```
import os
from unittest.mock import Mock, patch

import pytest
from dotenv import load_dotenv

from swarms import Cohere

# Load the environment variables
load_dotenv()
api_key = os.getenv("COHERE_API_KEY")


@pytest.fixture
def cohere_instance():
    return Cohere(cohere_api_key=api_key)


def test_cohere_custom_configuration(cohere_instance):
    # Test customizing Cohere configurations
    cohere_instance.model = "base"
    cohere_instance.temperature = 0.5
    cohere_instance.max_tokens = 100
    cohere_instance.k = 1
    cohere_instance.p = 0.8
    cohere_instance.frequency_penalty = 0.2
```

```
    cohere_instance.presence_penalty = 0.4
    response = cohere_instance("Customize configurations.")
    assert isinstance(response, str)


def test_cohere_api_error_handling(cohere_instance):
    # Test error handling when the API key is invalid
    cohere_instance.model = "base"
    cohere_instance.cohere_api_key = "invalid-api-key"
    with pytest.raises(Exception):
        cohere_instance("Error handling with invalid API key.")


def test_cohere_async_api_error_handling(cohere_instance):
    # Test async error handling when the API key is invalid
    cohere_instance.model = "base"
    cohere_instance.cohere_api_key = "invalid-api-key"
    with pytest.raises(Exception):
        cohere_instance.async_call(
            "Error handling with invalid API key."
        )


def test_cohere_stream_api_error_handling(cohere_instance):
    # Test error handling in streaming mode when the API key is invalid
    cohere_instance.model = "base"
```

```
    cohere_instance.cohere_api_key = "invalid-api-key"
    with pytest.raises(Exception):
        generator = cohere_instance.stream(
            "Error handling with invalid API key."
        )
        for token in generator:
            pass


def test_cohere_streaming_mode(cohere_instance):
    # Test the streaming mode for large text generation
    cohere_instance.model = "base"
    cohere_instance.streaming = True
    prompt = "Generate a lengthy text using streaming mode."
    generator = cohere_instance.stream(prompt)
    for token in generator:
        assert isinstance(token, str)


def test_cohere_streaming_mode_async(cohere_instance):
    # Test the async streaming mode for large text generation
    cohere_instance.model = "base"
    cohere_instance.streaming = True
    prompt = "Generate a lengthy text using async streaming mode."
    async_generator = cohere_instance.async_stream(prompt)
    for token in async_generator:
```

```
        assert isinstance(token, str)


def test_cohere_wrap_prompt(cohere_instance):
    prompt = "What is the meaning of life?"
    wrapped_prompt = cohere_instance._wrap_prompt(prompt)
    assert wrapped_prompt.startswith(cohere_instance.HUMAN_PROMPT)
    assert wrapped_prompt.endswith(cohere_instance.AI_PROMPT)


def test_cohere_convert_prompt(cohere_instance):
    prompt = "What is the meaning of life?"
    converted_prompt = cohere_instance.convert_prompt(prompt)
    assert converted_prompt.startswith(cohere_instance.HUMAN_PROMPT)
    assert converted_prompt.endswith(cohere_instance.AI_PROMPT)


def test_cohere_call_with_stop(cohere_instance):
    response = cohere_instance(
        "Translate to French.", stop=["stop1", "stop2"]
    )
    assert response == "Mocked Response from Cohere"


def test_cohere_stream_with_stop(cohere_instance):
    generator = cohere_instance.stream(
```

```
        "Write a story.", stop=["stop1", "stop2"]
    )
    for token in generator:
        assert isinstance(token, str)


def test_cohere_async_call_with_stop(cohere_instance):
    response = cohere_instance.async_call(
        "Tell me a joke.", stop=["stop1", "stop2"]
    )
    assert response == "Mocked Response from Cohere"


def test_cohere_async_stream_with_stop(cohere_instance):
    async_generator = cohere_instance.async_stream(
        "Translate to French.", stop=["stop1", "stop2"]
    )
    for token in async_generator:
        assert isinstance(token, str)


def test_cohere_get_num_tokens_with_count_tokens(cohere_instance):
    cohere_instance.count_tokens = Mock(return_value=10)
    text = "This is a test sentence."
