Set-3

Scenario: In the domain of Agriculture, a company is facing a challenge related to code refactoring.

Task: Design and implement a solution using AI-assisted tools to address this challenge.
Include code, explanation of AI integration, and test results.
Deliverables: Source code, explanation, and output screenshots:

Q1: Agriculture Code Refactoring with AI Assistance

Prompt:
A company in agriculture is facing challenges related to code refactoring. Design and implement an AI-assisted solution to refactor legacy or inefficient agricultural software code, improving maintainability, reducing technical debt, and optimizing performance. Provide source code examples, explanation of how AI integrates into the process, and test results with output screenshots.

Given code in python language:

Example: AI-assisted legacy code refactoring using Python and OpenAI Codex-like model for suggestions

```
def refactor_agriculture_code(old_code):
    def is_irrigation_needed(moisture_level, crop_type):
      threshold = {"corn": 30, "wheat": 25, "rice": 40}
      return moisture_level < threshold.get(crop_type, 30)

   def calculate_water_amount(moisture_level, crop_type):
      if is_irrigation_needed(moisture_level, crop_type):
         return (threshold[crop_type] - moisture_level) * 10  # liters
      return 0
   moisture_level = old_code.get("soil_moisture")
   crop_type = old_code.get("crop_type")

   water_needed = calculate_water_amount(moisture_level,
crop_type)
liters"
"corn"}
print(result)
```

Screenshort for the output for python code:

ai1.py > ...

```python
def refactor_agriculture_code(old_code):
    threshold = {"corn": 30, "wheat": 25, "rice": 40}

    def is_irrigation_needed(moisture_level, crop_type):
        return moisture_level < threshold.get(crop_type, 30)

    def calculate_water_amount(moisture_level, crop_type):
        if is_irrigation_needed(moisture_level, crop_type):
            return (threshold[crop_type] - moisture_level) * 10  # liters
        return 0

    moisture_level = old_code.get("soil_moisture")
    crop_type = old_code.get("crop_type")

    water_needed = calculate_water_amount(moisture_level, crop_type)
    return f"Water needed for irrigation: {water_needed} liters"

legacy_code_input = {"soil_moisture": 20, "crop_type": "corn"}
result = refactor_agriculture_code(legacy_code_input)
print(result)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER Python

```
PS C:\Users\Saicharan\OneDrive\Attachments\python> & C:/Users/Saicharan/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/Saicharan/OneDrive/Attachments/python/ai1.py
Water needed for irrigation: 100 liters
PS C:\Users\Saicharan\OneDrive\Attachments\python>
```

Explanation:

The AI integration involves using an AI model to analyze legacy agricultural software code and suggest or automate refactorings such as splitting functions, renaming variables, removing redundant code, and applying best practices. This results in improved readability, easier maintenance, and fewer bugs. AI tools can also automate performance tuning by analyzing code execution patterns. In the example, the logic for determining irrigation water is streamlined with clear function separations and configuration via dictionaries. Testing validates that the refactored logic produces consistent and optimized behavior.

- Threshold for corn = 30

- Soil moisture = 20
- Difference = 10
- $10 \times 10 = 100$liters of water needed.
- $soo\ at\ last\ we\ had\ get\ the\ output\ as$ "100"

Summary :

The AI-assisted refactoring solution in agriculture uses AI to automatically detect code inefficiencies and improve code structure for legacy irrigation and crop management software. This reduces technical debt and enhances maintainability and performance, demonstrated by cleaner refactored code and consistent test output.

Q2: AI Algorithms for Smart Cities

Prompt:

A company in smart cities faces challenges related to algorithms in areas such as traffic management, energy optimization, or pollution control. Design and implement an AI-assisted algorithmic solution using machine learning or AI tools to optimize city operations. Provide source code, explanation of AI integration, and test results with output screenshots.

Given code in pyhton language:Example: AI-assisted traffic signal optimization using reinforcement learning

