OFF-CMAB: NO ONLINE COMBINATORIAL BANDITS

Lingyao Meng
SIST
University of Shanghaitech
mengly2022@shanghaitech.edu.cn

June 8, 2025

ABSTRACT

Combinatorial Multi-Armed Bandit (CMAB) is a widely used sequential decision-making framework. Existing research mainly focuses on online environments, ignoring the huge consumption and privacy issues of online interactions. The Off-CMAB is an offline learning framework that uses the combined confidence lower bound (CLCB) algorithm, which combines pessimistic reward estimation with a combinatorial solver. At the same time, in order to improve the quality of offline data, two novel data coverage conditions are added, and it is proved that CLCB still has good performance under these conditions. The level of Off-CMAB is verified by three aspects: ranking learning, large language model (LLM) cache, and social influence maximization.

***K*eywords** Offline · Bandits Learning

1 Introduction

1.1 Background & Motivation

CMAB is a widely used sequential decision framework for solving decision problems with combinatorial action spaces. It is widely used in recommender systems, healthcare, autonomous driving and other fields. Traditional Online-CMAB relies on collecting real-time data through interaction with the environment, which is very effective in some cases, but also has significant limitations. In some practical applications, the cost of online exploration is very high, especially in areas where there are safety or ethical issues. For example, in the medical field, it may be inappropriate to conduct real-time experiments on patients. Online data collection may involve the use of sensitive information, which raises privacy and ethical issues, especially in scenarios involving individual health, financial status or personal privacy.

Off-CMAB is a framework for learning based on pre-collected offline datasets, aiming to overcome some challenges in Online-CMAB. By leveraging historical data, Off-CMAB avoids the high cost of online data collection and the privacy and security risks brought by real-time data collection. At the same time, Off-CMAB can handle nonlinear reward functions and general feedback models, and supports out-of-distribution samples (for example, the dataset does not contain the optimal or feasible actions), which also means that Off-CMAB is more robust than traditional offline frameworks. And the framework can not only handle combined action spaces, but also support complex reward structures.

1.2 CORE: Combinatorial Lower Confidence Bound

CLCB uses the pessimism principle to handle the uncertainty in the rewards, particularly when working with offline data that may not cover all actions adequately. This principle penalizes actions with insufficient data to avoid selecting suboptimal actions based on unreliable estimates. For each base arm, the algorithm constructs high-probability **lower confidence bounds** (LCBs), which help mitigate the uncertainty by ensuring that arms with fewer observations are not overestimated. At the combinatorial action level (i.e., when choosing a set of actions), CLCB employs an approximate combinatorial solver to handle the non-linear reward functions effectively.

The algorithm aims to minimize the suboptimality gap, which is the difference in rewards between the optimal action and the action chosen by the algorithm. By using LCB and combinatorial solvers, CLCB reduces this gap, ensuring that even in offline settings, the algorithm performs well and achieves near-optimal results under certain conditions.

2 Problem and Model Settings

Based on combinatorial multi-armed bandits with probabilistically triggered arms **(CMAB-T)** and frames the offline learning problem within this context, base arms, combinatorial actions, probabilistic arm triggering feedback, and the reward function, the offline data collection process and performance metrics for evaluating the effectiveness of offline learning algorithms.

2.1 Definations of CMAB-T

A sequential decision-making process where a learner selects a combinatorial action (a set of base arms) at each round. The environment provides feedback for the selected actions, but feedback for some arms is probabilistic, meaning not all arms within a chosen combinatorial action may be observed. In CMAB-T, a tuple $I := ([m], D, S, D_{\text{trig}}, R)$ defines the problem, where:

- $[m]$ represents the base arms.
- S is the set of feasible combinatorial actions (super arms).
- D denotes the distribution of base arm outcomes.
- D_{trig} is the probabilistic triggering function that dictates which base arms in a combinatorial action are observed based on the outcome of the action.
- R is the reward function that defines how the reward is determined based on the triggered arms' outcomes.

Each round, an action is chosen, and a reward is received based on the outcomes of the arms triggered by that action, which may not always include every arm in the chosen combinatorial action. This setup models real-world situations, like recommendation systems, where not all items in a recommendation list are viewed by a user, and feedback is incomplete.