    num_tokens = cohere_instance.get_num_tokens(text)
    assert num_tokens == 10
```

```
def test_cohere_get_num_tokens_without_count_tokens(cohere_instance):
    del cohere_instance.count_tokens
    with pytest.raises(NameError):
        text = "This is a test sentence."
        cohere_instance.get_num_tokens(text)


def test_cohere_wrap_prompt_without_human_ai_prompt(cohere_instance):
    del cohere_instance.HUMAN_PROMPT
    del cohere_instance.AI_PROMPT
    prompt = "What is the meaning of life?"
    with pytest.raises(NameError):
        cohere_instance._wrap_prompt(prompt)


def test_base_cohere_import():
    with patch.dict("sys.modules", {"cohere": None}):
        with pytest.raises(ImportError):
            pass


def test_base_cohere_validate_environment():
    values = {
        "cohere_api_key": "my-api-key",
```

```
        "user_agent": "langchain",
    }
    validated_values = Cohere.validate_environment(values)
    assert "client" in validated_values
    assert "async_client" in validated_values


def test_base_cohere_validate_environment_without_cohere():
    values = {
        "cohere_api_key": "my-api-key",
        "user_agent": "langchain",
    }
    with patch.dict("sys.modules", {"cohere": None}):
        with pytest.raises(ImportError):
            Cohere.validate_environment(values)


# Test cases for benchmarking generations with various models
def test_cohere_generate_with_command_light(cohere_instance):
    cohere_instance.model = "command-light"
    response = cohere_instance(
        "Generate text with Command Light model."
    )
    assert response.startswith(
        "Generated text with Command Light model"
    )
```

```
def test_cohere_generate_with_command(cohere_instance):
    cohere_instance.model = "command"
    response = cohere_instance("Generate text with Command model.")
    assert response.startswith("Generated text with Command model")


def test_cohere_generate_with_base_light(cohere_instance):
    cohere_instance.model = "base-light"
    response = cohere_instance("Generate text with Base Light model.")
    assert response.startswith("Generated text with Base Light model")


def test_cohere_generate_with_base(cohere_instance):
    cohere_instance.model = "base"
    response = cohere_instance("Generate text with Base model.")
    assert response.startswith("Generated text with Base model")


def test_cohere_generate_with_embed_english_v2(cohere_instance):
    cohere_instance.model = "embed-english-v2.0"
    response = cohere_instance(
        "Generate embeddings with English v2.0 model."
    )
    assert response.startswith(
```

```
        "Generated embeddings with English v2.0 model"
    )


def test_cohere_generate_with_embed_english_light_v2(cohere_instance):
    cohere_instance.model = "embed-english-light-v2.0"
    response = cohere_instance(
        "Generate embeddings with English Light v2.0 model."
    )
    assert response.startswith(
        "Generated embeddings with English Light v2.0 model"
    )


def test_cohere_generate_with_embed_multilingual_v2(cohere_instance):
    cohere_instance.model = "embed-multilingual-v2.0"
    response = cohere_instance(
        "Generate embeddings with Multilingual v2.0 model."
    )
    assert response.startswith(
        "Generated embeddings with Multilingual v2.0 model"
    )


def test_cohere_generate_with_embed_english_v3(cohere_instance):
    cohere_instance.model = "embed-english-v3.0"
```

```
    response = cohere_instance(
        "Generate embeddings with English v3.0 model."
    )
    assert response.startswith(
        "Generated embeddings with English v3.0 model"
    )


def test_cohere_generate_with_embed_english_light_v3(cohere_instance):
    cohere_instance.model = "embed-english-light-v3.0"
    response = cohere_instance(
        "Generate embeddings with English Light v3.0 model."
