tests/test\_code\_improver.py

import unittest

from unittest.mock import patch

from utils.code\_improver import improve\_code

class TestCodeImprover(unittest.TestCase):

@patch('utils.code\_improver.openai.ChatCompletion.create')

def test\_improve\_code\_success(self, mock\_openai\_create):

mock\_openai\_create.return\_value = {

'choices': [{'message': {'content': '/\* Improved RISC-V Pipeline Code \*/\nint main() { return 0; }'}}]

}

api\_key = 'fake\_api\_key'

prompt = 'Improve the following code.'

analysis = 'Performance can be optimized.'

current\_code = 'int main() { return 0; }'

improved\_code = improve\_code(api\_key, prompt, analysis, current\_code)

self.assertEqual(improved\_code, '/\* Improved RISC-V Pipeline Code \*/\nint main() { return 0; }')

@patch('utils.code\_improver.openai.ChatCompletion.create', side\_effect=Exception("API Error"))

def test\_improve\_code\_failure(self, mock\_openai\_create):

api\_key = 'fake\_api\_key'

prompt = 'Improve the following code.'

analysis = 'Performance can be optimized.'

current\_code = 'int main() { return 0; }'

improved\_code = improve\_code(api\_key, prompt, analysis, current\_code)

self.assertIsNone(improved\_code)

utils/code\_improver.py

import openai

def improve\_code(api\_key, prompt, analysis, current\_code):

try:

openai.api\_key = api\_key

response = openai.ChatCompletion.create(

prompt=prompt,

analysis=analysis,

current\_code=current\_code

)

improved\_code = response['choices'][0]['message']['content']

return improved\_code

except Exception as e:

print(f"Error al mejorar el código: {str(e)}")

return None

I added a try-excepting the function `improve\_code` to manage the occurring OpenAI API connections. If there's an exception, the function returns `None`. Also updated `test\_improve\_code\_failure` to verify in case of” None” error.