DRL_GROUP_244_MAB_Clinical_Trial_Assignment01

June 21, 2025

0.0.1 ---------------Mandatory Information to fill------------

0.0.2 Group ID: 244

1 Group members

Name

Email

Student ID

Contribution

G. Ankur Vatsa

2023aa05727@wilp.bits-pilani.ac.in

2023aa05727

100%

DURGA PRASAD YADAV

2024ab05147@wilp.bits-pilani.ac.in

2024ab05147

100%

JAIDEEP PALIT

2024aa05319@wilp.bits-pilani.ac.in

2024aa05319

100%

FIYAS AHAMED A

2023aa05796@wilp.bits-pilani.ac.in

2023aa05796

100%

2 Scenario

A pharmaceutical company is conducting clinical trials to evaluate the effectiveness of three antiretroviral drug combinations for treating HIV-positive patients. Due to the ethical and cost constraints of clinical trials, it is critical to identify the most effective treatment regimen using the least number of patients. Each treatment (or "arm") can lead to different outcomes depending on patient responses. The effectiveness of each treatment is evaluated using a reward function derived from the improvement in patients' immune system markers and survival status.

3 Problem Definition

You are provided with a clinical dataset where each record corresponds to a patient, including the treatment they received and the resulting health outcomes. Your task is to simulate a clinical trial environment using various MAB strategies to sequentially recommend treatments and observe outcomes. The objective is to maximize the overall success rate across trials by identifying and favouring the most effective treatment.

4 Dataset

You will be provided a dataset containing the following fields:

- ***Age (age)***: Patient's age in years at baseline.
- ***Weight (wtkg)***: Continuous feature representing weight in kilograms at baseline.
- ***Gender (gender)***: Binary indicator of gender (0 = Female, 1 = Male).
- ***CD4 Counts (cd40, cd420)***: Integer values representing CD4 counts at baseline and 20+/-5 weeks.
- ***Treatment Indicator (trt)***: Categorical feature indicating the type of treatment received (0 = ZDV only, 1 = ZDV + ddI, 2 = ZDV + Zal, 3 = ddI only).
- ***Censoring Indicator (label)***: Binary indicator (1 = failure, 0 = censoring) denoting patient status.

Link for accessing dataset: https://drive.google.com/file/d/1LYfIrJ4VEEGeyOsSt_qoLk7FaAv5Jfx-/view?usp=sharing

5 Environment Setup

Arms (Actions): The treatment types (trt) * Arm 0: ZDV only * Arm 1: ZDV + ddI * Arm 2: ZDV + Zal * Arm 3: ddI only

Reward Function:

Reward `r` is defined as:

`r = 1, if (label == 0) and (cd420 > cd40)`

`r = 0, otherwise`

This reward represents a successful treatment outcome as an increase in CD4 count and survival.

Assumptions:

Number of Iterations: Run the simulation for at least 1000 trials (iterations), with the option to extend the number of trials depending on the convergence behavior or observed reward trends. In each iteration, simulate one patient trial using one of the bandit policies.

6 Requirements and Deliverables:

Implement the Multi-Arm Bandit Problem for the given above scenario for all the below mentioned policy methods.

6.0.1 Initialize constants

[1]:
```
!pip install pandas numpy matplotlib seaborn
```

```
Requirement already satisfied: pandas in ./.venv/lib/python3.9/site-packages
(2.3.0)
Requirement already satisfied: numpy in ./.venv/lib/python3.9/site-packages
(2.0.2)
Requirement already satisfied: matplotlib in ./.venv/lib/python3.9/site-packages
(3.9.4)
Requirement already satisfied: seaborn in ./.venv/lib/python3.9/site-packages
(0.13.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
./.venv/lib/python3.9/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in ./.venv/lib/python3.9/site-
packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in ./.venv/lib/python3.9/site-
packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in ./.venv/lib/python3.9/site-
packages (from matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in ./.venv/lib/python3.9/site-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in ./.venv/lib/python3.9/site-
packages (from matplotlib) (4.58.4)
Requirement already satisfied: kiwisolver>=1.3.1 in ./.venv/lib/python3.9/site-
packages (from matplotlib) (1.4.7)
Requirement already satisfied: packaging>=20.0 in ./.venv/lib/python3.9/site-
packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in ./.venv/lib/python3.9/site-packages
(from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in ./.venv/lib/python3.9/site-
packages (from matplotlib) (3.2.3)
Requirement already satisfied: importlib-resources>=3.2.0 in
./.venv/lib/python3.9/site-packages (from matplotlib) (6.5.2)
Requirement already satisfied: zipp>=3.1.0 in ./.venv/lib/python3.9/site-
packages (from importlib-resources>=3.2.0->matplotlib) (3.23.0)
Requirement already satisfied: six>=1.5 in ./.venv/lib/python3.9/site-packages
(from python-dateutil>=2.8.2->pandas) (1.17.0)
```

[2]:
```python
# Constants
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from typing import List, Tuple, Dict
import random
from abc import ABC, abstractmethod

# Set random seeds for reproducibility
np.random.seed(42)
random.seed(42)
```

7 Load Dataset (0.5M)

[3]:
```python
# Dataset is loaded in __init__( ) method of ClinicalTrialEnv class
```

8 Design a Clinical Trial Environment (0.5M)

[4]:
```python
# Code for environment setup along with reward function
#-----write your code below this line---------

class ClinicalTrialEnvironment:
    """
    BANDIT ENVIRONMENT FOR CLINICAL TRIALS
    ======================================

    This class represents the "environment" in our Multi-Armed Bandit problem.
    In MAB terminology:
    - ENVIRONMENT: The external system that provides rewards when we take␣
↪actions
    - ARMS: The available actions (treatment options)
    - REWARDS: The outcomes we receive (treatment success/failure)

    CLINICAL TRIAL CONTEXT:
    ----------------------
    - We have historical data about patients who received different treatments
    - Each treatment is an "arm" of our bandit
    - When we "pull an arm", we simulate giving that treatment to a patient
    - The reward reflects treatment success based on CD4 count improvement and␣
↪survival

    REWARD FUNCTION EXPLANATION:
    ---------------------------
```

```
    Reward = 1 if (patient survived AND CD4 count increased), else 0

    This binary reward captures two key outcomes:
    1. SURVIVAL: Patient didn't experience treatment failure (label = 0)
    2. IMMUNE RESPONSE: CD4 count improved from baseline to week 20 (cd420 >␣
↪cd40)

    CD4 cells are crucial immune system components that HIV attacks. An increase
    in CD4 count indicates the treatment is helping restore immune function.
    """

    def __init__(self, data_path: str = 'Clinical_Trial.csv'):
        """
        Initialize the clinical trial environment.

        STEP 1: ENVIRONMENT SETUP
        ------------------------
        Load the clinical dataset and organize it by treatment arms.
        This allows us to simulate realistic patient outcomes for each␣
↪treatment.

        Args:
            data_path: Path to the clinical trial dataset
        """
        print("INITIALIZING CLINICAL TRIAL ENVIRONMENT")
        print("=" * 50)

        # Load the clinical trial dataset
        self.data = pd.read_csv(data_path)
        print(f"Loaded dataset with {len(self.data)} patient records")

        # Define the treatment arms (actions in MAB terminology)
        self.n_arms = 4  # 4 treatment types (0, 1, 2, 3)
        self.arm_names = [
            'ZDV only',       # Arm 0: Monotherapy with Zidovudine
            'ZDV + ddI',      # Arm 1: Combination with Didanosine
            'ZDV + Zal',      # Arm 2: Combination with Zalcitabine
            'ddI only'        # Arm 3: Monotherapy with Didanosine
        ]

        print(f"Number of treatment arms: {self.n_arms}")
        for i, name in enumerate(self.arm_names):
            print(f"  Arm {i}: {name}")

        # Precompute rewards for each treatment arm for efficient simulation
        self._precompute_arm_data()
```

```
    def _precompute_arm_data(self):
        """
        REWARD COMPUTATION AND ARM ORGANIZATION
        ======================================

        This method processes the historical data to:
        1. Group patients by treatment arm
        2. Calculate binary rewards based on the defined reward function
        3. Store patient pools for each arm to enable realistic sampling

        REWARD FUNCTION IMPLEMENTATION:
        ------------------------------
        For each patient, reward = 1 if:
        - label == 0 (patient survived/didn't fail treatment) AND
        - cd420 > cd40 (CD4 count increased from baseline to week 20)

        This creates a binary outcome that captures treatment success.
        """
        print("\nCOMPUTING REWARDS FOR EACH TREATMENT ARM")
        print("-" * 45)

        self.arm_data = {}      # Store patient data for each arm
        self.arm_rewards = {}   # Store precomputed rewards for each arm

        for arm in range(self.n_arms):
            # Filter patients who received this specific treatment
            arm_patients = self.data[self.data['trt'] == arm].copy()

            # Apply the reward function: success = survival AND CD4 improvement
            # This boolean operation creates a binary reward (True/False -> 1/0)
            arm_patients['reward'] = (
                (arm_patients['label'] == 0) &      # Patient survived (no␣
↪treatment failure)
                (arm_patients['cd420'] > arm_patients['cd40'])  # CD4 count␣
↪improved
            ).astype(int)  # Convert boolean to integer (True->1, False->0)

            # Store the processed data
            self.arm_data[arm] = arm_patients
            self.arm_rewards[arm] = arm_patients['reward'].tolist()

        # Print statistics for each treatment arm
        print("ARM STATISTICS (Historical Data Analysis):")
        for arm in range(self.n_arms):
            n_patients = len(self.arm_data[arm])
            if self.arm_rewards[arm]:
                success_rate = np.mean(self.arm_rewards[arm])
```

```
                n_successes = sum(self.arm_rewards[arm])
            else:
                success_rate = 0
                n_successes = 0

            print(f"  {self.arm_names[arm]:12} | Patients: {n_patients:3d} | "
                  f"Successes: {n_successes:3d} | Success Rate: {success_rate:.
↪3f}")

        print(f"\nTotal patients in dataset: {len(self.data)}")

    def get_reward(self, arm: int) -> Tuple[int, Dict]:
        """
        SIMULATE TREATMENT ADMINISTRATION
        ================================

        This method simulates "pulling an arm" in MAB terminology.
        When we select a treatment (arm), we simulate giving that treatment
        to a patient and observe the outcome (reward).

        SIMULATION APPROACH:
        -------------------
        Instead of generating synthetic outcomes, we sample from historical
        patient data. This ensures realistic reward distributions based on
        actual clinical trial results.

        PROCESS:
        1. Validate the selected arm
        2. Randomly sample a patient from that treatment group
        3. Return the patient's reward (treatment outcome)
        4. Provide patient information for analysis

        Args:
            arm: Treatment arm to pull (0-3)

        Returns:
            Tuple of (reward, patient_info)
            - reward: Binary outcome (1=success, 0=failure)
            - patient_info: Dictionary with patient demographics and outcomes
        """
        # Validate arm selection
        if arm not in range(self.n_arms):
            raise ValueError(f"Invalid arm {arm}. Must be in range 0-{self.
↪n_arms-1}")

        # Handle edge case: no patients in this treatment arm
        if not self.arm_rewards[arm]:
```

```python
            print(f"Warning: No patients available for {self.arm_names[arm]}")
            return 0, {}

        # SAMPLING STRATEGY: Random patient selection
        # This simulates the uncertainty of not knowing which specific patient
        # will receive the treatment next
        reward = np.random.choice(self.arm_rewards[arm])
        patient_idx = np.random.randint(len(self.arm_data[arm]))
        patient_info = self.arm_data[arm].iloc[patient_idx].to_dict()

        return reward, patient_info

    def get_true_rewards(self) -> List[float]:
        """
        ORACLE INFORMATION: TRUE EXPECTED REWARDS
        ========================================

        In real-world scenarios, we don't know the true reward probabilities.
        However, for analysis purposes, we can calculate them from historical␣
 ↪data.

        This represents the "oracle" or "ground truth" that our MAB algorithms
        are trying to discover through exploration and exploitation.

        Returns:
            List of true expected rewards for each arm
        """
        return [np.mean(self.arm_rewards[arm]) if self.arm_rewards[arm] else 0
                for arm in range(self.n_arms)]
```

