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

import pytest

from swarm_models.anthropic import Anthropic


# Mock the Anthropic API client for testing
class MockAnthropicClient:
    def __init__(self, *args, **kwargs):
        pass

    def completions_create(
        self, prompt, stop_sequences, stream, **kwargs
    ):
        return MockAnthropicResponse()


@pytest.fixture
def mock_anthropic_env():
    os.environ["ANTHROPIC_API_URL"] = "https://test.anthropic.com"
    os.environ["ANTHROPIC_API_KEY"] = "test_api_key"
    yield
    del os.environ["ANTHROPIC_API_URL"]
    del os.environ["ANTHROPIC_API_KEY"]
```

```
@pytest.fixture
def mock_requests_post():
    with patch("requests.post") as mock_post:
        yield mock_post


@pytest.fixture
def anthropic_instance():
    return Anthropic(model="test-model")


def test_anthropic_init_default_values(anthropic_instance):
    assert anthropic_instance.model == "test-model"
    assert anthropic_instance.max_tokens_to_sample == 256
    assert anthropic_instance.temperature is None
    assert anthropic_instance.top_k is None
    assert anthropic_instance.top_p is None
    assert anthropic_instance.streaming is False
    assert anthropic_instance.default_request_timeout == 600
    assert (
        anthropic_instance.anthropic_api_url
        == "https://test.anthropic.com"
    )
    assert anthropic_instance.anthropic_api_key == "test_api_key"
```

```
def test_anthropic_init_custom_values():
    anthropic_instance = Anthropic(
        model="custom-model",
        max_tokens_to_sample=128,
        temperature=0.8,
        top_k=5,
        top_p=0.9,
        streaming=True,
        default_request_timeout=300,
    )
    assert anthropic_instance.model == "custom-model"
    assert anthropic_instance.max_tokens_to_sample == 128
    assert anthropic_instance.temperature == 0.8
    assert anthropic_instance.top_k == 5
    assert anthropic_instance.top_p == 0.9
    assert anthropic_instance.streaming is True
    assert anthropic_instance.default_request_timeout == 300


def test_anthropic_default_params(anthropic_instance):
    default_params = anthropic_instance._default_params()
    assert default_params == {
        "max_tokens_to_sample": 256,
        "model": "test-model",
    }



def test_anthropic_run(
    mock_anthropic_env, mock_requests_post, anthropic_instance
):
    mock_response = Mock()
    mock_response.json.return_value = {"completion": "Generated text"}
    mock_requests_post.return_value = mock_response

    task = "Generate text"
    stop = ["stop1", "stop2"]

    completion = anthropic_instance.run(task, stop)

    assert completion == "Generated text"
    mock_requests_post.assert_called_once_with(
        "https://test.anthropic.com/completions",
        headers={"Authorization": "Bearer test_api_key"},
        json={
            "prompt": task,
            "stop_sequences": stop,
            "max_tokens_to_sample": 256,
            "model": "test-model",
        },
        timeout=600,
    )



def test_anthropic_call(
    mock_anthropic_env, mock_requests_post, anthropic_instance
):
    mock_response = Mock()
    mock_response.json.return_value = {"completion": "Generated text"}
    mock_requests_post.return_value = mock_response

    task = "Generate text"
    stop = ["stop1", "stop2"]

    completion = anthropic_instance(task, stop)

    assert completion == "Generated text"
    mock_requests_post.assert_called_once_with(
        "https://test.anthropic.com/completions",
        headers={"Authorization": "Bearer test_api_key"},
        json={
            "prompt": task,
            "stop_sequences": stop,
            "max_tokens_to_sample": 256,
            "model": "test-model",
        },
        timeout=600,
    )


def test_anthropic_exception_handling(
    mock_anthropic_env, mock_requests_post, anthropic_instance
):
    mock_response = Mock()
    mock_response.json.return_value = {"error": "An error occurred"}
    mock_requests_post.return_value = mock_response

    task = "Generate text"
    stop = ["stop1", "stop2"]

    with pytest.raises(Exception) as excinfo:
        anthropic_instance(task, stop)

    assert "An error occurred" in str(excinfo.value)


class MockAnthropicResponse:
    def __init__(self):
        self.completion = "Mocked Response from Anthropic"


def test_anthropic_instance_creation(anthropic_instance):
    assert isinstance(anthropic_instance, Anthropic)
```

```python
def test_anthropic_call_method(anthropic_instance):

    response = anthropic_instance("What is the meaning of life?")

    assert response == "Mocked Response from Anthropic"




def test_anthropic_stream_method(anthropic_instance):

    generator = anthropic_instance.stream("Write a story.")

    for token in generator:

        assert isinstance(token, str)




def test_anthropic_async_call_method(anthropic_instance):

    response = anthropic_instance.async_call("Tell me a joke.")

    assert response == "Mocked Response from Anthropic"




def test_anthropic_async_stream_method(anthropic_instance):

    async_generator = anthropic_instance.async_stream(

        "Translate to French."

    )

    for token in async_generator:

        assert isinstance(token, str)
```

```
def test_anthropic_get_num_tokens(anthropic_instance):
    text = "This is a test sentence."
    num_tokens = anthropic_instance.get_num_tokens(text)
    assert num_tokens > 0


# Add more test cases to cover other functionalities and edge cases of the Anthropic class


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


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


def test_anthropic_call_with_stop(anthropic_instance):
    response = anthropic_instance(
        "Translate to French.", stop=["stop1", "stop2"]
    )
    assert response == "Mocked Response from Anthropic"


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


def test_anthropic_async_call_with_stop(anthropic_instance):
    response = anthropic_instance.async_call(
        "Tell me a joke.", stop=["stop1", "stop2"]
    )
    assert response == "Mocked Response from Anthropic"


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



def test_anthropic_get_num_tokens_with_count_tokens(
    anthropic_instance,
):
    anthropic_instance.count_tokens = Mock(return_value=10)
    text = "This is a test sentence."
    num_tokens = anthropic_instance.get_num_tokens(text)
    assert num_tokens == 10



def test_anthropic_get_num_tokens_without_count_tokens(
    anthropic_instance,
):
    del anthropic_instance.count_tokens
    with pytest.raises(NameError):
        text = "This is a test sentence."
        anthropic_instance.get_num_tokens(text)



def test_anthropic_wrap_prompt_without_human_ai_prompt(
    anthropic_instance,
):
    del anthropic_instance.HUMAN_PROMPT
    del anthropic_instance.AI_PROMPT
```

```
prompt = "What is the meaning of life?"

with pytest.raises(NameError):

    anthropic_instance._wrap_prompt(prompt)
```