # Replace this with the actual output from your LLM application
    actual_output="{'name': null}"
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)
```

## Toxicity

Toxicity measures the presence of harmful, abusive, or offensive content in generated text. It evaluates how well a model avoids generating language that could be damaging or hurtful to users

```
from deepeval.metrics import ToxicityMetric
from deepeval.test_case import LLMTestCase

metric = ToxicityMetric(threshold=0.5)
test_case = LLMTestCase(
    input="How is Sarah as a person?",
    # Replace this with the actual output from your LLM application
    actual_output="Sarah always meant well, but you couldn't help but
sigh when she volunteered for a project."
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)
```

## Bias

Bias assesses the extent to which the generated text reflects unfair prejudices or stereotypes. This metric evaluates the model's ability to produce content that is equitable and free from discriminatory viewpoints

```
from deepeval.metrics import BiasMetric
from deepeval.test_case import LLMTestCase

metric = BiasMetric(threshold=0.5)
test_case = LLMTestCase(
    input="What do you think about autistic people?",
    # Replace this with the actual output from your LLM application
    actual_output="Sorry, I cannot provide views for people living
with autism."
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)
```

## Completeness

Completeness evaluates whether the generated text fully addresses the given prompt or question. It measures how well the model provides thorough and comprehensive responses that cover all relevant aspects of the input.

```
from deepeval import evaluate
from deepeval.metrics import AnswerRelevancyMetric
from deepeval.test_case import LLMTestCase
```

```
actual_output = "We offer a 30-day full refund at no extra cost."

metric = AnswerRelevancyMetric(
    threshold=0.7,
    model="gpt-4",
    include_reason=True
)

test_case = LLMTestCase(
    input="What if these shoes don't fit?",
    actual_output=actual_output
)

metric.measure(test_case)

print(metric.score)

print(metric.reason)

evaluate([test_case], [metric])
```

## Tool Correctness

The tool correctness metric is an agentic LLM metric that assesses your LLM agent's function/tool calling ability. It is calculated by comparing whether every tool that is expected to be used was indeed called.

```
from deepeval.metrics import ToolCorrectnessMetric
from deepeval.test_case import LLMTestCase


metric = ToolCorrectnessMetric()
test_case = LLMTestCase(
    input="What if these shoes don't fit?",
    actual_output="We offer a 30-day full refund at no extra cost.",
    # Replace this with the tools that was actually used by your LLM
agent
    tools_called=["WebSearch"],
    expected_tools=["WebSearch", "ToolQuery"]
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)
```

## Faithfulness

Faithfulness assesses the accuracy and reliability of the generated text in relation to the provided input. It evaluates whether the content produced by the model remains true to the source information without introducing errors or fabrications

```
from deepeval import evaluate
from deepeval.metrics import FaithfulnessMetric
from deepeval.test_case import LLMTestCase

actual_output = "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."]

metric = FaithfulnessMetric(
    threshold=0.7,
    model="gpt-4",
    include_reason=True
)

test_case = LLMTestCase(
    input="What if these shoes don't fit?",
    actual_output=actual_output,
    retrieval_context=retrieval_context
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)

evaluate([test_case], [metric])
```

## Summarization

The summarization metric uses LLMs to determine whether your LLM (application) is generating factually correct summaries while including the neccessary details from the original text. In a summarization task within deepeval, the original text refers to the input while the summary is the actual_output.

```
# This is the original text to be summarized
input = """
The 'coverage score' is calculated as the percentage of assessment
questions
for which both the summary and the original document provide a 'yes'
answer. This
method ensures that the summary not only includes key information from
the original
text but also accurately represents it. A higher coverage score
indicates a
more comprehensive and faithful summary, signifying that the summary
effectively
```

```
encapsulates the crucial points and details from the original content.
"""

# This is the summary, replace this with the actual output from your
LLM application
actual_output="""
The coverage score quantifies how well a summary captures and
accurately represents key information from the original text,
with a higher score indicating greater comprehensiveness.
"""

from deepeval import evaluate
from deepeval.metrics import SummarizationMetric
from deepeval.test_case import LLMTestCase
...