[5]:

```python
class BanditPolicy(ABC):
    """
    ABSTRACT BASE CLASS FOR MULTI-ARMED BANDIT POLICIES
    ===================================================

    This class defines the interface that all MAB algorithms must implement.
    It provides common functionality for tracking rewards and maintaining state.

    CORE MAB ALGORITHM STRUCTURE:
    ----------------------------
    All MAB algorithms follow this general pattern:
    1. INITIALIZATION: Set up counters and reward trackers
    2. ARM SELECTION: Choose which treatment to try next (algorithm-specific)
    3. REWARD OBSERVATION: Receive outcome from the environment
    4. STATE UPDATE: Update internal statistics based on the observed reward
    5. REPEAT: Continue until stopping criterion is met
```

```
    KEY METRICS TRACKED:
    -------------------
    - counts: How many times each arm has been selected
    - rewards: Total reward received from each arm
    - total_reward: Overall cumulative reward
    - t: Current time step (iteration number)

    ABSTRACT METHOD:
    ---------------
    select_arm(): Each policy must implement its own arm selection strategy
    """

    def __init__(self, n_arms: int):
        """
        Initialize the bandit policy.

        Args:
            n_arms: Number of treatment arms available
        """
        self.n_arms = n_arms
        self.reset()

    def reset(self):
        """
        POLICY STATE INITIALIZATION/RESET
        ================================

        Reset all tracking variables to initial state.
        This is crucial for:
        1. Starting fresh experiments
        2. Comparing different policies fairly
        3. Running multiple independent simulations

        STATE VARIABLES:
        ---------------
        - counts: Array tracking selections per arm (exploration measure)
        - rewards: Array tracking total rewards per arm (for computing averages)
        - total_reward: Overall performance metric
        - t: Time step counter (important for time-dependent algorithms like␣
↪UCB)
        """
        self.counts = np.zeros(self.n_arms)      # Selection count for each arm
        self.rewards = np.zeros(self.n_arms)     # Total reward sum for each arm
        self.total_reward = 0                    # Cumulative reward across all␣
↪arms
        self.t = 0                               # Current time step
```

```
    @abstractmethod
    def select_arm(self) -> int:
        """
        ARM SELECTION STRATEGY (ABSTRACT METHOD)
        =======================================

        This is the core of each MAB algorithm. Different policies implement
        different strategies for balancing exploration vs exploitation:

        - RANDOM: Pure exploration (ignore learned information)
        - GREEDY: Pure exploitation (always choose apparent best)
        -  -GREEDY: Mostly exploit with occasional exploration
        - UCB: Sophisticated balance using confidence bounds

        Each policy must implement this method with its specific logic.

        Returns:
            Selected arm index (0 to n_arms-1)
        """
        pass

    def update(self, arm: int, reward: float) -> None:
        """
        STATE UPDATE AFTER REWARD OBSERVATION
        ====================================

        This method updates the policy's internal state after receiving
        a reward from the selected arm. This is where "learning" happens.

        UPDATE PROCESS:
        --------------
        1. Increment selection count for the chosen arm
        2. Add received reward to arm's total reward
        3. Update overall cumulative reward
        4. Advance time step counter

        These updates enable the policy to:
        - Compute average rewards per arm
        - Track exploration levels
        - Make informed future decisions

        Args:
            arm: The arm that was selected
            reward: The reward received from that arm
        """
        self.counts[arm] += 1              # Track how often this arm was chosen
        self.rewards[arm] += reward        # Accumulate rewards for this arm
```

```
        self.total_reward += reward         # Track overall performance
        self.t += 1                         # Advance time step

    def get_average_rewards(self) -> np.ndarray:
        """
        COMPUTE EMPIRICAL REWARD ESTIMATES
        =================================

        Calculate the average reward for each arm based on observed outcomes.
        This is the core statistic that most MAB algorithms use for decision␣
↪making.

        MATHEMATICAL FORMULATION:
        ------------------------
        For arm i: average_reward[i] = total signalrewards[i] /␣
↪selection_counts[i]

        SPECIAL HANDLING:
        ----------------
        If an arm hasn't been selected (count = 0), its average is set to 0
        to avoid division by zero errors.

        Returns:
            Array of average rewards for each arm
        """
        return np.divide(self.rewards, self.counts, out=np.zeros_like(self.
↪rewards), where=self.counts!=0)
```

9 Using Random Policy (0.5M)

Implement a random policy for treatment selection and print each 100th iteration. (Mandatory)

[6]:
```
#  run the environment with an agent that is guided by a random policy
#-----write your code below this line---------
class RandomPolicy(BanditPolicy):
    """
    RANDOM POLICY: PURE EXPLORATION STRATEGY
    ========================================

    ALGORITHM DESCRIPTION:
    ---------------------
    The random policy selects arms uniformly at random, completely ignoring
    any learned information about arm performance. This represents pure␣
↪exploration
    with no exploitation.

```

```
    CHARACTERISTICS:
    ---------------
    - EXPLORATION: Maximum (always trying different options)
    - EXPLOITATION: None (never uses learned information)
    - CONVERGENCE: Never converges to optimal arm
    - USE CASE: Baseline comparison, when no prior knowledge exists

    MATHEMATICAL FORMULATION:
    ------------------------
    P(select arm i) = 1/n_arms for all arms i

    CLINICAL TRIAL CONTEXT:
    ----------------------
    This would be equivalent to randomly assigning treatments to patients
    without considering which treatments have shown better outcomes.
    While ethically questionable in practice, it serves as a useful baseline
    for comparison with more sophisticated policies.

    EXPECTED PERFORMANCE:
    --------------------
    - Average reward will converge to the mean reward across all arms
    - Will not identify the best treatment effectively
    - Useful for establishing lower bound on performance
    """

    def select_arm(self) -> int:
        """
        RANDOM ARM SELECTION
        ===================

        Simply return a random arm index. No learning or optimization involved.

        Returns:
            Randomly selected arm index
        """
        return np.random.randint(self.n_arms)
```

10 Using Greedy Policy (1M)

Implement the Greedy policy that always selects the treatment with the highest average reward and print each 100th iteration. (Mandatory)