Let $[m] = \{1, 2, \ldots, m\}$ be the set of base arms. The environment selects a distribution $D_{\text{arm}} \in D$ for each base arm, where D_{arm} defines the possible outcomes of each arm. At each round t, the environment draws outcomes $X_t = (X_{t,1}, X_{t,2}, \ldots, X_{t,m})$ for each base arm, where $X_{t,i}$ represents the outcome of base arm i at round t, drawn from D_{arm}. For each base arm i, μ_i denotes the expected outcome of arm i, which is the mean of the outcome distribution:

$$\mu_i := \mathbb{E}[X_{t,i}] = \mathbb{E}_{X_t \sim D_{\text{arm}}}[X_{t,i}]$$

Each action S is a subset of base arms. Formally, $S \subseteq [m]$, where $[m] = \{1, 2, \ldots, m\}$ is the set of base arms. At each round t, the learner selects a combinatorial action $S_t \in S$, which consists of a set of base arms chosen from the set $[m]$. The reward for each combinatorial action S_t depends on the outcomes of the arms in S_t, but not all arms within S_t may be observed due to the probabilistic triggering of arms. Let $\tau_t \subseteq S_t$ denote the set of triggered arms for action S_t at round t, where the arms in τ_t are the ones that provide feedback in that round. The reward function for combinatorial action S_t can be written as:

$$R(S_t, X_t, \tau_t) = r(S_t; \mu) = \mathbb{E}[R(S_t, X_t, \tau_t)],$$

where $r(S_t; \mu)$ is the expected reward for selecting action S_t, and it depends on the outcomes of the triggered arms in τ_t. After the learner selects a combinatorial action $S_t \in S$ at round t, the outcomes of the arms in S_t are not always observed. The feedback for each base arm in S_t is probabilistic, meaning some arms may not reveal their outcomes. Let $\tau_t \subseteq S_t$ represent the set of base arms that are triggered (i.e., for which feedback is received) after selecting action S_t. The set τ_t is a random subset of S_t, and the triggering of base arms follows a probabilistic distribution D_{trig}, which depends on the outcomes of the arms in S_t. The probabilistic triggering function $D_{\text{trig}}(S_t, X_t)$ determines the probability that each arm $i \in S_t$ is triggered based on the outcome vector X_t. For each arm i, the probability that it is triggered is given by $p_{\text{Darm},S_t,i}$, where:

$$p_{\text{Darm},S_t,i} = \mathbb{P}(\text{arm } i \text{ is triggered} \mid S_t, X_t)$$

The reward for the combinatorial action S_t is determined by the outcomes of the triggered arms τ_t, and the expected reward is calculated as:

$$r(S_t; \mu) = \mathbb{E}[R(S_t, X_t, \tau_t)]$$

where $R(S_t, X_t, \tau_t)$ is the reward function for the combinatorial action, which depends on the triggered arms τ_t. The expected reward for action S_t is:

$$r(S_t; \mu) = \mathbb{E}[R(S_t, X_t, \tau_t)]$$

The reward function for S_t can be expressed as a function of the triggered arms:

$$r(S_t; \mu) = 1 - \prod_{i \in S_t \setminus \tau_t} (1 - \mu_i)$$

Note: If all arms in S_t are triggered (i.e., $\tau_t = S_t$), the reward becomes:

$$r(S_t; \mu) = 1 - \prod_{i \in S_t} (1 - \mu_i)$$

The following are two kinds **Reward conditions**:

1. **Monotonicity Condition:** ensures that the reward function $r(S; \mu)$ is monotonically non-decreasing with respect to the expected outcomes of the base arms. For any two reward vectors μ and μ', if $\mu_i \leq \mu'_i$ for all $i \in [m]$, then the reward for any combinatorial action S satisfies:

$$r(S; \mu) \leq r(S; \mu').$$

2. **1-norm Triggering Probability Modulated (TPM) Bounded Smoothness Condition:** bounds the change in reward due to changes in the expected outcomes of the base arms. The smoothness condition ensures that the reward for a combinatorial action does not change too abruptly.For any two reward vectors μ and μ', and any combinatorial action S, the reward difference is bounded by:

$$|r(S; \mu) - r(S; \mu')| \leq B_1 \sum_{i \in [m]} p_{\text{Darm},S_i} |\mu_i - \mu'_i|$$

where B_1 is the smoothness coefficient and p_{Darm,S_i} is the probability that base arm i is triggered by the action S. This condition ensures that changes in μ_i lead to proportional changes in the reward for the action S, with larger changes in μ_i causing larger reward changes, but controlled by the triggering probabilities p_{Darm,S_i}.

