```
from unittest.mock import patch

import pytest
import torch

from swarm_models.idefics import (
    AutoProcessor,
    Idefics,
    IdeficsForVisionText2Text,
)



@pytest.fixture
def idefics_instance():
    with patch(
        "torch.cuda.is_available", return_value=False
    ):  # Assuming tests are run on CPU for simplicity
        instance = Idefics()
    return instance



# Basic Tests
def test_init_default(idefics_instance):
    assert idefics_instance.device == "cpu"
    assert idefics_instance.max_length == 100
    assert not idefics_instance.chat_history
```

```
@pytest.mark.parametrize(
    "device,expected",
    [
        (None, "cpu"),
        ("cuda", "cuda"),
        ("cpu", "cpu"),
    ],
)
def test_init_device(device, expected):
    with patch(
        "torch.cuda.is_available",
        return_value=True if expected == "cuda" else False,
    ):
        instance = Idefics(device=device)
    assert instance.device == expected



# Test `run` method
def test_run(idefics_instance):
    prompts = [["User: Test"]]
    with patch.object(
        idefics_instance, "processor"
    ) as mock_processor, patch.object(
        idefics_instance, "model"
```

```
    ) as mock_model:
        mock_processor.return_value = {
            "input_ids": torch.tensor([1, 2, 3])
        }
        mock_model.generate.return_value = torch.tensor([1, 2, 3])
        mock_processor.batch_decode.return_value = ["Test"]

        result = idefics_instance.run(prompts)

    assert result == ["Test"]


# Test `__call__` method (using the same logic as run for simplicity)
def test_call(idefics_instance):
    prompts = [["User: Test"]]
    with patch.object(
        idefics_instance, "processor"
    ) as mock_processor, patch.object(
        idefics_instance, "model"
    ) as mock_model:
        mock_processor.return_value = {
            "input_ids": torch.tensor([1, 2, 3])
        }
        mock_model.generate.return_value = torch.tensor([1, 2, 3])
        mock_processor.batch_decode.return_value = ["Test"]
```

```
        result = idefics_instance(prompts)

    assert result == ["Test"]


# Test `chat` method
def test_chat(idefics_instance):
    user_input = "User: Hello"
    response = "Model: Hi there!"
    with patch.object(
        idefics_instance, "run", return_value=[response]
    ):
        result = idefics_instance.chat(user_input)

    assert result == response
    assert idefics_instance.chat_history == [user_input, response]


# Test `set_checkpoint` method
def test_set_checkpoint(idefics_instance):
    new_checkpoint = "new_checkpoint"
    with patch.object(
        IdeficsForVisionText2Text, "from_pretrained"
    ) as mock_from_pretrained, patch.object(
        AutoProcessor, "from_pretrained"
    ):
```

```
        idefics_instance.set_checkpoint(new_checkpoint)

    mock_from_pretrained.assert_called_with(
        new_checkpoint, torch_dtype=torch.bfloat16
    )


# Test `set_device` method
def test_set_device(idefics_instance):
    new_device = "cuda"
    with patch.object(idefics_instance.model, "to"):
        idefics_instance.set_device(new_device)

    assert idefics_instance.device == new_device


# Test `set_max_length` method
def test_set_max_length(idefics_instance):
    new_length = 150
    idefics_instance.set_max_length(new_length)
    assert idefics_instance.max_length == new_length


# Test `clear_chat_history` method
def test_clear_chat_history(idefics_instance):
    idefics_instance.chat_history = ["User: Test", "Model: Response"]
```

```
    idefics_instance.clear_chat_history()
    assert not idefics_instance.chat_history


# Exception Tests
def test_run_with_empty_prompts(idefics_instance):
    with pytest.raises(
        Exception
    ):  # Replace Exception with the actual exception that may arise for an empty prompt.
        idefics_instance.run([])


# Test `run` method with batched_mode set to False
def test_run_batched_mode_false(idefics_instance):
    task = "User: Test"
    with patch.object(
        idefics_instance, "processor"
    ) as mock_processor, patch.object(
        idefics_instance, "model"
    ) as mock_model:
        mock_processor.return_value = {
            "input_ids": torch.tensor([1, 2, 3])
        }
        mock_model.generate.return_value = torch.tensor([1, 2, 3])
        mock_processor.batch_decode.return_value = ["Test"]
```

```
        idefics_instance.batched_mode = False
        result = idefics_instance.run(task)

    assert result == ["Test"]


# Test `run` method with an exception
def test_run_with_exception(idefics_instance):
    task = "User: Test"
    with patch.object(
        idefics_instance, "processor"
    ) as mock_processor:
        mock_processor.side_effect = Exception("Test exception")
        with pytest.raises(Exception):
            idefics_instance.run(task)


# Test `set_model_name` method
def test_set_model_name(idefics_instance):
    new_model_name = "new_model_name"
    with patch.object(
        IdeficsForVisionText2Text, "from_pretrained"
    ) as mock_from_pretrained, patch.object(
        AutoProcessor, "from_pretrained"
    ):
        idefics_instance.set_model_name(new_model_name)
```

```
    assert idefics_instance.model_name == new_model_name
    mock_from_pretrained.assert_called_with(
        new_model_name, torch_dtype=torch.bfloat16
    )


# Test `__init__` method with device set to None
def test_init_device_none():
    with patch(
        "torch.cuda.is_available",
        return_value=False,
    ):
        instance = Idefics(device=None)
    assert instance.device == "cpu"


# Test `__init__` method with device set to "cuda"
def test_init_device_cuda():
    with patch(
        "torch.cuda.is_available",
        return_value=True,
    ):
        instance = Idefics(device="cuda")
    assert instance.device == "cuda"
```