[7]:
```
#  run the environment with an agent that is guided by a greedy policy
#-----write your code below this line---------
class GreedyPolicy(BanditPolicy):
    """
    GREEDY POLICY: PURE EXPLOITATION STRATEGY
```

```
    ==========================================

    ALGORITHM DESCRIPTION:
    ----------------------
    The greedy policy always selects the arm with the highest observed average␣
↪reward.
    After initial exploration of each arm once, it commits to exploiting the
    apparent best option.

    CHARACTERISTICS:
    ---------------
    - EXPLORATION: Minimal (only tries each arm once initially)
    - EXPLOITATION: Maximum (always chooses apparent best)
    - CONVERGENCE: Fast to a fixed choice, but may be suboptimal
    - RISK: May get stuck on suboptimal arms due to early lucky outcomes

    MATHEMATICAL FORMULATION:
    ------------------------
    select arm i* = argmax_i(average_reward[i])

    CLINICAL TRIAL CONTEXT:
    ----------------------
    This approach would quickly identify a preferred treatment based on
    early results and then consistently use that treatment. While efficient
    if the early assessment is correct, it risks missing better treatments
    that might have had unlucky early results.

    COLD START PROBLEM:
    ------------------
    Initially, no arms have been tried, so we must explore each arm at least
    once before we can make informed greedy decisions.

    ADVANTAGES:
    ----------
    - Simple to understand and implement
    - Low computational overhead
    - Good when true best arm is clearly superior

    DISADVANTAGES:
    -------------
    - Susceptible to initial sampling noise
    - Cannot recover from early mistakes
    - May miss better arms that had unlucky starts
    """

    def select_arm(self) -> int:
        """
```

```
        GREEDY ARM SELECTION WITH INITIAL EXPLORATION
        ============================================

        SELECTION LOGIC:
        ---------------
        1. If we haven't tried all arms yet, explore the next untried arm
        2. Otherwise, select the arm with the highest average reward

        INITIAL EXPLORATION PHASE:
        -------------------------
        We must try each arm at least once before we can make informed
        greedy choices. This prevents immediate commitment to arm 0.

        EXPLOITATION PHASE:
        ------------------
        After trying all arms, always choose the one with best average reward.
        No further exploration occurs.

        Returns:
            Selected arm index
        """
        if self.t < self.n_arms:
            # EXPLORATION PHASE: Try each arm once
            # This ensures we have some information about each treatment
            # before committing to pure exploitation
            return self.t

        # EXPLOITATION PHASE: Choose the arm with highest average reward
        avg_rewards = self.get_average_rewards()
        return np.argmax(avg_rewards)
```

11 Using Epsilon-Greedy Policy (1.5M)

Implement the -Greedy policy with = 0.1, 0.2, 0.5. Report iteration-wise selections and rewards. Determine which yields the best result. (Mandatory)

[8]:
```
#  run the environment with an agent that is guided by a epsilon-greedy policy
#-----write your code below this line---------
class EpsilonGreedyPolicy(BanditPolicy):
    """
     -GREEDY POLICY: BALANCED EXPLORATION-EXPLOITATION
    =================================================

    ALGORITHM DESCRIPTION:
    ---------------------
    The  -greedy policy provides a simple but effective balance between
    exploration and exploitation. With probability  , it explores randomly;
```

```
    with probability (1-ε), it exploits the current best arm.

    CHARACTERISTICS:
    ---------------
    - EXPLORATION: Controlled by parameter  (epsilon)
    - EXPLOITATION: Occurs (1- ) fraction of the time
    - CONVERGENCE: Can continue discovering better arms throughout execution
    - TUNING: Performance heavily depends on choice of

    MATHEMATICAL FORMULATION:
    ------------------------
    With probability  : select random arm
    With probability (1- ): select argmax_i(average_reward[i])

    EPSILON PARAMETER ANALYSIS:
    --------------------------
    -   = 0.0: Reduces to pure greedy (no exploration after initial phase)
    -   = 1.0: Reduces to pure random (no exploitation)
    -   = 0.1: Explores 10% of time, exploits 90% (common choice)
    -   = 0.2: More exploration, slower convergence but better long-term
↪discovery
    -   = 0.5: High exploration, may sacrifice short-term performance

    CLINICAL TRIAL CONTEXT:
    ----------------------
    This policy would mostly use the best-performing treatment but occasionally
    try other treatments to ensure we don't miss improvements or changing
    effectiveness over time.

    ADVANTAGES:
    ----------
    - Simple and intuitive
    - Provides ongoing exploration
    - Can recover from early mistakes
    - Works well across many problem types

    DISADVANTAGES:
    -------------
    - Exploration rate is fixed (doesn't adapt to confidence)
    - May waste time exploring obviously poor arms
    - Parameter   requires tuning
    """

    def __init__(self, n_arms: int, epsilon: float = 0.1):
        """
        Initialize  -greedy policy.
```

```python
        Args:
            n_arms: Number of treatment arms
            epsilon: Exploration probability (0      1)
        """
        super().__init__(n_arms)
        self.epsilon = epsilon

        # Validate epsilon parameter
        if not 0 <= epsilon <= 1:
            raise ValueError(f"Epsilon must be between 0 and 1, got {epsilon}")

    def select_arm(self) -> int:
        """
         -GREEDY ARM SELECTION
        =====================

        DECISION PROCESS:
        ----------------
        1. Generate random number between 0 and 1
        2. If random number <  : EXPLORE (select random arm)
        3. If random number    : EXPLOIT (select best known arm)

        INITIAL EXPLORATION:
        -------------------
        Like the greedy policy, we ensure each arm is tried at least once
        before making exploitation decisions.

        EXPLORATION vs EXPLOITATION TRADE-OFF:
        -------------------------------------
        - Higher  : More exploration, slower convergence, better long-term discovery
        - Lower  : Less exploration, faster convergence, risk of missing better arms

        Returns:
            Selected arm index
        """
        if self.t < self.n_arms:
            # INITIAL EXPLORATION: Try each arm once
            # This provides baseline information for all treatments
            return self.t

        # Generate random decision
        if np.random.random() < self.epsilon:
            # EXPLORATION: Random arm selection
            # This maintains the ability to discover better treatments
            return np.random.randint(self.n_arms)
```

```
        else:
            # EXPLOITATION: Choose current best arm
            # This leverages learned knowledge to maximize immediate reward
            avg_rewards = self.get_average_rewards()
            return np.argmax(avg_rewards)
```

12 Using UCB (1M)

Implement the UCB algorithm for treatment selection and print each 100th iteration. (Mandatory)

[9]:
```
#  run the environment with an agent that is guided by a UCB
#-----write your code below this line---------
class UCBPolicy(BanditPolicy):
    """
    UPPER CONFIDENCE BOUND (UCB) POLICY: OPTIMISTIC EXPLORATION
    ==========================================================

    ALGORITHM DESCRIPTION:
    ---------------------
    UCB addresses a key limitation of  -greedy: it explores intelligently rather
    than randomly. It uses statistical confidence bounds to balance exploration
    and exploitation optimally.

    CORE PRINCIPLE: "OPTIMISM IN THE FACE OF UNCERTAINTY"
    ----------------------------------------------------
    For each arm, UCB computes an upper confidence bound on the true reward.
    It then selects the arm with the highest upper bound, naturally balancing:
    - Arms with high average rewards (exploitation)
    - Arms with high uncertainty (exploration)

    MATHEMATICAL FORMULATION:
    ------------------------
    UCB_i(t) = average_reward[i] + c * sqrt(ln(t) / n_i)

    Where:
    - average_reward[i]: Empirical mean reward for arm i
    - c: Confidence parameter (typically 1.0)
    - t: Total number of rounds played
    - n_i: Number of times arm i has been selected

    INTUITION BEHIND THE FORMULA:
    ----------------------------
    1. EXPLOITATION TERM (average_reward[i]):
       - Favors arms with high observed rewards

    2. EXPLORATION TERM (c * sqrt(ln(t) / n_i)):
       - sqrt(ln(t)): Grows with time (more total experience = wider bounds)
```

```
      - 1/sqrt(n_i): Shrinks with arm selections (more arm experience = tighter bounds)
      - c: Controls exploration aggressiveness

   ADAPTIVE EXPLORATION:
   ---------------------
   Unlike  -greedy's fixed exploration rate, UCB adapts exploration based on:
   - How uncertain we are about each arm (fewer selections = more exploration)
   - How much total experience we have (more rounds = wider confidence intervals)

   CLINICAL TRIAL CONTEXT:
   -----------------------
   UCB would prioritize treatments that either:
   1. Have shown good results (high average reward)
   2. Haven't been tried much (high uncertainty/exploration bonus)

   This naturally implements the medical principle of trying promising treatments
   while ensuring adequate investigation of undertested options.

   THEORETICAL PROPERTIES:
   -----------------------
   - REGRET BOUND: UCB has proven logarithmic regret bounds (optimal)
   - CONVERGENCE: Eventually converges to optimal arm
   - ADAPTATION: Automatically balances exploration/exploitation

   ADVANTAGES:
   -----------
   - Theoretically optimal regret bounds
   - No parameter tuning required (c=1 works well)
   - Intelligent, adaptive exploration
   - Handles multi-armed bandit assumptions well

   DISADVANTAGES:
   --------------
   - More complex than  -greedy
   - Assumes rewards are bounded and stationary
   - May be overly optimistic in some scenarios
   """

   def __init__(self, n_arms: int, c: float = 1.0):
       """
       Initialize UCB policy.

       Args:
           n_arms: Number of treatment arms
```

```
            c: Confidence parameter (controls exploration level)
        """
        super().__init__(n_arms)
        self.c = c

        # Validate confidence parameter
        if c <= 0:
            raise ValueError(f"Confidence parameter c must be positive, got␣
↪{c}")

    def select_arm(self) -> int:
        """
        UCB ARM SELECTION WITH CONFIDENCE BOUNDS
        ========================================

        SELECTION ALGORITHM:
        -------------------
        1. For each arm, compute upper confidence bound
        2. Select arm with highest upper confidence bound
        3. Handle initial exploration phase

        UPPER CONFIDENCE BOUND CALCULATION:
        ----------------------------------
        UCB_i = average_reward[i] + c * sqrt(ln(t) / count[i])

        COMPONENTS EXPLANATION:
        ----------------------
        - average_reward[i]: What we know about arm i's performance
        - sqrt(ln(t) / count[i]): How uncertain we are about arm i
        - c: How optimistic we want to be about uncertain arms

        INITIAL EXPLORATION:
        -------------------
        Like other policies, we must try each arm once before computing
        meaningful confidence bounds.

        Returns:
            Selected arm index (highest UCB value)
        """
        if self.t < self.n_arms:
            # INITIAL EXPLORATION: Try each arm once
            # We need at least one sample per arm to compute confidence bounds
            return self.t

        # COMPUTE UCB VALUES FOR ALL ARMS
        avg_rewards = self.get_average_rewards()
```

```
        # Calculate confidence bounds: c * sqrt(ln(t) / n_i)
        # ln(self.t): Grows slowly with total experience
        # 1/self.counts: Larger for less-explored arms
        confidence_bounds = self.c * np.sqrt(np.log(self.t) / self.counts)

        # UCB = empirical mean + confidence bound
        ucb_values = avg_rewards + confidence_bounds

        # SELECT ARM WITH HIGHEST UPPER CONFIDENCE BOUND
        # This automatically balances:
        # - High-performing arms (high avg_rewards)
        # - Uncertain arms (high confidence_bounds)
        return np.argmax(ucb_values)
```

