tests/test\_command\_interpreter.py

import unittest

from unittest.mock import patch

from utils.command\_interpreter import interpret\_command

class TestCommandInterpreter(unittest.TestCase):

@patch('utils.command\_interpreter.openai.ChatCompletion.create')

def test\_interpret\_command\_success(self, mock\_openai\_create):

mock\_openai\_create.return\_value = {

'choices': [{'message': {'content': 'make clean'}}]

}

api\_key = 'fake\_api\_key'

prompt = 'You are an assistant that converts commands.'

user\_command = 'Clean the build directory.'

system\_command = interpret\_command(api\_key, prompt, user\_command)

self.assertEqual(system\_command, 'make clean')

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

def test\_interpret\_command\_failure(self, mock\_openai\_create):

api\_key = 'fake\_api\_key'

prompt = 'You are an assistant that converts commands.'

user\_command = 'Clean the build directory.'

system\_command = interpret\_command(api\_key, prompt, user\_command)

self.assertIsNone(system\_command)

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

unittest.main()

The corrected `interpret\_command` function is:

utils/command\_interpreter.py

import openai

def interpret\_command(api\_key, prompt, user\_command):

try:

openai.api\_key = api\_key

response = openai.ChatCompletion.create(

prompt=prompt,

user\_command=user\_command

)

system\_command = response['choices'][0]['message']['content']

return system\_command

except Exception as e:

print(f"Error interpreting command: {str(e)}")

return None