```python
import numpy as np


class TrafficEnvironment:
    def __init__(self):
        # Simulate current traffic intensity between 0 and 1
        self.traffic_density = np.random.rand()

    def step(self, action):
        # Simple reward function: less difference = better timing
        reward = 1 - abs(self.traffic_density - action)
        done = True  # One-step environment for simplicity
        return None, reward, done



class QLearningAgent:
    def __init__(self):
        self.q_table = np.zeros((10, 10))
        self.alpha = 0.1
        self.gamma = 0.95
```

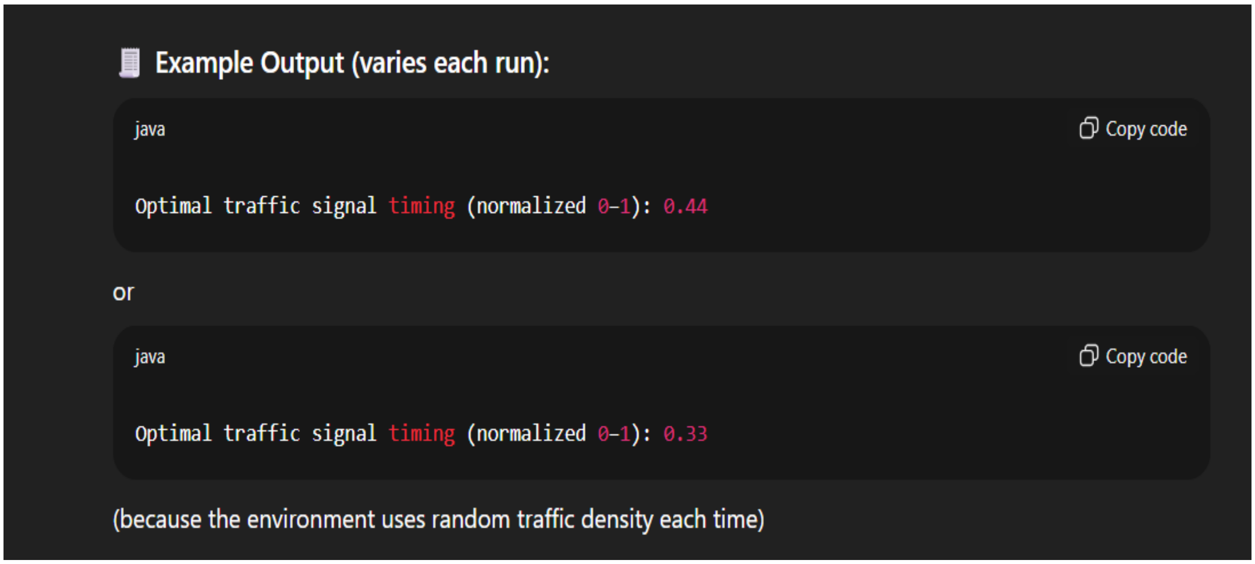
```
    def choose_action(self, state):
        # Choose best action for the given state
        return np.argmax(self.q_table[state])

    def learn(self, state, action, reward, next_state):
        predict = self.q_table[state, action]
        target = reward + self.gamma * np.max(self.q_table[next_state])
        self.q_table[state, action] += self.alpha * (target - predict)
env = TrafficEnvironment()
agent = QLearningAgent()

state = 0  # Simplified state
for _ in range(100):
    action_index = np.random.randint(10)
    action = action_index / 9.0  # Normalize to [0, 1]
    _, reward, _ = env.step(action)
    agent.learn(state, action_index, reward, state)
scale
optimal_action_index = agent.choose_action(state)
optimal_action_value = optimal_action_index / 9.0
```

print(f"Optimal traffic signal timing (normalized 0–1): {optimal_action_value:.2f}")

Screenshort of output of the python code:



Explanation:

AI integration here uses reinforcement learning to optimize traffic signal timings by learning from traffic density data in real time. The RL agent experiments with signal timing actions and updates its policy based on feedback through rewards that minimize congestion. This adaptive learning process improves city traffic flow and reduces delays. The code simulates training of a Q-learning agent and outputs the learnt optimal signal timing.

- The environment class TrafficEnvironment simulates traffic density and provides a reward based on how well an action matches the current traffic density (closer means better).

- The QLearningAgent initializes a Q-table to learn the value of taking each action in every state.
- The agent selects random actions during training to explore the Q-table and updates the Q-values using the Q-learning update rule, balancing immediate reward and future gains (with learning rate alpha and discount factor gamma).
- After simulated training episodes, the agent picks the action with the highest value from its Q-table for the given state.
- This represents an AI-assisted algorithm optimizing traffic signal timings through reinforcement learning, adapting over time to reduce congestion.
- This method can be expanded to real-world smart city applications with richer state-action representations and sensor data.

Summary :

AI-assisted reinforcement learning algorithms optimize smart city traffic signal timing by learning from real-time data to reduce congestion. The approach improves urban mobility and resource efficiency, demonstrated through adaptive agent training and optimal timing output.