    )
    assert response.startswith(
        "Generated embeddings with English Light v3.0 model"
    )


def test_cohere_generate_with_embed_multilingual_v3(cohere_instance):
    cohere_instance.model = "embed-multilingual-v3.0"
    response = cohere_instance(
        "Generate embeddings with Multilingual v3.0 model."
    )
    assert response.startswith(
        "Generated embeddings with Multilingual v3.0 model"
    )
```

```
def test_cohere_generate_with_embed_multilingual_light_v3(
    cohere_instance,
):
    cohere_instance.model = "embed-multilingual-light-v3.0"
    response = cohere_instance(
        "Generate embeddings with Multilingual Light v3.0 model."
    )
    assert response.startswith(
        "Generated embeddings with Multilingual Light v3.0 model"
    )


# Add more test cases to benchmark other models and functionalities


def test_cohere_call_with_command_model(cohere_instance):
    cohere_instance.model = "command"
    response = cohere_instance("Translate to French.")
    assert isinstance(response, str)


def test_cohere_call_with_base_model(cohere_instance):
    cohere_instance.model = "base"
    response = cohere_instance("Translate to French.")
```

```
    assert isinstance(response, str)


def test_cohere_call_with_embed_english_v2_model(cohere_instance):
    cohere_instance.model = "embed-english-v2.0"
    response = cohere_instance("Translate to French.")
    assert isinstance(response, str)


def test_cohere_call_with_embed_english_v3_model(cohere_instance):
    cohere_instance.model = "embed-english-v3.0"
    response = cohere_instance("Translate to French.")
    assert isinstance(response, str)


def test_cohere_call_with_embed_multilingual_v2_model(
    cohere_instance,
):
    cohere_instance.model = "embed-multilingual-v2.0"
    response = cohere_instance("Translate to French.")
    assert isinstance(response, str)


def test_cohere_call_with_embed_multilingual_v3_model(
    cohere_instance,
):
```

```
    cohere_instance.model = "embed-multilingual-v3.0"
    response = cohere_instance("Translate to French.")
    assert isinstance(response, str)


def test_cohere_call_with_invalid_model(cohere_instance):
    cohere_instance.model = "invalid-model"
    with pytest.raises(ValueError):
        cohere_instance("Translate to French.")


def test_cohere_call_with_long_prompt(cohere_instance):
    prompt = "This is a very long prompt. " * 100
    response = cohere_instance(prompt)
    assert isinstance(response, str)


def test_cohere_call_with_max_tokens_limit_exceeded(cohere_instance):
    cohere_instance.max_tokens = 10
    prompt = (
        "This is a test prompt that will exceed the max tokens limit."
    )
    with pytest.raises(ValueError):
        cohere_instance(prompt)
```

```
def test_cohere_stream_with_command_model(cohere_instance):
    cohere_instance.model = "command"
    generator = cohere_instance.stream("Write a story.")
    for token in generator:
        assert isinstance(token, str)


def test_cohere_stream_with_base_model(cohere_instance):
    cohere_instance.model = "base"
    generator = cohere_instance.stream("Write a story.")
    for token in generator:
        assert isinstance(token, str)


def test_cohere_stream_with_embed_english_v2_model(cohere_instance):
    cohere_instance.model = "embed-english-v2.0"
    generator = cohere_instance.stream("Write a story.")
    for token in generator:
        assert isinstance(token, str)


def test_cohere_stream_with_embed_english_v3_model(cohere_instance):
    cohere_instance.model = "embed-english-v3.0"
    generator = cohere_instance.stream("Write a story.")
    for token in generator:
        assert isinstance(token, str)
```

```
def test_cohere_stream_with_embed_multilingual_v2_model(
    cohere_instance,
):
    cohere_instance.model = "embed-multilingual-v2.0"
    generator = cohere_instance.stream("Write a story.")
    for token in generator:
        assert isinstance(token, str)


def test_cohere_stream_with_embed_multilingual_v3_model(
    cohere_instance,
):
    cohere_instance.model = "embed-multilingual-v3.0"
    generator = cohere_instance.stream("Write a story.")