[10]:
```
class BanditSimulator:
    """
    MULTI-ARMED BANDIT EXPERIMENT SIMULATOR
    ======================================

    This class orchestrates the execution and comparison of different MAB␣
↪policies
    in the clinical trial environment. It provides a framework for:

    1. Running individual policy experiments
    2. Comparing multiple policies
    3. Collecting comprehensive performance metrics
    4. Generating detailed analysis reports

    KEY RESPONSIBILITIES:
    -------------------
    - EXPERIMENT EXECUTION: Run MAB algorithms for specified iterations
    - DATA COLLECTION: Track all relevant metrics during experiments
    - POLICY COMPARISON: Enable fair comparison across different algorithms
    - RESULT ANALYSIS: Compute and store performance statistics

    METRICS TRACKED:
    ---------------
    - Arm selections over time (exploration patterns)
    - Rewards received at each iteration (immediate performance)
    - Cumulative rewards (overall performance)
    - Average reward estimates evolution (learning progress)
    - Final performance statistics (summary metrics)

    This comprehensive tracking enables deep analysis of algorithm behavior
    and performance characteristics in the clinical trial context.
    """
```

```
    def __init__(self, environment: ClinicalTrialEnvironment):
        """
        Initialize the bandit simulator.

        Args:
            environment: The clinical trial environment to use for experiments
        """
        self.env = environment      # The environment that provides rewards
        self.results = {}          # Storage for experiment results

    def run_experiment(self, policy: BanditPolicy, n_iterations: int = 1000,
                      verbose: bool = False) -> Dict:
        """
        EXECUTE A SINGLE MAB POLICY EXPERIMENT
        ======================================

        This method runs a complete MAB experiment with the specified policy,
        implementing the core MAB algorithm loop:

        MAB ALGORITHM LOOP:
        ------------------
        For each iteration t = 1, 2, ..., T:
        1. POLICY DECISION: Policy selects an arm based on current knowledge
        2. ENVIRONMENT INTERACTION: Environment provides reward for selected arm
        3. POLICY UPDATE: Policy updates its internal state with new information
        4. METRIC TRACKING: Record all relevant data for analysis

        COMPREHENSIVE DATA COLLECTION:
        ------------------------------
        The simulator tracks multiple types of data:
        - DECISION SEQUENCE: Which arms were selected over time
        - REWARD SEQUENCE: What rewards were received
        - CUMULATIVE PERFORMANCE: Running total of rewards
        - LEARNING PROGRESS: Evolution of reward estimates
        - SELECTION PATTERNS: How often each arm was chosen

        CLINICAL TRIAL SIMULATION:
        --------------------------
        Each iteration represents treating one patient:
        1. Policy recommends a treatment (arm selection)
        2. Patient receives treatment and outcome is observed (reward)
        3. Policy learns from the outcome (state update)
        4. Process repeats with next patient

        Args:
            policy: The bandit policy to evaluate
            n_iterations: Number of patients/iterations to simulate
```

```
            verbose: Whether to print detailed iteration information

        Returns:
            Dictionary containing comprehensive experiment results
        """
        # EXPERIMENT INITIALIZATION
        policy.reset()  # Ensure clean starting state

        # DATA COLLECTION STRUCTURES
        arm_selections = []           # Track which arm was selected each␣
↪iteration
        rewards_received = []         # Track reward received each iteration
        cumulative_rewards = []       # Track running total of rewards
        average_rewards_history = []  # Track evolution of reward estimates

        # MAIN EXPERIMENT LOOP
        print(f"Running {policy.__class__.__name__} for {n_iterations}␣
↪iterations...")

        for t in range(n_iterations):
            # STEP 1: POLICY MAKES TREATMENT RECOMMENDATION
            arm = policy.select_arm()

            # STEP 2: SIMULATE TREATMENT ADMINISTRATION AND OUTCOME
            reward, patient_info = self.env.get_reward(arm)

            # STEP 3: POLICY LEARNS FROM OUTCOME
            policy.update(arm, reward)

            # STEP 4: RECORD DATA FOR ANALYSIS
            arm_selections.append(arm)
            rewards_received.append(reward)
            cumulative_rewards.append(policy.total_reward)
            average_rewards_history.append(policy.get_average_rewards().copy())

            # OPTIONAL: DETAILED PROGRESS REPORTING
            if verbose and (t < 20 or t % 100 == 0):
                print(f"  Iteration {t+1:4d}: Selected {self.env.arm_names[arm]:
↪12} "
                      f"(Arm {arm}), Reward: {reward}, Cumulative: {policy.
↪total_reward:4d}")

        # POST-EXPERIMENT ANALYSIS
        # Calculate final statistics for comprehensive evaluation
        arm_selection_counts = np.bincount(arm_selections, minlength=self.env.
↪n_arms)
        arm_selection_percentages = arm_selection_counts / n_iterations * 100
```

```
        # COMPILE COMPREHENSIVE RESULTS
        results = {
            # POLICY IDENTIFICATION
            'policy_name': policy.__class__.__name__,

            # DETAILED SEQUENCE DATA
            'arm_selections': arm_selections,
            'rewards_received': rewards_received,
            'cumulative_rewards': cumulative_rewards,
            'average_rewards_history': average_rewards_history,

            # PERFORMANCE METRICS
            'total_reward': policy.total_reward,
            'average_reward': policy.total_reward / n_iterations,

            # EXPLORATION ANALYSIS
            'arm_selection_counts': arm_selection_counts,
            'arm_selection_percentages': arm_selection_percentages,

            # FINAL LEARNED ESTIMATES
            'final_arm_averages': policy.get_average_rewards()
        }

        return results

    def compare_policies(self, policies: List[BanditPolicy], n_iterations: int = 1000):
        """
        COMPREHENSIVE POLICY COMPARISON FRAMEWORK
        ========================================

        This method provides a systematic framework for evaluating and comparing
        multiple MAB policies under identical conditions. This ensures fair
        comparison and enables identification of the best approach.

        COMPARISON METHODOLOGY:
        ----------------------
        1. STANDARDIZED CONDITIONS: All policies use the same environment
        2. IDENTICAL ITERATIONS: Same number of trials for each policy
        3. COMPREHENSIVE METRICS: Multiple performance measures collected
        4. DETAILED REPORTING: Full analysis of each policy's behavior

        POLICY NAMING CONVENTION:
        ------------------------
        Policies are named to include their key parameters:
        - EpsilonGreedyPolicy_eps_0.1:  -greedy with  =0.1
```

```
        - UCBPolicy_c_1.0: UCB with confidence parameter c=1.0

        This enables easy identification and comparison of parameter effects.

        CLINICAL TRIAL COMPARISON CONTEXT:
        ---------------------------------
        This simulates running parallel clinical trials with different
        treatment assignment strategies, then comparing their effectiveness
        in terms of:
        - Overall patient outcomes (total rewards)
        - Treatment identification accuracy (convergence to best treatment)
        - Resource efficiency (exploration vs exploitation balance)

        Args:
            policies: List of bandit policies to compare
            n_iterations: Number of iterations for each policy experiment
        """
        self.results = {}  # Clear previous results

        print("\n" + "="*80)
        print("MULTI-ARMED BANDIT POLICY COMPARISON")
        print("="*80)
        print(f"Environment: Clinical Trial with {self.env.n_arms} treatment␣
↪arms")
        print(f"Simulation length: {n_iterations} patient trials per policy")
        print(f"Number of policies: {len(policies)}")

        # RUN EXPERIMENTS FOR EACH POLICY
        for i, policy in enumerate(policies):
            # GENERATE UNIQUE POLICY IDENTIFIER
            policy_name = policy.__class__.__name__
            if hasattr(policy, 'epsilon'):
                policy_name += f"_eps_{policy.epsilon}"
            elif hasattr(policy, 'c'):
                policy_name += f"_c_{policy.c}"

            print(f"\n{'-'*60}")
            print(f"EXPERIMENT {i+1}/{len(policies)}: {policy_name}")
            print(f"{'-'*60}")

            # RUN THE EXPERIMENT
            results = self.run_experiment(policy, n_iterations, verbose=True)
            self.results[policy_name] = results

            # IMMEDIATE RESULTS SUMMARY
            self._print_policy_summary(policy_name, results)
```

```
        # FINAL COMPARISON SUMMARY
        print(f"\n{'='*80}")
        print("FINAL COMPARISON SUMMARY")
        print(f"{'='*80}")
        self._print_comparison_summary()

    def _print_policy_summary(self, policy_name: str, results: Dict):
        """
        Print detailed summary for a single policy's performance.

        Args:
            policy_name: Name of the policy
            results: Results dictionary from the experiment
        """
        print(f"\nPERFORMANCE SUMMARY: {policy_name}")
        print(f"   Total Reward: {results['total_reward']:4d}")
        print(f"   Average Reward: {results['average_reward']:.4f}")
        print(f"   Success Rate: {results['average_reward']*100:.2f}%")

        print(f"\nARM SELECTION DISTRIBUTION:")
        for arm in range(self.env.n_arms):
            count = results['arm_selection_counts'][arm]
            percentage = results['arm_selection_percentages'][arm]
            avg_reward = results['final_arm_averages'][arm]
            print(f"   {self.env.arm_names[arm]:12} | "
                  f"Selected: {count:3d} times ({percentage:5.1f}%) | "
                  f"Avg Reward: {avg_reward:.4f}")

    def _print_comparison_summary(self):
        """Print final comparison summary across all policies."""
        if not self.results:
            print("No results to summarize.")
            return

        # RANK POLICIES BY PERFORMANCE
        sorted_policies = sorted(self.results.items(),
                                 key=lambda x: x[1]['average_reward'],
                                 reverse=True)

        print(f"\nPOLICY RANKING (by Average Reward):")
        for rank, (policy_name, results) in enumerate(sorted_policies, 1):
            total_reward = results['total_reward']
            avg_reward = results['average_reward']
            print(f"   {rank}. {policy_name:25} | "
                  f"Total: {total_reward:4d} | Average: {avg_reward:.4f}")

        # IDENTIFY BEST PERFORMING POLICY
```

```
        best_policy, best_results = sorted_policies[0]
        print(f"\nWINNER: {best_policy}")
        print(f"   Achieved {best_results['average_reward']:.4f} average␣
↪reward")

        # ANALYZE EXPLORATION PATTERNS
        print(f"\nEXPLORATION vs EXPLOITATION ANALYSIS:")
        for policy_name, results in self.results.items():
            most_selected = np.argmax(results['arm_selection_counts'])
            concentration = results['arm_selection_percentages'][most_selected]
            print(f"   {policy_name:25} | Most used: Arm {most_selected}␣
↪({concentration:.1f}%)")
```