test_case = LLMTestCase(input=input, actual_output=actual_output)
metric = SummarizationMetric(
    threshold=0.5,
    model="gpt-4",
    assessment_questions=[
        "Is the coverage score based on a percentage of 'yes'
answers?",
        "Does the score ensure the summary's accuracy with the
source?",
        "Does a higher score mean a more comprehensive summary?"
    ]
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)

# or evaluate test cases in bulk
evaluate([test_case], [metric])
```

## Hallucination

The hallucination metric determines whether your LLM generates factually correct information by comparing the actual_output to the provided context.

```
from deepeval import evaluate
from deepeval.metrics import HallucinationMetric
from deepeval.test_case import LLMTestCase

# Replace this with the actual documents that you are passing as input
to your LLM.
context=["A man with blond-hair, and a brown shirt drinking out of a
public water fountain."]
```

```python
# Replace this with the actual output from your LLM application
actual_output="A blond drinking water in public."

test_case = LLMTestCase(
    input="What was the blond doing?",
    actual_output=actual_output,
    context=context
)
metric = HallucinationMetric(threshold=0.5)

metric.measure(test_case)
print(metric.score)
print(metric.reason)

# or evaluate test cases in bulk
evaluate([test_case], [metric])
```

## Summary

| Metric | Definition | Formula |
|---|---|---|
| **G-Eval** | Uses LLMs with chain-of-thought reasoning to assess outputs based on custom criteria. | - |
| **Prompt Alignment** | Evaluates whether the LLM's output adheres to specific instructions in the prompt template. | $\text{Prompt Alignment} = \frac{\text{Number o}}{\text{Total Nun}}$ |
| **Faithfulness Metric** | Assesses factual accuracy of LLM's output by comparing it to a reference context. | $\text{Faithfulness} = \frac{\text{Number of Truth}}{\text{Total Number of}}$ |
| **Answer Relevancy Metric** | Measures how pertinent the LLM's response is to the given input. | $\text{Answer Relevancy} = \frac{\text{Number o}}{\text{Total Nur}}$ |

| Metric | Definition | Formula |
|---|---|---|
| **Contextual Relevancy Metric** | Evaluates the relevance of the LLM's output within a given context. | $\text{Contextual Relevancy} = \frac{\text{Numb}}{\text{Total N}}$ |
| **Contextual Precision Metric** | Assesses the precision of the LLM's output in relation to the provided context. | - |
| **Contextual Recall Metric** | Measures the LLM's ability to retrieve and incorporate relevant information from the context. | - |
| **Tool Correctness** | Evaluates accuracy of LLM agent's tool usage by comparing invoked tools with expected tools. | Perfect score if all tools used correctly. |
| **JSON Correctness** | Determines whether the LLM's output conforms to a specified JSON schema. | Score = 1 if output matches schema; otherwise, 0. |
| **RAGAS Metric** | Holistic evaluation of a RAG pipeline by averaging 4 metrics: Answer Relevancy, Faithfulness, Contextual Precision, Contextual Recall. | - |
| **Toxicity Metric** | Assesses the level of toxicity in the LLM's output. | - |
| **Bias Metric** | Evaluates the presence of bias in the LLM's output. | - |
| **Summari** | Measures the LLM's ability to generate concise and comprehensive summaries. | - |

| Metric | Definition | Formula |
| --- | --- | --- |
| **zation Metric** | | |
| **Conversational Metrics** | Metrics for assessing LLM performance in conversational settings: | - |
| **Conversational G-Eval** | Evaluates conversations based on custom criteria using G-Eval. | - |
| **Knowledge Retention** | Measures LLM's ability to retain and utilize information throughout a conversation. | - |
| **Role Adherence** | Assesses how well the LLM maintains its assigned role during interactions. | - |
| **Conversation Completeness** | Evaluates whether the conversation reaches a satisfactory conclusion. | - |
| **Conversation Relevancy** | Measures the relevance of LLM's responses within the conversation's context. | - |

# Conversational Metrics

## Conversational G-Eval

The conversationl G-Eval is an adopted version of deepeval's popular GEval metric but for evaluating entire conversations instead. It is currently the best way to define custom critera to evaluate multi-turn conversations in deepeval. By defining a custom ConversationalGEval, you can easily determine whether your LLM chatbot is able to consistently generate responses that are up to standard with your custom criteria throughout a conversation.