    for token in generator:
        assert isinstance(token, str)


def test_cohere_async_call_with_command_model(cohere_instance):
    cohere_instance.model = "command"
    response = cohere_instance.async_call("Translate to French.")
    assert isinstance(response, str)
```

```
def test_cohere_async_call_with_base_model(cohere_instance):
    cohere_instance.model = "base"
    response = cohere_instance.async_call("Translate to French.")
    assert isinstance(response, str)


def test_cohere_async_call_with_embed_english_v2_model(
    cohere_instance,
):
    cohere_instance.model = "embed-english-v2.0"
    response = cohere_instance.async_call("Translate to French.")
    assert isinstance(response, str)


def test_cohere_async_call_with_embed_english_v3_model(
    cohere_instance,
):
    cohere_instance.model = "embed-english-v3.0"
    response = cohere_instance.async_call("Translate to French.")
    assert isinstance(response, str)


def test_cohere_async_call_with_embed_multilingual_v2_model(
    cohere_instance,
):
    cohere_instance.model = "embed-multilingual-v2.0"
```

```
    response = cohere_instance.async_call("Translate to French.")
    assert isinstance(response, str)


def test_cohere_async_call_with_embed_multilingual_v3_model(
    cohere_instance,
):
    cohere_instance.model = "embed-multilingual-v3.0"
    response = cohere_instance.async_call("Translate to French.")
    assert isinstance(response, str)


def test_cohere_async_stream_with_command_model(cohere_instance):
    cohere_instance.model = "command"
    async_generator = cohere_instance.async_stream("Write a story.")
    for token in async_generator:
        assert isinstance(token, str)


def test_cohere_async_stream_with_base_model(cohere_instance):
    cohere_instance.model = "base"
    async_generator = cohere_instance.async_stream("Write a story.")
    for token in async_generator:
        assert isinstance(token, str)
```

```
def test_cohere_async_stream_with_embed_english_v2_model(
    cohere_instance,
):
    cohere_instance.model = "embed-english-v2.0"
    async_generator = cohere_instance.async_stream("Write a story.")
    for token in async_generator:
        assert isinstance(token, str)


def test_cohere_async_stream_with_embed_english_v3_model(
    cohere_instance,
):
    cohere_instance.model = "embed-english-v3.0"
    async_generator = cohere_instance.async_stream("Write a story.")
    for token in async_generator:
        assert isinstance(token, str)


def test_cohere_async_stream_with_embed_multilingual_v2_model(
    cohere_instance,
):
    cohere_instance.model = "embed-multilingual-v2.0"
    async_generator = cohere_instance.async_stream("Write a story.")
    for token in async_generator:
        assert isinstance(token, str)
```

```
def test_cohere_async_stream_with_embed_multilingual_v3_model(
    cohere_instance,
):
    cohere_instance.model = "embed-multilingual-v3.0"
    async_generator = cohere_instance.async_stream("Write a story.")
    for token in async_generator:
        assert isinstance(token, str)


def test_cohere_representation_model_embedding(cohere_instance):
    # Test using the Representation model for text embedding
    cohere_instance.model = "embed-english-v3.0"
    embedding = cohere_instance.embed(
        "Generate an embedding for this text."
    )
    assert isinstance(embedding, list)
    assert len(embedding) > 0


def test_cohere_representation_model_classification(cohere_instance):
    # Test using the Representation model for text classification
    cohere_instance.model = "embed-english-v3.0"
    classification = cohere_instance.classify("Classify this text.")
    assert isinstance(classification, dict)
    assert "class" in classification
```

```
    assert "score" in classification


def test_cohere_representation_model_language_detection(
    cohere_instance,
):
    # Test using the Representation model for language detection
    cohere_instance.model = "embed-english-v3.0"
    language = cohere_instance.detect_language(
        "Detect the language of this text."
    )
    assert isinstance(language, str)


def test_cohere_representation_model_max_tokens_limit_exceeded(
    cohere_instance,
):
    # Test handling max tokens limit exceeded error
    cohere_instance.model = "embed-english-v3.0"
    cohere_instance.max_tokens = 10
    prompt = (
        "This is a test prompt that will exceed the max tokens limit."