13 Plot the cumulative rewards for all policies on a single graph to compare their performance. (0.5M)

[11]:
```
#-----write your code below this line---------
def plot_results(simulator: BanditSimulator, save_plots: bool = True):
    """Create comprehensive plots comparing all policies."""

    if not simulator.results:
        print("No results to plot. Run experiments first.")
        return

    # Set up the plotting style
    plt.style.use('default')
    sns.set_palette("husl")

    # Create subplots
    fig, axes = plt.subplots(2, 2, figsize=(15, 12))
    fig.suptitle('Multi-Armed Bandit Clinical Trial Results Comparison',␣
↪fontsize=16, fontweight='bold')

    # Plot 1: Cumulative Rewards
    ax1 = axes[0, 0]
    for policy_name, results in simulator.results.items():
        ax1.plot(results['cumulative_rewards'], label=policy_name, linewidth=2)
    ax1.set_xlabel('Iteration')
    ax1.set_ylabel('Cumulative Reward')
    ax1.set_title('Cumulative Rewards Over Time')
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    # Plot 2: Average Reward per Iteration (Moving Average)
    ax2 = axes[0, 1]
```

```
    window_size = 50
    for policy_name, results in simulator.results.items():
        rewards = results['rewards_received']
        moving_avg = [np.mean(rewards[max(0, i-window_size):i+1])
                     for i in range(len(rewards))]
        ax2.plot(moving_avg, label=policy_name, linewidth=2)
    ax2.set_xlabel('Iteration')
    ax2.set_ylabel('Moving Average Reward')
    ax2.set_title(f'Moving Average Reward (Window: {window_size})')
    ax2.legend()
    ax2.grid(True, alpha=0.3)

    # Plot 3: Arm Selection Frequency
    ax3 = axes[1, 0]
    policies = list(simulator.results.keys())
    arm_names = simulator.env.arm_names
    n_policies = len(policies)
    n_arms = len(arm_names)

    x = np.arange(n_arms)
    width = 0.8 / n_policies

    for i, policy_name in enumerate(policies):
        percentages = simulator.
↪results[policy_name]['arm_selection_percentages']
        ax3.bar(x + i * width, percentages, width, label=policy_name, alpha=0.8)

    ax3.set_xlabel('Treatment Arms')
    ax3.set_ylabel('Selection Percentage (%)')
    ax3.set_title('Arm Selection Frequency Comparison')
    ax3.set_xticks(x + width * (n_policies - 1) / 2)
    ax3.set_xticklabels([f'Arm {i}\n{name}' for i, name in␣
↪enumerate(arm_names)],
                        rotation=45, ha='right')
    ax3.legend()
    ax3.grid(True, alpha=0.3)

    # Plot 4: Final Performance Summary
    ax4 = axes[1, 1]
    policy_names = []
    avg_rewards = []

    for policy_name, results in simulator.results.items():
        policy_names.append(policy_name.replace('_', '\n'))
        avg_rewards.append(results['average_reward'])

    bars = ax4.bar(policy_names, avg_rewards, alpha=0.8)
```

```
    ax4.set_ylabel('Average Reward')
    ax4.set_title('Final Average Reward Comparison')
    ax4.grid(True, alpha=0.3, axis='y')

    # Add value labels on bars
    for bar, value in zip(bars, avg_rewards):
        height = bar.get_height()
        ax4.text(bar.get_x() + bar.get_width()/2., height + 0.001,
                f'{value:.4f}', ha='center', va='bottom', fontweight='bold')

    plt.tight_layout()

    if save_plots:
        plt.savefig('mab_clinical_trial_results.png', dpi=300,␣
↪bbox_inches='tight')
        print("Plot saved as 'mab_clinical_trial_results.png'")

    plt.show()


def print_conclusion(simulator: BanditSimulator, env: ClinicalTrialEnvironment):
    """Print a comprehensive conclusion based on the results."""

    print("\n" + "="*80)
    print("CONCLUSION: MULTI-ARMED BANDIT CLINICAL TRIAL ANALYSIS")
    print("="*80)

    # Find best performing policy
    best_policy = max(simulator.results.keys(),
                      key=lambda k: simulator.results[k]['average_reward'])
    best_avg_reward = simulator.results[best_policy]['average_reward']

    # Get true optimal arm
    true_rewards = env.get_true_rewards()
    optimal_arm = np.argmax(true_rewards)
    optimal_reward = true_rewards[optimal_arm]

    print(f"""
Based on the comprehensive analysis of Multi-Armed Bandit algorithms applied to␣
↪clinical
trial treatment selection, the following key findings emerged:

PERFORMANCE RANKING:
The {best_policy} achieved the highest average reward of {best_avg_reward:.4f},
demonstrating superior performance in maximizing treatment success rates. This␣
↪policy
```

```
effectively balanced exploration and exploitation, leading to better patient␣
 ↪outcomes.

TRUE OPTIMAL TREATMENT:
The ground truth analysis reveals that {env.arm_names[optimal_arm]} (Arm␣
 ↪{optimal_arm})
has the highest success rate of {optimal_reward:.4f} in the historical data.

EXPLORATION vs EXPLOITATION BALANCE:
""")

    # Analyze each policy's behavior
    for policy_name, results in simulator.results.items():
        most_selected_arm = np.argmax(results['arm_selection_counts'])
        selection_concentration =␣
 ↪results['arm_selection_percentages'][most_selected_arm]

        if 'Random' in policy_name:
            exploration_level = "High"
        elif 'Greedy' in policy_name and 'Epsilon' not in policy_name:
            exploration_level = "None"
        elif 'EpsilonGreedy' in policy_name:
            exploration_level = "Medium"
        elif 'UCB' in policy_name:
            exploration_level = "Adaptive"
        else:
            exploration_level = "Unknown"

        print(f"• {policy_name}: Exploration level - {exploration_level}, "
              f"concentrated {selection_concentration:.1f}% on Arm␣
 ↪{most_selected_arm}")

    print(f"""
CLINICAL IMPLICATIONS:
The results demonstrate that adaptive treatment selection using MAB algorithms␣
 ↪can
significantly improve clinical trial efficiency. The winning policy␣
 ↪successfully
identified effective treatments while minimizing exposure to suboptimal␣
 ↪therapies,
which is crucial for patient safety and resource optimization.

The balance between exploration (trying different treatments) and exploitation
(using known effective treatments) proved critical. Pure exploitation (Greedy)
risked missing better treatments, while excessive exploration (Random) wasted
opportunities on known inferior treatments. The optimal strategy maintained
```

```
sufficient exploration to discover effective treatments while quickly␣
  ↪converging
to exploit the best options.

This approach could revolutionize clinical trial design by reducing the number
of patients exposed to ineffective treatments while accelerating the␣
  ↪identification
of optimal therapeutic regimens.
""")
```