$\star$"**Triggering Probability Modulated**" refers to a technique used in combinatorial multi-armed bandit problems in the context of probabilistically triggered arms. It addresses how the probability of observing feedback (or a reward) from a base arm is modulated based on the action that is chosen and the resulting outcomes of the arms within that action.

Algorithm 1: Combinatorial Multi-armed Bandits with Probabilistically Triggered Arms

Input: Dataset $D = \{(S_t, \tau_t, (X_{t,i})_{i \in \tau_t})\}_{t=1}^{n}$, where S_t is the selected combinatorial action, and $\tau_t \subseteq S_t$ is the set of triggered arms

Output: Action S_t for each round t

for *each round* t **do**
  Select a combinatorial action $S_t \in S$;
  for *each base arm* $i \in S_t$ **do**
    Draw outcome $X_{t,i}$ from the distribution $D_{\text{arm},i}$;
  end
  for *each base arm* $i \in S_t$ **do**
    Calculate the triggering probability $p_{\text{Darm},S_t,i}$ for each arm;
  end
  Observe the set of triggered arms $\tau_t \subseteq S_t$ based on the probabilistic triggering function $D_{\text{trig}}(S_t, X_t)$;
  Calculate the reward for action S_t based on the triggered arms using the reward function $R(S_t, X_t, \tau_t)$;
  Update the expected reward based on the feedback from the triggered arms;
end

2.2 Process about Offline Data

In order to show one of the greatest features of **Off-CMAB**. We focus on the setup for offline learning, where the learner uses pre-collected data to make decisions rather than actively interacting with the environment in real-time.

2.2.1 Offline Datasets

The learner has access to a pre-collected dataset $D = \{(S_t, \tau_t, (X_{t,i})_{i\in\tau_t})\}_{t=1}^{n}$, where S_t represents the combinatorial action (super arm) selected at round t, $\tau_t \subseteq S_t$ denotes the set of arms that were triggered (i.e., the arms that provided feedback) in round t, $X_{t,i}$ is the outcome for each triggered arm $i \in \tau_t$.
The data in the offline dataset is assumed to be collected according to some distribution D_S, where the actions S_t are selected according to this data collection distribution. This distribution can be arbitrary, and the dataset may not necessarily contain optimal actions.

2.2.2 Performance Metric

The performance of an offline learning algorithm is measured by the suboptimality gap, which is defined as the difference in expected reward between the optimal action S^* and the action $\hat{S}$ chosen by the algorithm:

$$\text{SubOpt}(\hat{S}; \alpha, I) = \alpha \cdot r(S^*; \mu) - r(\hat{S}; \mu)$$

where $r(S; \mu)$ is the reward function for action S, and α is an approximation factor that measures the closeness to the optimal action. The goal is to minimize the suboptimality gap, meaning that the chosen action should be as close as possible to the optimal action in terms of expected reward.

2.2.3 Data Coverage

The quality of the offline dataset affects the ability to estimate the optimal action. Reasonable data coverage ensure that the dataset contains enough information to allow the learner to approximate the optimal policy effectively.
Function 1: Infinity-norm TPM Data Coverage ensures that the dataset provides sufficient coverage for the arms in the combinatorial action space, focusing on the most important arms that contribute to the optimal action.

$$\max_{i\in[m]} \frac{p_{\text{Darm},S_i^*}}{p_{\text{Darm},D_{S_i}}} \leq C_*^\infty$$

- p_{Darm,S_i^*} is the probability that arm i is triggered in the optimal action S^*,
- $p_{\text{Darm},D_{S_i}}$ is the probability that arm i is triggered in the data collection distribution D_S,
- C_*^∞ is a constant representing the quality of the data coverage (i.e., how well the dataset approximates the optimal policy).

Function 2: 1-norm TPM Data Coverage quantifies how well the dataset covers the distribution of base arms across different actions, and it ensures that there are enough samples for each arm in the dataset, especially for the arms that are crucial to the optimal action.

$$\sum_{i\in[m]} \frac{p_{\text{Darm},S_i^*}}{p_{\text{Darm},D_{S_i}}} \leq C_*^1$$

- C_*^1 is a constant representing the quality of data coverage in terms of 1-norm.