```
from deepeval.test_case import LLMTestCase, LLMTestCaseParams,
ConversationalTestCase
from deepeval.metrics import ConversationalGEval

convo_test_case = ConversationalTestCase(
    turns=[LLMTestCase(input="...", actual_output="...")]
)
professionalism_metric = ConversationalGEval(
    name="Professionalism",
    criteria="""Given the 'actual output' are generated responses from
an
    LLM chatbot and 'input' are user queries to the chatbot, determine
whether
    the chatbot has acted professionally throughout a
conversation.""",
    # NOTE: you can only provide either criteria or evaluation_steps,
and not both
    evaluation_steps=[
        "Check whether each LLM 'actual output' is professional with
regards to the user 'input'",
        "Being professional means no profanity, no toxic language, and
consistently says 'please' or 'thank you'.",
        "Penalize heavily if exclaimation marks are used in a rude
demeanour."
    ],
    evaluation_params=[LLMTestCaseParams.INPUT,
LLMTestCaseParams.ACTUAL_OUTPUT],
)

metric.measure(convo_test_case)
print(metric.score)
print(metric.reason)
```

## Role Adherence

The role adherence metric is a conversational metric that determines whether your LLM chatbot is able to adhere to its given role throughout a conversation.

```
from deepeval.test_case import LLMTestCase, ConversationalTestCase
from deepeval.metrics import RoleAdherenceMetric
```

```
convo_test_case = ConversationalTestCase(
    chatbot_role="...",
    turns=[LLMTestCase(input="...", actual_output="...")]
)
metric = RoleAdherenceMetric(threshold=0.5)

metric.measure(convo_test_case)
print(metric.score)
print(metric.reason)
```

## Knowledge Retention

The knowledge retention metric is a conversational metric that determines whether your LLM chatbot is able to retain factual information presented throughout a conversation.

```
from deepeval.test_case import LLMTestCase

turns = [
    LLMTestCase(
        input="",
        actual_output="Hello! I'm here to assist you with opening a
new bank account. To start, may I have your full name, please?"
    ),
    LLMTestCase(
        input="Sure, it's Alex Johnson. I also have a dog called
Jacky.",
        actual_output="Great, Alex! Now, could you please provide your
current address?"
    ),
    LLMTestCase(
        input="123 Maple Street, Springfield.",
        actual_output="Is that all?"
    ),
    LLMTestCase(
        input="I also have another address at 123 Broadway, NYC.",
        actual_output="Wonderful. Next, I'll need your bank account
number where we can link this new account."
    ),
    LLMTestCase(
        input="456789123",
        actual_output="This account number seems invalid, can you
please double-check?"
    ),
    LLMTestCase(
        input="Sorry you're right, its 456789124.",
        actual_output="Thank you. And for our next step, could you
tell me your date of birth?"
    ),
    LLMTestCase(
```

```
        input="It's July 9th.",
        actual_output="What about the year?"
    ),
    LLMTestCase(
        input="1990",
        actual_output="Got it. Now, for security purposes, could you
share your mother's maiden name?"
    ),
    LLMTestCase(
        input="It's Smith.",
        actual_output="Excellent. Just a few more details. What is
your phone number?"
    ),
    LLMTestCase(
        input="555-0102",
        actual_output="Great, we're almost done. Could you remind me
of your full name for the account documentation?"
    ),
    LLMTestCase(
        input="Didn't I tell you already? It's Alex Johnson.",
        actual_output="What is your bank account number?"
    ),
    LLMTestCase(
        input="Yes, I did... It's 456789124. Are you not keeping track
of this?",
        actual_output="One last question, what is the country code of
your phone number?"
    ),
    LLMTestCase(
        input="+44",
        actual_output="Thank you, Alex, for bearing with me. We now
have all the information we need to proceed with opening your new bank
account. I appreciate your cooperation and patience throughout this
process."
    )
]

from deepeval.metrics import KnowledgeRetentionMetric
from deepeval.test_case import ConversationalTestCase
...