    )
    with pytest.raises(ValueError):
        cohere_instance.embed(prompt)
```

```
# Add more production-grade test cases based on real-world scenarios



def test_cohere_representation_model_multilingual_embedding(
    cohere_instance,
):
    # Test using the Representation model for multilingual text embedding
    cohere_instance.model = "embed-multilingual-v3.0"
    embedding = cohere_instance.embed(
        "Generate multilingual embeddings."
    )
    assert isinstance(embedding, list)
    assert len(embedding) > 0



def test_cohere_representation_model_multilingual_classification(
    cohere_instance,
):
    # Test using the Representation model for multilingual text classification
    cohere_instance.model = "embed-multilingual-v3.0"
    classification = cohere_instance.classify(
        "Classify multilingual text."
    )
    assert isinstance(classification, dict)
    assert "class" in classification
```

```
    assert "score" in classification


def test_cohere_representation_model_multilingual_language_detection(
    cohere_instance,
):
    # Test using the Representation model for multilingual language detection
    cohere_instance.model = "embed-multilingual-v3.0"
    language = cohere_instance.detect_language(
        "Detect the language of multilingual text."
    )
    assert isinstance(language, str)


def test_cohere_representation_model_multilingual_max_tokens_limit_exceeded(
    cohere_instance,
):
    # Test handling max tokens limit exceeded error for multilingual model
    cohere_instance.model = "embed-multilingual-v3.0"
    cohere_instance.max_tokens = 10
    prompt = (
        "This is a test prompt that will exceed the max tokens limit"
        " for multilingual model."
    )
    with pytest.raises(ValueError):
        cohere_instance.embed(prompt)
```

```
def test_cohere_representation_model_multilingual_light_embedding(
    cohere_instance,
):
    # Test using the Representation model for multilingual light text embedding
    cohere_instance.model = "embed-multilingual-light-v3.0"
    embedding = cohere_instance.embed(
        "Generate multilingual light embeddings."
    )
    assert isinstance(embedding, list)
    assert len(embedding) > 0



def test_cohere_representation_model_multilingual_light_classification(
    cohere_instance,
):
    # Test using the Representation model for multilingual light text classification
    cohere_instance.model = "embed-multilingual-light-v3.0"
    classification = cohere_instance.classify(
        "Classify multilingual light text."
    )
    assert isinstance(classification, dict)
    assert "class" in classification
    assert "score" in classification
```

```
def test_cohere_representation_model_multilingual_light_language_detection(
    cohere_instance,
):
    # Test using the Representation model for multilingual light language detection
    cohere_instance.model = "embed-multilingual-light-v3.0"
    language = cohere_instance.detect_language(
        "Detect the language of multilingual light text."
    )
    assert isinstance(language, str)


def test_cohere_representation_model_multilingual_light_max_tokens_limit_exceeded(
    cohere_instance,
):
    # Test handling max tokens limit exceeded error for multilingual light model
    cohere_instance.model = "embed-multilingual-light-v3.0"
    cohere_instance.max_tokens = 10
    prompt = (
        "This is a test prompt that will exceed the max tokens limit"
        " for multilingual light model."
    )
    with pytest.raises(ValueError):
        cohere_instance.embed(prompt)
```

```
def test_cohere_command_light_model(cohere_instance):
    # Test using the Command Light model for text generation
    cohere_instance.model = "command-light"
    response = cohere_instance(
        "Generate text using Command Light model."
    )
    assert isinstance(response, str)


def test_cohere_base_light_model(cohere_instance):
    # Test using the Base Light model for text generation
    cohere_instance.model = "base-light"
    response = cohere_instance(
        "Generate text using Base Light model."
