```
import pytest
from PIL import Image

from swarm_models.ssd_1b import SSD1B


# Create fixtures if needed
@pytest.fixture
def ssd1b_model():
    return SSD1B()


# Basic tests for model initialization and method call
def test_ssd1b_model_initialization(ssd1b_model):
    assert ssd1b_model is not None


def test_ssd1b_call(ssd1b_model):
    task = "A painting of a dog"
    neg_prompt = "ugly, blurry, poor quality"
    image_url = ssd1b_model(task, neg_prompt)
    assert isinstance(image_url, str)
    assert image_url.startswith(
        "https://"
    )  # Assuming it starts with "https://"
```

```
# Add more tests for various aspects of the class and methods



# Example of a parameterized test for different tasks
@pytest.mark.parametrize(
    "task", ["A painting of a cat", "A painting of a tree"]
)
def test_ssd1b_parameterized_task(ssd1b_model, task):
    image_url = ssd1b_model(task)
    assert isinstance(image_url, str)
    assert image_url.startswith(
        "https://"
    )  # Assuming it starts with "https://"



# Example of a test using mocks to isolate units of code
def test_ssd1b_with_mock(ssd1b_model, mocker):
    mocker.patch(
        "your_module.StableDiffusionXLPipeline"
    )  # Mock the pipeline
    task = "A painting of a cat"
    image_url = ssd1b_model(task)
    assert isinstance(image_url, str)
    assert image_url.startswith(
        "https://"
```

```
    )  # Assuming it starts with "https://"


def test_ssd1b_call_with_cache(ssd1b_model):
    task = "A painting of a dog"
    neg_prompt = "ugly, blurry, poor quality"
    image_url1 = ssd1b_model(task, neg_prompt)
    image_url2 = ssd1b_model(task, neg_prompt)  # Should use cache
    assert image_url1 == image_url2


def test_ssd1b_invalid_task(ssd1b_model):
    invalid_task = ""
    with pytest.raises(ValueError):
        ssd1b_model(invalid_task)


def test_ssd1b_failed_api_call(ssd1b_model, mocker):
    mocker.patch(
        "your_module.StableDiffusionXLPipeline"
    )  # Mock the pipeline to raise an exception
    task = "A painting of a cat"
    with pytest.raises(Exception):
        ssd1b_model(task)
```

```
def test_ssd1b_process_batch_concurrently(ssd1b_model):
    tasks = [
        "A painting of a dog",
        "A beautiful sunset",
        "A portrait of a person",
    ]
    results = ssd1b_model.process_batch_concurrently(tasks)
    assert isinstance(results, list)
    assert len(results) == len(tasks)


def test_ssd1b_process_empty_batch_concurrently(ssd1b_model):
    tasks = []
    results = ssd1b_model.process_batch_concurrently(tasks)
    assert isinstance(results, list)
    assert len(results) == 0


def test_ssd1b_download_image(ssd1b_model):
    task = "A painting of a dog"
    neg_prompt = "ugly, blurry, poor quality"
    image_url = ssd1b_model(task, neg_prompt)
    img = ssd1b_model._download_image(image_url)
    assert isinstance(img, Image.Image)
```

```
def test_ssd1b_generate_uuid(ssd1b_model):
    uuid_str = ssd1b_model._generate_uuid()
    assert isinstance(uuid_str, str)
    assert len(uuid_str) == 36  # UUID format


def test_ssd1b_rate_limited_call(ssd1b_model):
    task = "A painting of a dog"
    image_url = ssd1b_model.rate_limited_call(task)
    assert isinstance(image_url, str)
    assert image_url.startswith("https://")


# Test cases for additional scenarios and behaviors
def test_ssd1b_dashboard_printing(ssd1b_model, capsys):
    ssd1b_model.dashboard = True
    ssd1b_model.print_dashboard()
    captured = capsys.readouterr()
    assert "SSD1B Dashboard:" in captured.out


def test_ssd1b_generate_image_name(ssd1b_model):
    task = "A painting of a dog"
    img_name = ssd1b_model._generate_image_name(task)
    assert isinstance(img_name, str)
    assert len(img_name) > 0
```

```
def test_ssd1b_set_width_height(ssd1b_model, mocker):
    img = mocker.MagicMock()
    width, height = 800, 600
    result = ssd1b_model.set_width_height(img, width, height)
    assert result == img.resize.return_value


def test_ssd1b_read_img(ssd1b_model, mocker):
    img = mocker.MagicMock()
    result = ssd1b_model.read_img(img)
    assert result == img.open.return_value


def test_ssd1b_convert_to_bytesio(ssd1b_model, mocker):
    img = mocker.MagicMock()
    img_format = "PNG"
    result = ssd1b_model.convert_to_bytesio(img, img_format)
    assert isinstance(result, bytes)


def test_ssd1b_save_image(ssd1b_model, mocker, tmp_path):
    img = mocker.MagicMock()
    img_name = "test.png"
    save_path = tmp_path / img_name
```

```
    ssd1b_model._download_image(img, img_name, save_path)
    assert save_path.exists()


def test_ssd1b_repr_str(ssd1b_model):
    task = "A painting of a dog"
    image_url = ssd1b_model(task)
    assert repr(ssd1b_model) == f"SSD1B(image_url={image_url})"
    assert str(ssd1b_model) == f"SSD1B(image_url={image_url})"
```