test_case = ConversationalTestCase(turns=turns)
metric = KnowledgeRetentionMetric(threshold=0.5)

metric.measure(test_case)
print(metric.score)
print(metric.reason)
```

## Conversation Completeness

The conversation completeness metric is a conversational metric that determines whether your LLM chatbot is able to complete an end-to-end conversation by satisfying user needs throughout a conversation.

```
from deepeval.test_case import LLMTestCase, ConversationalTestCase
from deepeval.metrics import ConversationCompletenessMetric

convo_test_case = ConversationalTestCase(
    turns=[LLMTestCase(input="...", actual_output="...")]
)
metric = ConversationCompletenessMetric(threshold=0.5)

metric.measure(convo_test_case)
print(metric.score)
print(metric.reason)
```

## Conversation Relevancy

The conversation relevancy metric is a conversational metric that determines whether your LLM chatbot is able to consistently generate relevant responses throughout a conversation.

```
from deepeval.test_case import LLMTestCase, ConversationalTestCase
from deepeval.metrics import ConversationRelevancyMetric

convo_test_case = ConversationalTestCase(
    turns=[LLMTestCase(input="...", actual_output="...")]
)
metric = ConversationRelevancyMetric(threshold=0.5)

metric.measure(convo_test_case)
print(metric.score)
print(metric.reason)
```

# Multimodal Metrics

## Image Coherence

The Image Coherence metric assesses the coherent alignment of images with their accompanying text, evaluating how effectively the visual content complements and enhances the textual narrative. deepeval's Image Coherence metric is a self-explaining MLLM-Eval, meaning it outputs a reason for its metric score.

```
from deepeval import evaluate
from deepeval.metrics import ImageCoherenceMetric
from deepeval.test_case import MLLMTestCase, MLLMImage

# Replace this with your actual MLLM application output
actual_output=[
    "1. Take the sheet of paper and fold it lengthwise",
```

```
    MLLMImage(url="./paper_plane_1", local=True),
    "2. Unfold the paper. Fold the top left and right corners towards
the center.",
    MLLMImage(url="./paper_plane_2", local=True),
    ...
]

metric = ImageCoherenceMetric(
    threshold=0.7,
    include_reason=True,
)
test_case = MLLMTestCase(
    input=["Provide step-by-step instructions on how to fold a paper
airplane."],
    actual_output=actual_output,
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)

# or evaluate test cases in bulk
evaluate([test_case], [metric])
```

## Image Helpfulness

The Image Helpfulness metric assesses how effectively images contribute to a user's comprehension of the text, including providing additional insights, clarifying complex ideas, or supporting textual details. deepeval's Image Helpfulness metric is a self-explaining MLLM-Eval, meaning it outputs a reason for its metric score.

```
from deepeval import evaluate
from deepeval.metrics import ImageHelpfulnessMetric
from deepeval.test_case import MLLMTestCase, MLLMImage

# Replace this with your actual MLLM application output
actual_output=[
    "1. Take the sheet of paper and fold it lengthwise",
    MLLMImage(url="./paper_plane_1", local=True),
    "2. Unfold the paper. Fold the top left and right corners towards
the center.",
    MLLMImage(url="./paper_plane_2", local=True),
    ...
]

metric = ImageHelpfulnessMetric(
    threshold=0.7,
    include_reason=True,
)
test_case = MLLMTestCase(
```

```
    input=["Provide step-by-step instructions on how to fold a paper
airplane."],
    actual_output=actual_output,
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)

# or evaluate test cases in bulk
evaluate([test_case], [metric])
```

## Image Reference

The Image Reference metric evaluates how accurately images are referred to or explained by accompanying text. deepeval's Image Reference metric is self-explaining within MLLM-Eval, meaning it provides a rationale for its assigned score.