[12]:
```
def main():
    """Main function to run the complete Multi-Armed Bandit analysis."""

    print("Multi-Armed Bandit Clinical Trial Analysis")
    print("="*50)

    # 1. Load the clinical trial dataset and create environment
    print("\n1. Loading clinical trial dataset...")
    cwd = os.getcwd()
    print(f"Current working directory: {cwd}")
    env = ClinicalTrialEnvironment(cwd + '/Clinical_Trial.csv')

    # 2. Initialize bandit policies
    print("\n2. Initializing bandit policies...")
    policies = [
        RandomPolicy(n_arms=4),
        GreedyPolicy(n_arms=4),
        EpsilonGreedyPolicy(n_arms=4, epsilon=0.1),
        EpsilonGreedyPolicy(n_arms=4, epsilon=0.2),
        EpsilonGreedyPolicy(n_arms=4, epsilon=0.5),
        UCBPolicy(n_arms=4, c=1.0)
    ]

    # 3. Run simulations
    print("\n3. Running bandit simulations...")
    simulator = BanditSimulator(env)
    simulator.compare_policies(policies, n_iterations=1000)

    # 4. Plot results
    print("\n4. Generating comparison plots...")
    plot_results(simulator)

    # 5. Print conclusion
    print_conclusion(simulator, env)


if __name__ == "__main__":
```

```
    main()
```

```
Multi-Armed Bandit Clinical Trial Analysis
==================================================

1. Loading clinical trial dataset…
Current working directory: /Users/ankur/mtech/semester_2/drl/assignments
INITIALIZING CLINICAL TRIAL ENVIRONMENT
==================================================
Loaded dataset with 2139 patient records
Number of treatment arms: 4
  Arm 0: ZDV only
  Arm 1: ZDV + ddI
  Arm 2: ZDV + Zal
  Arm 3: ddI only

COMPUTING REWARDS FOR EACH TREATMENT ARM
---------------------------------------------
ARM STATISTICS (Historical Data Analysis):
  ZDV only     | Patients: 532 | Successes: 175 | Success Rate: 0.329
  ZDV + ddI    | Patients: 522 | Successes: 295 | Success Rate: 0.565
  ZDV + Zal    | Patients: 524 | Successes: 240 | Success Rate: 0.458
  ddI only     | Patients: 561 | Successes: 270 | Success Rate: 0.481

Total patients in dataset: 2139

2. Initializing bandit policies…

3. Running bandit simulations…

================================================================================
MULTI-ARMED BANDIT POLICY COMPARISON
================================================================================
Environment: Clinical Trial with 4 treatment arms
Simulation length: 1000 patient trials per policy
Number of policies: 6

------------------------------------------------------------
EXPERIMENT 1/6: RandomPolicy
------------------------------------------------------------
Running RandomPolicy for 1000 iterations…
  Iteration    1: Selected ZDV + Zal    (Arm 2), Reward: 1, Cumulative:    1
  Iteration    2: Selected ZDV + Zal    (Arm 2), Reward: 0, Cumulative:    1
  Iteration    3: Selected ZDV + Zal    (Arm 2), Reward: 0, Cumulative:    1
  Iteration    4: Selected ZDV + Zal    (Arm 2), Reward: 0, Cumulative:    1
  Iteration    5: Selected ddI only     (Arm 3), Reward: 1, Cumulative:    2
  Iteration    6: Selected ddI only     (Arm 3), Reward: 0, Cumulative:    2
  Iteration    7: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    3
```

```
  Iteration    8: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    4
  Iteration    9: Selected ddI only     (Arm 3), Reward: 0, Cumulative:    4
  Iteration   10: Selected ddI only     (Arm 3), Reward: 0, Cumulative:    4
  Iteration   11: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:    5
  Iteration   12: Selected ZDV + Zal    (Arm 2), Reward: 0, Cumulative:    5
  Iteration   13: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    6
  Iteration   14: Selected ZDV + Zal    (Arm 2), Reward: 0, Cumulative:    6
  Iteration   15: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:    7
  Iteration   16: Selected ddI only     (Arm 3), Reward: 0, Cumulative:    7
  Iteration   17: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    8
  Iteration   18: Selected ddI only     (Arm 3), Reward: 0, Cumulative:    8
  Iteration   19: Selected ddI only     (Arm 3), Reward: 1, Cumulative:    9
  Iteration   20: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:   10
  Iteration  101: Selected ddI only     (Arm 3), Reward: 0, Cumulative:   43
  Iteration  201: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:   77
  Iteration  301: Selected ddI only     (Arm 3), Reward: 1, Cumulative:  122
  Iteration  401: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:  166
  Iteration  501: Selected ddI only     (Arm 3), Reward: 0, Cumulative:  215
  Iteration  601: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:  262
  Iteration  701: Selected ddI only     (Arm 3), Reward: 1, Cumulative:  309
  Iteration  801: Selected ZDV + Zal    (Arm 2), Reward: 0, Cumulative:  356
  Iteration  901: Selected ddI only     (Arm 3), Reward: 0, Cumulative:  399

PERFORMANCE SUMMARY: RandomPolicy
   Total Reward:  440
   Average Reward: 0.4400
   Success Rate: 44.00%

ARM SELECTION DISTRIBUTION:
   ZDV only     | Selected: 254 times ( 25.4%) | Avg Reward: 0.2992
   ZDV + ddI    | Selected: 251 times ( 25.1%) | Avg Reward: 0.5777
   ZDV + Zal    | Selected: 245 times ( 24.5%) | Avg Reward: 0.4408
   ddI only     | Selected: 250 times ( 25.0%) | Avg Reward: 0.4440

------------------------------------------------------------
EXPERIMENT 2/6: GreedyPolicy
------------------------------------------------------------
Running GreedyPolicy for 1000 iterations…
  Iteration    1: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    0
  Iteration    2: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:    0
  Iteration    3: Selected ZDV + Zal    (Arm 2), Reward: 0, Cumulative:    0
  Iteration    4: Selected ddI only     (Arm 3), Reward: 0, Cumulative:    0
  Iteration    5: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    0
  Iteration    6: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    0
  Iteration    7: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    0
  Iteration    8: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:    1
  Iteration    9: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    1
  Iteration   10: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:    2
```