    )
    assert isinstance(response, str)


def test_cohere_generate_summarize_endpoint(cohere_instance):
    # Test using the Co.summarize() endpoint for text summarization
    cohere_instance.model = "command"
    response = cohere_instance.summarize("Summarize this text.")
    assert isinstance(response, str)


def test_cohere_representation_model_english_embedding(
```

```
    cohere_instance,
):
    # Test using the Representation model for English text embedding
    cohere_instance.model = "embed-english-v3.0"
    embedding = cohere_instance.embed("Generate English embeddings.")
    assert isinstance(embedding, list)
    assert len(embedding) > 0


def test_cohere_representation_model_english_classification(
    cohere_instance,
):
    # Test using the Representation model for English text classification
    cohere_instance.model = "embed-english-v3.0"
    classification = cohere_instance.classify(
        "Classify English text."
    )
    assert isinstance(classification, dict)
    assert "class" in classification
    assert "score" in classification


def test_cohere_representation_model_english_language_detection(
    cohere_instance,
):
    # Test using the Representation model for English language detection
```

```
    cohere_instance.model = "embed-english-v3.0"
    language = cohere_instance.detect_language(
        "Detect the language of English text."
    )
    assert isinstance(language, str)


def test_cohere_representation_model_english_max_tokens_limit_exceeded(
    cohere_instance,
):
    # Test handling max tokens limit exceeded error for English model
    cohere_instance.model = "embed-english-v3.0"
    cohere_instance.max_tokens = 10
    prompt = (
        "This is a test prompt that will exceed the max tokens limit"
        " for English model."
    )
    with pytest.raises(ValueError):
        cohere_instance.embed(prompt)


def test_cohere_representation_model_english_light_embedding(
    cohere_instance,
):
    # Test using the Representation model for English light text embedding
    cohere_instance.model = "embed-english-light-v3.0"
```

```
    embedding = cohere_instance.embed(
        "Generate English light embeddings."
    )
    assert isinstance(embedding, list)
    assert len(embedding) > 0


def test_cohere_representation_model_english_light_classification(
    cohere_instance,
):
    # Test using the Representation model for English light text classification
    cohere_instance.model = "embed-english-light-v3.0"
    classification = cohere_instance.classify(
        "Classify English light text."
    )
    assert isinstance(classification, dict)
    assert "class" in classification
    assert "score" in classification


def test_cohere_representation_model_english_light_language_detection(
    cohere_instance,
):
    # Test using the Representation model for English light language detection
    cohere_instance.model = "embed-english-light-v3.0"
    language = cohere_instance.detect_language(
```

```
        "Detect the language of English light text."
    )
    assert isinstance(language, str)


def test_cohere_representation_model_english_light_max_tokens_limit_exceeded(
    cohere_instance,
):
    # Test handling max tokens limit exceeded error for English light model
    cohere_instance.model = "embed-english-light-v3.0"
    cohere_instance.max_tokens = 10
    prompt = (
        "This is a test prompt that will exceed the max tokens limit"
        " for English light model."
    )
    with pytest.raises(ValueError):
        cohere_instance.embed(prompt)


def test_cohere_command_model(cohere_instance):
    # Test using the Command model for text generation
    cohere_instance.model = "command"
    response = cohere_instance(
        "Generate text using the Command model."
    )
    assert isinstance(response, str)
```

```
# Add more production-grade test cases based on real-world scenarios


def test_cohere_invalid_model(cohere_instance):
    # Test using an invalid model name
    cohere_instance.model = "invalid-model"
    with pytest.raises(ValueError):
        cohere_instance("Generate text using an invalid model.")


def test_cohere_base_model_generation_with_max_tokens(
    cohere_instance,
):
    # Test generating text using the base model with a specified max_tokens limit
    cohere_instance.model = "base"
    cohere_instance.max_tokens = 20
    prompt = "Generate text with max_tokens limit."
    response = cohere_instance(prompt)
    assert len(response.split()) <= 20


def test_cohere_command_light_generation_with_stop(cohere_instance):
    # Test generating text using the command-light model with stop words
    cohere_instance.model = "command-light"
```

```
prompt = "Generate text with stop words."

stop = ["stop", "words"]

response = cohere_instance(prompt, stop=stop)

assert all(word not in response for word in stop)
```