# Mock OpenAI API response to match expected format

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)

expected\_code = '/\* Improved RISC-V Pipeline Code \*/\nint main() { return 0; }'

self.assertEqual(improved\_code, expected\_code)

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

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

# Simulate an API failure by raising an exception

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.side\_effect = Exception("API Error")

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)

# Expecting `None` if an exception occurs

self.assertIsNone(improved\_code)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()