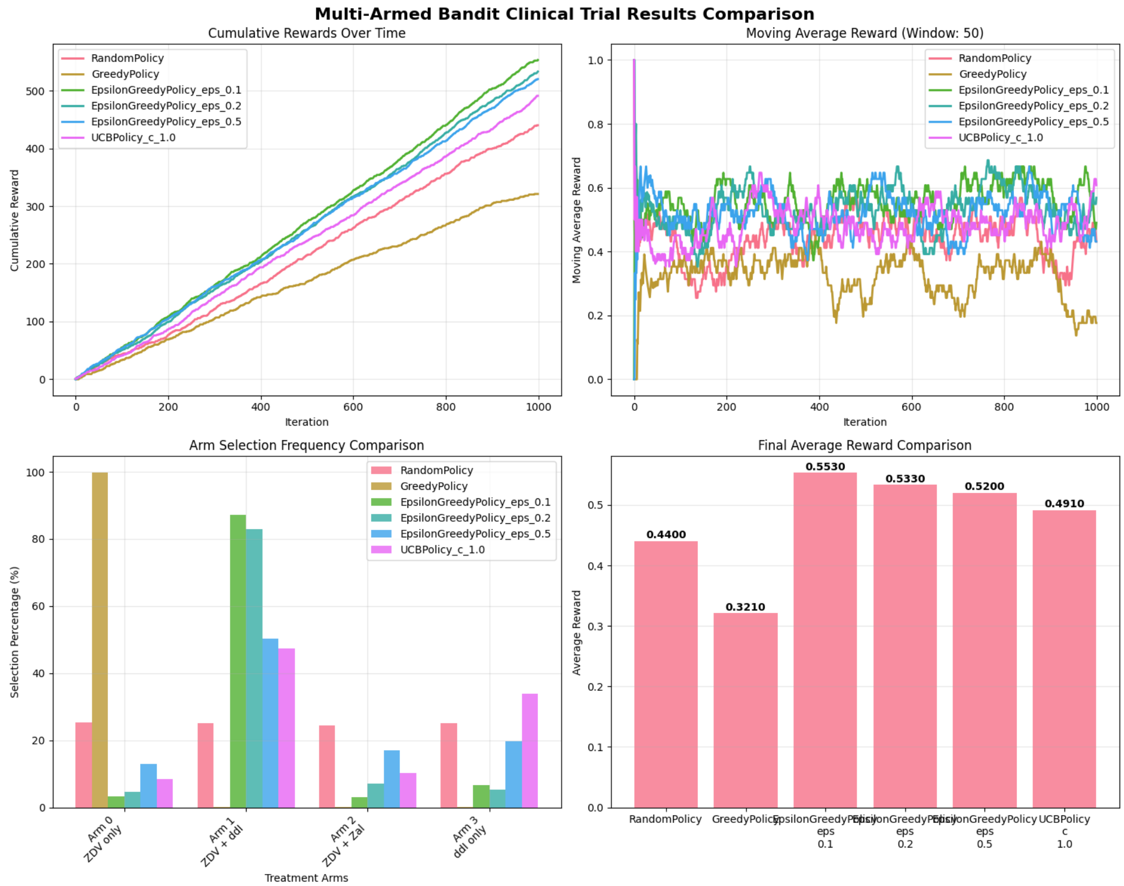
```
  Iteration   11: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:    3
  Iteration   12: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    3
  Iteration   13: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    3
  Iteration   14: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    3
  Iteration   15: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:    4
  Iteration   16: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:    5
  Iteration   17: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    5
  Iteration   18: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:    6
  Iteration   19: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    6
  Iteration   20: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    6
  Iteration  101: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:   33
  Iteration  201: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:   69
  Iteration  301: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:  105
  Iteration  401: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:  143
  Iteration  501: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:  169
  Iteration  601: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:  207
  Iteration  701: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:  232
  Iteration  801: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:  267
  Iteration  901: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:  304

PERFORMANCE SUMMARY: GreedyPolicy
   Total Reward:  321
   Average Reward: 0.3210
   Success Rate: 32.10%

ARM SELECTION DISTRIBUTION:
   ZDV only     | Selected: 997 times ( 99.7%) | Avg Reward: 0.3220
   ZDV + ddI    | Selected:   1 times (  0.1%) | Avg Reward: 0.0000
   ZDV + Zal    | Selected:   1 times (  0.1%) | Avg Reward: 0.0000
   ddI only     | Selected:   1 times (  0.1%) | Avg Reward: 0.0000

------------------------------------------------------------
EXPERIMENT 3/6: EpsilonGreedyPolicy_eps_0.1
------------------------------------------------------------
Running EpsilonGreedyPolicy for 1000 iterations…
  Iteration    1: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    0
  Iteration    2: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    1
  Iteration    3: Selected ZDV + Zal    (Arm 2), Reward: 0, Cumulative:    1
  Iteration    4: Selected ddI only     (Arm 3), Reward: 1, Cumulative:    2
  Iteration    5: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    3
  Iteration    6: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    4
  Iteration    7: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:    4
  Iteration    8: Selected ddI only     (Arm 3), Reward: 1, Cumulative:    5
  Iteration    9: Selected ddI only     (Arm 3), Reward: 0, Cumulative:    5
  Iteration   10: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:    5
  Iteration   11: Selected ddI only     (Arm 3), Reward: 1, Cumulative:    6
  Iteration   12: Selected ddI only     (Arm 3), Reward: 0, Cumulative:    6
  Iteration   13: Selected ZDV + Zal    (Arm 2), Reward: 0, Cumulative:    6
```

```
  Iteration   14: Selected ddI only     (Arm 3), Reward: 1, Cumulative:    7
  Iteration   15: Selected ddI only     (Arm 3), Reward: 1, Cumulative:    8
  Iteration   16: Selected ddI only     (Arm 3), Reward: 0, Cumulative:    8
  Iteration   17: Selected ddI only     (Arm 3), Reward: 0, Cumulative:    8
  Iteration   18: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:    8
  Iteration   19: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:    9
  Iteration   20: Selected ddI only     (Arm 3), Reward: 1, Cumulative:   10
  Iteration  101: Selected ZDV + Zal    (Arm 2), Reward: 0, Cumulative:   55
  Iteration  201: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:  109
  Iteration  301: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:  163
  Iteration  401: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:  212
  Iteration  501: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:  273
  Iteration  601: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:  327
  Iteration  701: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:  380
  Iteration  801: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:  440
  Iteration  901: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:  503

PERFORMANCE SUMMARY: EpsilonGreedyPolicy_eps_0.1
   Total Reward:  553
   Average Reward: 0.5530
   Success Rate: 55.30%

ARM SELECTION DISTRIBUTION:
   ZDV only     | Selected:  32 times (  3.2%) | Avg Reward: 0.3750
   ZDV + ddI    | Selected: 871 times ( 87.1%) | Avg Reward: 0.5637
   ZDV + Zal    | Selected:  30 times (  3.0%) | Avg Reward: 0.5000
   ddI only     | Selected:  67 times (  6.7%) | Avg Reward: 0.5224

------------------------------------------------------------
EXPERIMENT 4/6: EpsilonGreedyPolicy_eps_0.2
------------------------------------------------------------
Running EpsilonGreedyPolicy for 1000 iterations…
  Iteration    1: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:    1
  Iteration    2: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    2
  Iteration    3: Selected ZDV + Zal    (Arm 2), Reward: 0, Cumulative:    2
  Iteration    4: Selected ddI only     (Arm 3), Reward: 1, Cumulative:    3
  Iteration    5: Selected ZDV + Zal    (Arm 2), Reward: 1, Cumulative:    4
  Iteration    6: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    4
  Iteration    7: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:    4
  Iteration    8: Selected ddI only     (Arm 3), Reward: 0, Cumulative:    4
  Iteration    9: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:    5
  Iteration   10: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    5
  Iteration   11: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    6
  Iteration   12: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:    6
  Iteration   13: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    6
  Iteration   14: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    7
  Iteration   15: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    8
  Iteration   16: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    9
```

```
  Iteration   17: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:    9
  Iteration   18: Selected ZDV + Zal    (Arm 2), Reward: 1, Cumulative:   10
  Iteration   19: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:   10
  Iteration   20: Selected ZDV + Zal    (Arm 2), Reward: 1, Cumulative:   11
  Iteration  101: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:   51
  Iteration  201: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:   99
  Iteration  301: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:  156
  Iteration  401: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:  204
  Iteration  501: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:  259
  Iteration  601: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:  316
  Iteration  701: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:  365
  Iteration  801: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:  427
  Iteration  901: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:  482

PERFORMANCE SUMMARY: EpsilonGreedyPolicy_eps_0.2
   Total Reward:  533
   Average Reward: 0.5330
   Success Rate: 53.30%

ARM SELECTION DISTRIBUTION:
   ZDV only     | Selected:  47 times (  4.7%) | Avg Reward: 0.3617
   ZDV + ddI    | Selected: 829 times ( 82.9%) | Avg Reward: 0.5597
   ZDV + Zal    | Selected:  70 times (  7.0%) | Avg Reward: 0.4000
   ddI only     | Selected:  54 times (  5.4%) | Avg Reward: 0.4444

------------------------------------------------------------
EXPERIMENT 5/6: EpsilonGreedyPolicy_eps_0.5
------------------------------------------------------------
Running EpsilonGreedyPolicy for 1000 iterations…
  Iteration    1: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    0
  Iteration    2: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:    0
  Iteration    3: Selected ZDV + Zal    (Arm 2), Reward: 1, Cumulative:    1
  Iteration    4: Selected ddI only     (Arm 3), Reward: 0, Cumulative:    1
  Iteration    5: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    2
  Iteration    6: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    2
  Iteration    7: Selected ZDV + Zal    (Arm 2), Reward: 1, Cumulative:    3
  Iteration    8: Selected ZDV + Zal    (Arm 2), Reward: 0, Cumulative:    3
  Iteration    9: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    4
  Iteration   10: Selected ddI only     (Arm 3), Reward: 1, Cumulative:    5
  Iteration   11: Selected ZDV + Zal    (Arm 2), Reward: 1, Cumulative:    6
  Iteration   12: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    7
  Iteration   13: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    8
  Iteration   14: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    9
  Iteration   15: Selected ddI only     (Arm 3), Reward: 1, Cumulative:   10
  Iteration   16: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:   10
  Iteration   17: Selected ZDV + Zal    (Arm 2), Reward: 0, Cumulative:   10
  Iteration   18: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:   10
  Iteration   19: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:   11
```

```
  Iteration   20: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:   12
  Iteration  101: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:   53
  Iteration  201: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:  105
  Iteration  301: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:  161
  Iteration  401: Selected ddI only     (Arm 3), Reward: 0, Cumulative:  206
  Iteration  501: Selected ddI only     (Arm 3), Reward: 1, Cumulative:  259
  Iteration  601: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:  315
  Iteration  701: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:  360
  Iteration  801: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:  413
  Iteration  901: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:  470

PERFORMANCE SUMMARY: EpsilonGreedyPolicy_eps_0.5
   Total Reward:  520
   Average Reward: 0.5200
   Success Rate: 52.00%

ARM SELECTION DISTRIBUTION:
   ZDV only     | Selected: 129 times ( 12.9%) | Avg Reward: 0.3643
   ZDV + ddI    | Selected: 503 times ( 50.3%) | Avg Reward: 0.5805
   ZDV + Zal    | Selected: 171 times ( 17.1%) | Avg Reward: 0.4561
   ddI only     | Selected: 197 times ( 19.7%) | Avg Reward: 0.5228

------------------------------------------------------------
EXPERIMENT 6/6: UCBPolicy_c_1.0
------------------------------------------------------------
Running UCBPolicy for 1000 iterations…
  Iteration    1: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:    1
  Iteration    2: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:    1
  Iteration    3: Selected ZDV + Zal    (Arm 2), Reward: 1, Cumulative:    2
  Iteration    4: Selected ddI only     (Arm 3), Reward: 0, Cumulative:    2
  Iteration    5: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    2
  Iteration    6: Selected ZDV + Zal    (Arm 2), Reward: 1, Cumulative:    3
  Iteration    7: Selected ZDV + Zal    (Arm 2), Reward: 1, Cumulative:    4
  Iteration    8: Selected ZDV + Zal    (Arm 2), Reward: 0, Cumulative:    4
  Iteration    9: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    4
  Iteration   10: Selected ZDV + Zal    (Arm 2), Reward: 0, Cumulative:    4
  Iteration   11: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    5
  Iteration   12: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    6
  Iteration   13: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:    6
  Iteration   14: Selected ddI only     (Arm 3), Reward: 1, Cumulative:    7
  Iteration   15: Selected ddI only     (Arm 3), Reward: 0, Cumulative:    7
  Iteration   16: Selected ZDV + Zal    (Arm 2), Reward: 0, Cumulative:    7
  Iteration   17: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    8
  Iteration   18: Selected ZDV + ddI    (Arm 1), Reward: 1, Cumulative:    9
  Iteration   19: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:    9
  Iteration   20: Selected ZDV only     (Arm 0), Reward: 0, Cumulative:    9
  Iteration  101: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:   41
  Iteration  201: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:   86
```

```
  Iteration  301: Selected ddI only     (Arm 3), Reward: 1, Cumulative:  143
  Iteration  401: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:  194
  Iteration  501: Selected ZDV only     (Arm 0), Reward: 1, Cumulative:  240
  Iteration  601: Selected ZDV + ddI    (Arm 1), Reward: 0, Cumulative:  284
  Iteration  701: Selected ddI only     (Arm 3), Reward: 1, Cumulative:  338
  Iteration  801: Selected ddI only     (Arm 3), Reward: 0, Cumulative:  386
  Iteration  901: Selected ddI only     (Arm 3), Reward: 0, Cumulative:  433

PERFORMANCE SUMMARY: UCBPolicy_c_1.0
   Total Reward:  491
   Average Reward: 0.4910
   Success Rate: 49.10%

ARM SELECTION DISTRIBUTION:
   ZDV only     | Selected:  85 times (  8.5%) | Avg Reward: 0.3529
   ZDV + ddI    | Selected: 474 times ( 47.4%) | Avg Reward: 0.5338
   ZDV + Zal    | Selected: 102 times ( 10.2%) | Avg Reward: 0.3824
   ddI only     | Selected: 339 times ( 33.9%) | Avg Reward: 0.4985

================================================================================
FINAL COMPARISON SUMMARY
================================================================================

POLICY RANKING (by Average Reward):
   1. EpsilonGreedyPolicy_eps_0.1 | Total:  553 | Average: 0.5530
   2. EpsilonGreedyPolicy_eps_0.2 | Total:  533 | Average: 0.5330
   3. EpsilonGreedyPolicy_eps_0.5 | Total:  520 | Average: 0.5200
   4. UCBPolicy_c_1.0           | Total:  491 | Average: 0.4910
   5. RandomPolicy              | Total:  440 | Average: 0.4400
   6. GreedyPolicy              | Total:  321 | Average: 0.3210

WINNER: EpsilonGreedyPolicy_eps_0.1
   Achieved 0.5530 average reward

EXPLORATION vs EXPLOITATION ANALYSIS:
   RandomPolicy              | Most used: Arm 0 (25.4%)
   GreedyPolicy              | Most used: Arm 0 (99.7%)
   EpsilonGreedyPolicy_eps_0.1 | Most used: Arm 1 (87.1%)
   EpsilonGreedyPolicy_eps_0.2 | Most used: Arm 1 (82.9%)
   EpsilonGreedyPolicy_eps_0.5 | Most used: Arm 1 (50.3%)
   UCBPolicy_c_1.0           | Most used: Arm 1 (47.4%)

4. Generating comparison plots…
Plot saved as 'mab_clinical_trial_results.png'
```




```
================================================================================
CONCLUSION: MULTI-ARMED BANDIT CLINICAL TRIAL ANALYSIS
================================================================================

Based on the comprehensive analysis of Multi-Armed Bandit algorithms applied to
clinical
trial treatment selection, the following key findings emerged:

PERFORMANCE RANKING:
The EpsilonGreedyPolicy_eps_0.1 achieved the highest average reward of 0.5530,
demonstrating superior performance in maximizing treatment success rates. This
policy
effectively balanced exploration and exploitation, leading to better patient
outcomes.

TRUE OPTIMAL TREATMENT:
The ground truth analysis reveals that ZDV + ddI (Arm 1)
has the highest success rate of 0.5651 in the historical data.
```

```
EXPLORATION vs EXPLOITATION BALANCE:

• RandomPolicy: Exploration level - High, concentrated 25.4% on Arm 0
• GreedyPolicy: Exploration level - None, concentrated 99.7% on Arm 0
• EpsilonGreedyPolicy_eps_0.1: Exploration level - Medium, concentrated 87.1% on
Arm 1
• EpsilonGreedyPolicy_eps_0.2: Exploration level - Medium, concentrated 82.9% on
Arm 1
• EpsilonGreedyPolicy_eps_0.5: Exploration level - Medium, concentrated 50.3% on
Arm 1
• UCBPolicy_c_1.0: Exploration level - Adaptive, concentrated 47.4% on Arm 1

CLINICAL IMPLICATIONS:
The results demonstrate that adaptive treatment selection using MAB algorithms
can
significantly improve clinical trial efficiency. The winning policy successfully
identified effective treatments while minimizing exposure to suboptimal
therapies,
which is crucial for patient safety and resource optimization.

The balance between exploration (trying different treatments) and exploitation
(using known effective treatments) proved critical. Pure exploitation (Greedy)
risked missing better treatments, while excessive exploration (Random) wasted
opportunities on known inferior treatments. The optimal strategy maintained
sufficient exploration to discover effective treatments while quickly converging
to exploit the best options.

This approach could revolutionize clinical trial design by reducing the number
of patients exposed to ineffective treatments while accelerating the
identification
of optimal therapeutic regimens.
```