$\star\star$ **Why ∞-norn & 1-norm**: The p-norm can smoothly account for the quality of coverage of the dataset, it relies too much on local fluctuations in the data. In the multi-arm problem, we are not interested in the small-scale differences in each benchmark arm, so it is not the best fit for this scenario.

3 CLCB: Theoretical Analysis

3.1 Algorithm Design

The algorithm aims to handle uncertainty in offline datasets by applying a pessimistic approach.For each base arm, it computes lower confidence bounds (LCBs) using empirical estimates and penalizing arms with fewer observations.These LCBs are then used as inputs for a combinatorial oracle, which selects an action that approximately maximizes the expected reward.The pessimism principle helps prevent choosing actions that have high uncertainty due to limited data.

Algorithm 2: CLCB: Combinatorial Lower Confidence Bound Algorithm for Off-CMAB

Input: Offline dataset $D = \{(S_t, \tau_t, (X_{t,i})_{i\in\tau_t})\}_{t=1}^{n}$, oracle ORACLE($\mu$), failure probability δ

Output: Selected action $\hat{S}$

for *each arm* $i \in [m]$ **do**
  Initialize counter $N_i = 0$;
  Initialize empirical mean $\hat{\mu}_i = 0$;
end
for *each round* $t = 1, 2, \ldots, n$ **do**
  for *each arm* $i \in S_t$ **do**
    Update the counter $N_i = N_i + 1$;
    Update the empirical mean $\hat{\mu}_i = \frac{1}{N_i} \sum_{t'} I\{i \in \tau_{t'}\} X_{t',i}$;
  end
  for *each arm* $i \in [m]$ **do**
    Compute the lower confidence bound (LCB) for arm i:

$$\text{LCB}_i = \hat{\mu}_i - \sqrt{\frac{\log(4mn/\delta)}{2N_i}}$$

  end
  Call the oracle to select the action $\hat{S} = \text{ORACLE}(\text{LCB}_1, \text{LCB}_2, \ldots, \text{LCB}_m)$;
end
return *The action* $\hat{S}$

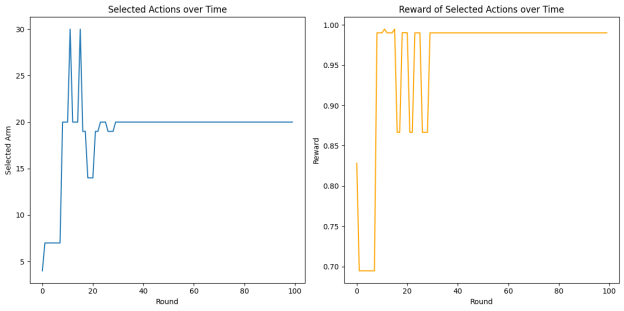

Figure 1: CLCB: arms = 40, rounds = 100, delta = 0.05

3.2 Theoretical limits of the suboptimality gap

Under certain conditions (monotonicity, 1-norm TPM smoothness, and infinity-norm TPM data coverage), the suboptimality gap for the action selected by the CLCB algorithm is bounded by:

$$\text{SubOpt}(\hat{S}; \alpha, I) \leq 2\alpha B_1 K_2^* \sqrt{\frac{2\log(2mn/\delta)}{n}} C_\infty^*$$

- $\hat{S}$ is the action selected by the algorithm, S^* is the optimal action.
- α is the approximation ratio of the oracle used in the algorithm.
- B_1 is a smoothness coefficient from Condition 2 (1-norm TPM smoothness).
- K_2^* is the ℓ_2-norm size of the optimal action.
- C_∞^* is the data coverage coefficient.
- n is the number of samples in the dataset.
- δ is the failure probability, indicating the confidence level.

The suboptimality gap is defined as the difference between the expected reward of the optimal action S^* and the expected reward of the selected action $\hat{S}$:

$$\text{SubOpt}(\hat{S}; \alpha, I) = r(S^*; \mu) - r(\hat{S}; \mu)$$

Where $r(S; \mu)$ is the expected reward for a given action S and mean vector μ.
Our goal is to bound this gap for the CLCB algorithm, which selects actions based on confidence bounds derived from offline data.
Decomposing the suboptimality gap into three components:

$$\text{SubOpt}(\hat{S}; \alpha, I) = \underbrace{r(S^*; \mu) - r(S^*; \hat{\mu})}_{\text{Uncertainty Gap}} + \underbrace{r(S^*; \hat{\mu}) - r(\hat{S}; \hat{\mu})}_{\text{Oracle Gap}} + \underbrace{r(\hat{S}; \hat{\mu}) - r(\hat{S}; \mu)}_{\text{Pessimism Gap}}$$

- **Uncertainty Gap**: This term quantifies how far the estimated means $\hat{\mu}$ are from the true means μ.
- **Oracle Gap**: This term corresponds to the gap in rewards between the optimal action S^* and the action $\hat{S}$ selected by the oracle based on the estimated means $\hat{\mu}$.
- **Pessimism Gap**: This term quantifies the difference between the reward of the action selected by the algorithm under the estimated means and the true reward.