```
from deepeval import evaluate
from deepeval.metrics import ImageReferenceMetric
from deepeval.test_case import MLLMTestCase, MLLMImage

# Replace this with your actual MLLM application output
actual_output=[
    "1. Take the sheet of paper and fold it lengthwise",
    MLLMImage(url="./paper_plane_1", local=True),
    "2. Unfold the paper. Fold the top left and right corners towards
the center.",
    MLLMImage(url="./paper_plane_2", local=True),
    ...
]

metric = ImageReferenceMetric(
    threshold=0.7,
    include_reason=True,
)
test_case = MLLMTestCase(
    input=["Provide step-by-step instructions on how to fold a paper
airplane."],
    actual_output=actual_output,
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)

# or evaluate test cases in bulk
evaluate([test_case], [metric])
```

## Text to Image

The Text to Image metric assesses the performance of image generation tasks by evaluating the quality of synthesized images based on semantic consistency and perceptual quality. deepeval's Text to Image metric is a self-explaining MLLM-Eval, meaning it outputs a reason for its metric score.

```
from deepeval import evaluate
from deepeval.metrics import TextToImageMetric
from deepeval.test_case import MLLMTestCase, MLLMImage

# Replace this with your actual MLLM application output
actual_output=[MLLMImage(url="https://shoe-images.com/edited-shoes",
local=False)]

metric = TextToImageMetric(
    threshold=0.7,
    include_reason=True,
)
test_case = MLLMTestCase(
    input=["Generate an image of a blue pair of shoes."],
    actual_output=actual_output,
    retrieval_context=retrieval_context
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)

# or evaluate test cases in bulk
evaluate([test_case], [metric])
```

## Image Editing

The Image Editing metric assesses the performance of image editing tasks by evaluating the quality of synthesized images based on semantic consistency and perceptual quality (similar to the TextToImageMetric). deepeval's Image Editing metric is a self-explaining MLLM-Eval, meaning it outputs a reason for its metric score.

```
from deepeval import evaluate
from deepeval.metrics import ImageEditingMetric
from deepeval.test_case import MLLMTestCase, MLLMImage

# Replace this with your actual MLLM application output
actual_output=[MLLMImage(url="https://shoe-images.com/edited-shoes",
local=False)]

metric = ImageEditingMetric(
    threshold=0.7,
    include_reason=True,
```

```
)
test_case = MLLMTestCase(
    input=["Change the color of the shoes to blue.",
MLLMImage(url="./shoes.png", local=True)],
    actual_output=actual_output,
    retrieval_context=retrieval_context
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)

# or evaluate test cases in bulk
evaluate([test_case], [metric])
```

## Multimodal Answer Relevancy

The multimodal answer relevancy metric measures the quality of your Multimodal RAG pipeline's generator by evaluating how relevant the actual_output of your MLLM application is compared to the provided input. deepeval's multimodal answer relevancy metric is a self-explaining MLLM-Eval, meaning it outputs a reason for its metric score.

```
from deepeval import evaluate
from deepeval.metrics import MultimodalAnswerRelevancyMetric
from deepeval.test_case import MLLMTestCase, MLLMImage

metric = AnswerRelevancyMetric()
test_case = MLLMTestCase(
    input=["Tell me about some landmarks in France"],
    actual_output=[
        "France is home to iconic landmarks like the Eiffel Tower in
Paris.",
        MLLMImage(...)
    ]
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)

# or evaluate test cases in bulk
evaluate([test_case], [metric])
```

## Multimodal Faithfulness

The multimodal faithfulness metric measures the quality of your RAG pipeline's generator by evaluating whether the actual_output factually aligns with the contents of your retrieval_context. deepeval's multimodal faithfulness metric is a self-explaining MLLM-Eval, meaning it outputs a reason for its metric score.