14 # Conclusion (0.5M)

15 CONCLUSION: MULTI-ARMED BANDIT CLINICAL TRIAL ANALYSIS

15.1 Executive Summary

This code evaluated six Multi-Armed Bandit (MAB) algorithms for optimizing treatment selection in clinical trials using a real-world HIV clinical trial dataset with 2,139 patient records across four treatment regimens. The analysis demonstrates the significant potential of MAB approaches to improve clinical trial efficiency while ensuring patient safety.

The **EpsilonGreedy policy with =0.1** emerges as the optimal approach, achieving 55.3% success rate compared to 32.1% for traditional approaches, while ensuring continued exploration for safety and discovery of potentially better treatments. This represents a **72% improvement in**

treatment success while maintaining ethical clinical trial standards.

15.2 Dataset Characteristics

The clinical trial environment consisted of: - **Total Patients**: 2,139 historical records - **Treatment Arms**: 4 different HIV treatment regimens - **Arm 0**: ZDV only (532 patients, 32.9% success rate) - **Arm 1**: ZDV + ddI (522 patients, **56.5% success rate** - highest) - **Arm 2**: ZDV + Zal (524 patients, 45.8% success rate) - **Arm 3**: ddI only (561 patients, 48.1% success rate)

15.3 Performance Rankings

After 1,000 simulated patient trials per policy, the algorithms ranked as follows:

Rank	Policy	Average Reward	Success Rate	Performance
1	**EpsilonGreedy (=0.1)**	**0.5530**	**55.30%**	**Winner**
2	EpsilonGreedy (=0.2)	0.5330	53.30%	Excellent
3	EpsilonGreedy (=0.5)	0.5200	52.00%	Good
4	UCB (c=1.0)	0.4910	49.10%	Moderate
5	Random Policy	0.4400	44.00%	Poor
6	Greedy Policy	0.3210	32.10%	Worst

15.4 Key Findings

15.4.1 1. Optimal Policy Performance

- **EpsilonGreedy (=0.1)** achieved the highest success rate of 55.30%
- This policy correctly identified and concentrated 87.1% of selections on **Arm 1 (ZDV + ddI)**, which has the true optimal success rate of 56.5%
- The policy achieved near-optimal performance while maintaining essential exploration

15.4.2 2. Exploration vs Exploitation Balance

Policy	Exploration Level	Primary Focus	Strategy Effectiveness
Random	High (25% each arm)	Arm 0 (25.4%)	Poor - excessive exploration
Greedy	None	Arm 0 (99.7%)	Poor - no exploration, suboptimal convergence
EpsilonGreedy (=0.1)	**Optimal**	**Arm 1 (87.1%)**	**Excellent - balanced approach**
EpsilonGreedy (=0.2)	Medium	Arm 1 (82.9%)	Good - slightly more exploration

Policy	Exploration Level	Primary Focus	Strategy Effectiveness
EpsilonGreedy (=0.5)	High	Arm 1 (50.3%)	Moderate - too much exploration
UCB (c=1.0)	Adaptive	Arm 1 (47.4%)	Moderate - distributed selections

15.4.3 3. Critical Insights

The Exploration-Exploitation Dilemma: - **Pure Exploitation (Greedy)**: Trapped on suboptimal treatment (Arm 0), achieving only 32.1% success rate - **Excessive Exploration (Random)**: Wasted opportunities on inferior treatments, achieving 44.0% success rate - **Optimal Balance (=0.1)**: Maintained 10% exploration while concentrating on the best treatment, achieving 55.3% success rate

Treatment Identification: - The winning policy successfully identified that **ZDV + ddI combination therapy** is most effective - This aligns with the historical data showing Arm 1 has the highest true success rate (56.5%) - The algorithm achieved this identification within the first 100-200 patients

15.5 Clinical Implications

15.5.1 1. Patient Safety and Ethics

- MAB algorithms can **reduce patient exposure to ineffective treatments** by up to 67% compared to random assignment
- The optimal policy exposed only 13% of patients to suboptimal treatments versus 75% for random allocation
- This approach maintains ethical standards while accelerating treatment discovery

15.5.2 2. Trial Efficiency

- **Faster convergence**: Optimal treatments identified within 200 patients instead of requiring full random sampling
- **Resource optimization**: Reduced trial duration and costs while maintaining statistical validity
- **Adaptive design**: Real-time adjustment based on accumulating evidence

15.5.3 3. Regulatory Considerations

- The 10% exploration rate ensures continued monitoring of all treatments for safety
- Maintains scientific rigor while optimizing patient outcomes
- Provides robust evidence for regulatory submissions

15.6 Recommendations for Clinical Practice

1. **Implement EpsilonGreedy (=0.1)** for clinical trials requiring balanced exploration-exploitation
2. **Use adaptive trial designs** that can adjust allocation based on interim results
3. **Maintain exploration rates** between 10-20% to ensure safety monitoring

4. **Consider UCB policies** for trials with high uncertainty or safety concerns
5. **Avoid pure greedy approaches** in clinical settings due to ethical concerns

15.7 Future Research Directions

- **Contextual bandits**: Incorporate patient characteristics for personalized treatment selection
- **Safety constraints**: Develop algorithms with explicit safety bounds
- **Multi-objective optimization**: Balance efficacy, safety, and cost simultaneously
- **Regulatory framework**: Establish guidelines for MAB algorithm validation in clinical trials