The **uncertainty gap** is bounded using the **confidence bounds** for each arm. The CLCB algorithm uses a lower confidence bound (LCB) for each base arm i, which is:

$$\hat{\mu}_i - \frac{\sqrt{2\log(2mn/\delta)}}{2N_i}$$

- $\hat{\mu}_i$ is the empirical mean of base arm i.
- N_i is the number of times base arm i has been triggered in the dataset.
- m is the number of base arms, and n is the number of samples.

Using **Hoeffding's inequality** and the above LCB, we can ensure that with high probability (at least $1 - \delta$):

$$|\hat{\mu}_i - \mu_i| \leq \frac{\sqrt{2\log(2mn/\delta)}}{2N_i}$$

Thus, the uncertainty gap for each base arm is bounded by:

$$r(S^*; \mu) - r(S^*; \hat{\mu}) \leq \sum_{i \in S^*} p_{S^*}(i) \cdot \frac{\sqrt{2\log(2mn/\delta)}}{2N_i}$$

Where $p_{S^*}(i)$ is the probability that base arm i is included in the optimal action S^*.
The oracle gap measures the difference between the expected rewards of the optimal action and the action selected by the oracle based on the estimated means $\hat{\mu}$. Since the oracle uses the estimated means $\hat{\mu}$ to select an action, it can be shown (using the fact that the oracle is a greedy selection procedure) that:

$$r(S^*; \hat{\mu}) - r(\hat{S}; \hat{\mu}) \leq B_1 \cdot \sum_{i \in S^*} p_{S^*}(i) \cdot |\hat{\mu}_i - \mu_i|$$

Where B_1 is the smoothness coefficient from Condition 2, which bounds the sensitivity of the reward to changes in the base arm means.
Substituting the bounds on $|\hat{\mu}_i - \mu_i|$, we obtain the upper bound on the oracle gap:

$$r(S^*; \hat{\mu}) - r(\hat{S}; \hat{\mu}) \leq B_1 \cdot \sum_{i \in S^*} p_{S^*}(i) \cdot \frac{\sqrt{2\log(2mn/\delta)}}{2N_i}$$

Finally, the **pessimism gap** is bounded by the difference between the reward of the action selected by the algorithm under the estimated means and the true reward. This gap is controlled by the pessimism principle and is bounded by:

$$r(\hat{S}; \hat{\mu}) - r(\hat{S}; \mu) \leq 2 \cdot \alpha \cdot B_1 \cdot \sum_{i \in S^*} p_{S^*}(i) \cdot \frac{\sqrt{2\log(2mn/\delta)}}{2N_i}$$

The final upper bound on the suboptimality gap:

$$\text{SubOpt}(\hat{S}; \alpha, I) \leq 2\alpha B_1 K_2^* \sqrt{\frac{2\log(2mn/\delta)}{n}} C_\infty^*$$

Where K_2^* is the ℓ_2-norm size of the optimal action, and C_∞^* is the data coverage coefficient.

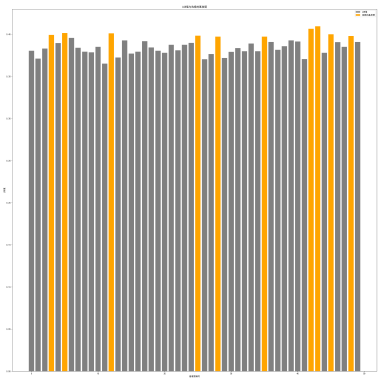

Figure 2: A picture for show Theorem 1

3.3 Lower bound of Problem K-path

In the K**-path problem**, there are m base arms, and these arms are grouped into paths. Each path contains k arms, and the goal is to find the best path that maximizes the expected reward. The problem is framed as selecting a subset of base arms, where each subset represents a combinatorial action that corresponds to selecting a path.