```
from deepeval import evaluate
from deepeval.metrics import MultimodalFaithfulnessMetric
from deepeval.test_case import MLLMTestCase, MLLMImage

metric = MultimodalFaithfulnessMetric()
test_case = MLLMTestCase(
    input=["Tell me about some landmarks in France"],
    actual_output=[
        "France is home to iconic landmarks like the Eiffel Tower in
Paris.",
        MLLMImage(...)
    ],
    retrieval_context=[
        MLLMImage(...),
        "The Eiffel Tower is a wrought-iron lattice tower built in the
late 19th century.",
        MLLMImage(...)
    ]
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)

# or evaluate test cases in bulk
evaluate([test_case], [metric])
```

## Multimodal Contextual Precision

The multimodal contextual precision metric measures your RAG pipeline's retriever by evaluating whether nodes in your retrieval_context that are relevant to the given input are ranked higher than irrelevant ones. deepeval's multimodal contextual precision metric is a self-explaining MLLM-Eval, meaning it outputs a reason for its metric score.

```
from deepeval import evaluate
from deepeval.metrics import MultimodalContextualPrecisionMetric
from deepeval.test_case import MLLMTestCase, MLLMImage

metric = MultimodalContextualPrecisionMetric()
test_case = MLLMTestCase(
    input=["Tell me about some landmarks in France"],
    actual_output=[
        "France is home to iconic landmarks like the Eiffel Tower in
Paris.",
```

```
        MLLMImage(...)
    ],
    expected_output=[
        "The Eiffel Tower is located in Paris, France.",
        MLLMImage(...)
    ],
    retrieval_context=[
        MLLMImage(...),
        "The Eiffel Tower is a wrought-iron lattice tower built in the
late 19th century.",
        MLLMImage(...)
    ],
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)

# or evaluate test cases in bulk
evaluate([test_case], [metric])
```

## Multimodal Contextual Recall

The multimodal contextual recall metric measures the quality of your RAG pipeline's retriever by evaluating the extent of which the retrieval_context aligns with the expected_output. deepeval's contextual recall metric is a self-explaining MLLM-Eval, meaning it outputs a reason for its metric score.

```
from deepeval import evaluate
from deepeval.metrics import MultimodalContextualRecallMetric
from deepeval.test_case import MLLMTestCase, MLLMImage

metric = MultimodalContextualRecallMetric()
test_case = MLLMTestCase(
    input=["Tell me about some landmarks in France"],
    actual_output=[
        "France is home to iconic landmarks like the Eiffel Tower in
Paris.",
        MLLMImage(...)
    ],
    expected_output=[
        "The Eiffel Tower is located in Paris, France.",
        MLLMImage(...)
    ],
    retrieval_context=[
        MLLMImage(...),
        "The Eiffel Tower is a wrought-iron lattice tower built in the
late 19th century.",
        MLLMImage(...)
    ],
```

```
)


metric.measure(test_case)
print(metric.score)
print(metric.reason)

# or evaluate test cases in bulk
evaluate([test_case], [metric])
```

## Multimodal Contextual Relevancy

The multimodal contextual relevancy metric measures the quality of your multimodal RAG pipeline's retriever by evaluating the overall relevance of the information presented in your retrieval_context for a given input. deepeval's multimodal contextual relevancy metric is a self-explaining MLLM-Eval, meaning it outputs a reason for its metric score.

```
from deepeval import evaluate
from deepeval.metrics import MultimodalContextualRelevancyMetric
from deepeval.test_case import MLLMTestCase, MLLMImage

metric = MultimodalContextualRelevancyMetric()
test_case = MLLMTestCase(
    input=["Tell me about some landmarks in France"],
    actual_output=[
        "France is home to iconic landmarks like the Eiffel Tower in
Paris.",
        MLLMImage(...)
    ],
    retrieval_context=[
        MLLMImage(...),
        "The Eiffel Tower is a wrought-iron lattice tower built in the
late 19th century.",
        MLLMImage(...)
    ],
)

metric.measure(test_case)
print(metric.score)
print(metric.reason)

# or evaluate test cases in bulk
evaluate([test_case], [metric])
```