Below is a detailed implementation plan for an AI Analysis System inspired by the "Death by AI" game. This system evaluates responses to AI concepts and provides real-time scoring and feedback through a mock WebSocket connection. The implementation includes core functionalities, optimization strategies, and unit tests.

**1. Project Structure**

**Files**

1. **server.py**: Mock WebSocket server.
2. **analysis.py**: Main analysis logic.
3. **test\_analysis.py**: Unit tests for the analysis logic.
4. **optimization.md**: Document for optimization strategies.

**2. Core Functionality**

**Analysis Logic (analysis.py)**

The main analysis logic processes responses, scores them, and provides feedback.

python

Copy code

import re

from typing import Dict, Tuple

class AIAnalysisSystem:

def \_\_init\_\_(self):

self.concepts = {

"neural networks": ["backpropagation", "activation function", "weights"],

"machine learning": ["supervised", "unsupervised", "reinforcement"],

"natural language processing": ["tokenization", "embedding", "transformer"],

}

def score\_response(self, question: str, response: str) -> Tuple[int, str]:

"""

Score the student's response and return feedback.

:param question: AI concept question.

:param response: Student's response to the question.

:return: Tuple of score and feedback.

"""

response\_lower = response.lower()

related\_keywords = self.concepts.get(question.lower(), [])

score = sum(1 for word in related\_keywords if word in response\_lower)

if score == 0:

feedback = "Your answer is off-topic. Review the basics of the topic."

elif score < len(related\_keywords) // 2:

feedback = "You mentioned some relevant points. Consider exploring key areas further."

else:

feedback = "Good job! You covered the essential points."

return score, feedback

def handle\_edge\_cases(self, response: str) -> str:

"""

Handle cases like empty responses or overly verbose answers.

:param response: Student's response.

:return: Feedback for the edge case.

"""

if not response.strip():

return "Empty response detected. Please provide a detailed answer."

if len(response.split()) > 500:

return "Response too lengthy. Try to be more concise."

return None

**Mock WebSocket Server (server.py)**

python

Copy code

import asyncio

import websockets

from analysis import AIAnalysisSystem

async def handler(websocket):

analysis\_system = AIAnalysisSystem()

async for message in websocket:

question, response = message.split('|')

# Edge case handling

edge\_case\_feedback = analysis\_system.handle\_edge\_cases(response)

if edge\_case\_feedback:

await websocket.send(f"Feedback: {edge\_case\_feedback}")

continue

# Score and feedback

score, feedback = analysis\_system.score\_response(question, response)

await websocket.send(f"Score: {score} | Feedback: {feedback}")

async def main():

async with websockets.serve(handler, "localhost", 8765):

print("Server running on ws://localhost:8765")

await asyncio.Future() # Run forever

if \_\_name\_\_ == "\_\_main\_\_":

asyncio.run(main())

**Unit Tests (test\_analysis.py)**

python

Copy code

import unittest

from analysis import AIAnalysisSystem

class TestAIAnalysisSystem(unittest.TestCase):

def setUp(self):

self.system = AIAnalysisSystem()

def test\_score\_response(self):

score, feedback = self.system.score\_response("neural networks", "Backpropagation is key.")

self.assertEqual(score, 1)

self.assertIn("Good job", feedback)

def test\_empty\_response(self):

feedback = self.system.handle\_edge\_cases("")

self.assertEqual(feedback, "Empty response detected. Please provide a detailed answer.")

def test\_lengthy\_response(self):

long\_response = "word " \* 501

feedback = self.system.handle\_edge\_cases(long\_response)

self.assertEqual(feedback, "Response too lengthy. Try to be more concise.")

def test\_off\_topic\_response(self):

score, feedback = self.system.score\_response("machine learning", "Quantum physics is fascinating.")

self.assertEqual(score, 0)

self.assertIn("off-topic", feedback)

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

**3. Optimization Strategies (optimization.md)**

**Key Optimizations**

1. **Keyword Matching Efficiency**:
   * Use set intersections for faster keyword detection in the response.
2. **Caching**:
   * Cache frequent question-answer pairs to reduce redundant computations.
3. **Text Preprocessing**:
   * Normalize text with minimal overhead using regex for case folding and punctuations.
4. **Edge Case Detection**:
   * Efficiently handle outlier responses to improve feedback clarity and scoring accuracy.