- m: Total number of base arms, k: Number of arms in each path.
- $S \subseteq [m]$: A combinatorial action, which is a subset of base arms (a path).
- The reward for each action S is a function of the mean values of the selected arms.

The optimal action S^* is the path that maximizes the expected reward, which is the sum of the mean values μ_i of the selected arms in that path:

$$r(S^*; \mu) = \sum_{i \in S^*} \mu_i$$

where μ_i is the expected reward of base arm i.
The goal of the bandit algorithm is to select a path S, such that the reward $r(S; \mu)$ is close to $r(S^*; \mu)$, and the **suboptimality gap** $\text{SubOpt}(S; \mu) = r(S^*; \mu) - r(S; \mu)$ is minimized.
Let $\mathcal{D} = (D_{\text{arm}}, D_{\text{S}})$ be a distribution of the data, where D_{arm} is the distribution over the arms, and D_{S} is the distribution over the combinatorial actions (paths). For the k-path problem, we can derive a lower bound on the suboptimality gap as follows:

$$\inf_{A} \sup_{(D_{\text{arm}}, D_{\text{S}}) \in \text{k-path}(m,k,C^*_\infty)} \mathbb{E}\left[r(S^*; \mu) - r(A(D); \mu)\right] \geq \min\left(1, \sqrt{\frac{C^*_\infty}{n}}\right)$$

$\mathbb{E}$ is the expectation over the randomness of the data D. C^*_∞ is the **data coverage coefficient**, which quantifies how well the data collection process covers the optimal action S^*. n is the number of samples used to estimate the rewards. $A(D)$ is the action chosen by the algorithm based on the dataset D.
The lower bound in this theorem shows that there is a minimum gap between the reward of the optimal action S^* and any action chosen by an algorithm. This gap depends on the data coverage and the sample size.
KL Divergence and Covering Arguments: The proof involves Kullback-Leibler divergence and covering arguments that help quantify the minimum information loss (i.e., suboptimality) caused by limited data. The idea is that the gap between the optimal action and the selected action can be bounded below by the divergence between the optimal action's distribution and the distribution of actions chosen by the algorithm.

4 Applications of the Off-CMAB Framewor

This part demonstrates how the framework can address challenges such as handling nonlinear reward functions, general feedback models, and out-of-distribution action samples.

4.1 Learning to Rank (Cascading Bandit)

Off-CMAB can be used to optimize recommendation systems, such as search engines and e-commerce platforms, by learning how to rank items based on user interactions. The framework handles the cascading ban-

dit problem, where a list of items is presented to a user, and only the first satisfactory item is revealed.

Algorithm 3: Learning to Rank: Off-CMAB for Cascading Bandits

Input: Offline dataset $D = \{(S_t, \tau_t, (X_{t,i})_{i\in\tau_t})\}_{t=1}^{n}$ where S_t is the ranked list of items, τ_t is the set of triggered arms, and $(X_{t,i})$ is the observed feedback.
Output: The optimal ranked list S^* based on offline data.

for $t = 1$ ***to*** n **do**
  for *each arm* $i \in [m]$ **do**
    Calculate: Empirical mean of base arm $\hat{\mu}_i = \frac{1}{N_i} \sum_{t=1}^{n} I\{i \in \tau_t\} X_{t,i}$;
    Calculate: Lower confidence bound (LCB) for arm i:

$$\bar{\mu}_i = \hat{\mu}_i - \sqrt{\frac{2\log\left(\frac{4mn}{\delta}\right)}{N_i}}$$

  end
  Call oracle: $S^* = \text{Top-k}(\bar{\mu}_1, \ldots, \bar{\mu}_m)$;
  Return: S^*, the top-ranked list;
end

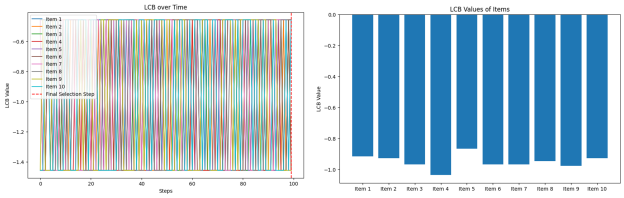

Figure 3: simulation: Learning to rank

4.2 LLM (Large Language Model) Caching

The framework is applied to improve memory management for large language models, aiming to minimize computational costs by caching responses to frequently asked queries. This is essential in real-time, resource-intensive LLM applications.

Algorithm 4: CLCB-LLM-C: Combinatorial Lower Confidence Bound Algorithm for LLM Cache

Input: Offline dataset $D = \{(M_t, q_t, c_t)\}_{t=1}^{n}$ where M_t is the cache, q_t is the query, and c_t is the cost feedback.
Output: The optimal cache M^* based on offline data.

for $t = 1$ ***to*** n **do**
  for *each query* $q \in Q$ **do**
    Calculate: Counter $N(q) = \sum_{t=1}^{n} Iq = q_t$;
    Calculate: Cache miss counter $N_c(q) = \sum_{t=1}^{n} Iq = q_t$ and $q_t \notin M_t$;
    Calculate: Empirical mean of probability $p(q) = \frac{N(q)}{n}$;
    Calculate: Empirical mean of cost $c(q) = \frac{\sum_{t=1}^{n} Iq=q_t \text{ and } q_t \notin M_t c_t}{N_c(q)}$;
    Calculate: Upper confidence bound (UCB) of the cost $c(q) + \sqrt{\frac{2\log(4mn/\delta)}{N_c(q)}}$;
  end
  Call oracle: $M^* = \text{Top-k}\left(\{p(q) \cdot c(q)\}_{q\in Q}\right)$;
  Return: Optimal cache M^*;
end

Algorithm 5: Improved CLCB-LLM-C: Combinatorial Lower Confidence Bound Algorithm for LLM Cache with Streaming

Input: Offline dataset $D = \{(M_t, q_t, c_t)\}_{t=1}^{n}$, query set Q, cache size k, confidence δ.
Output: The optimized cache M^*.

for $t = 1$ ***to*** n **do**
  for *each query* $q \in Q$ **do**
    Update: $N(q)$ (number of times q is seen), $N_c(q)$ (number of times q is missed in cache);
    Update: Empirical means of $p(q)$ and $c(q)$;
    Compute: UCB for cost: $c(q) + \sqrt{\frac{2\log(4mn/\delta)}{N_c(q)}}$;
  end
  if $|M_t| < k$ **then**
    Add query q_t **to cache:** $M_{t+1} = M_t \cup \{q_t\}$;
  end
  else if *cache is full* **then**
    Find the least valuable query: Replace least valuable query with q_t;
    Update cache: $M_{t+1} = M_t - \{q_{\min}\} \cup \{q_t\}$;
  end
  Call oracle: $M^* = \text{Top-k}\left(\{p(q) \cdot c(q)\}_{q \in Q}\right)$;
  Return: Optimized cache M^*;
end

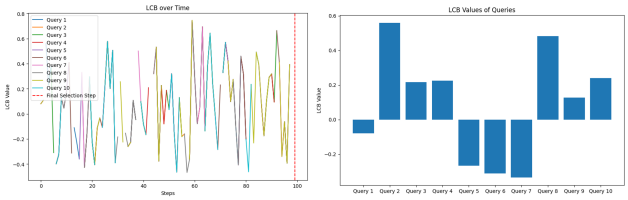

Figure 4: simulation: Large Language Cache

4.3 Social Influence Maximization

Off-CMAB is used to optimize the selection of seed nodes in a social network for viral marketing or epidemic control. The framework handles node-level feedback, making it suitable for real-world scenarios where full edge-level feedback is unavailable.

Algorithm 6: Off-CMAB for Social Influence Maximization with Node-level Feedback

Input: Offline dataset $D = \{(S_t, \tau_t, (X_{t,i})_{i \in \tau_t})\}_{t=1}^{n}$, where S_t is the selected seed set, τ_t is the set of activated nodes, and $(X_{t,i})$ is the node-level feedback (1 for activated, 0 for not activated).
Output: The optimal seed set S^* based on offline data.

for $t = 1$ ***to*** n **do**
  for *each node* $i \in V$ **do**
    Calculate: Empirical mean of activation probability $\hat{p}_i = \frac{1}{n}\sum_{t=1}^{n} I\{i \in \tau_t\}$;
    Calculate: Empirical mean of influence spread $r_i = \frac{1}{n}\sum_{t=1}^{n} I\{i \in \tau_t\} X_{t,i}$;
    Compute: Lower confidence bound (LCB) of influence spread $LCB_i = r_i - \sqrt{\frac{2\log(4mn/\delta)}{N_i}}$, where N_i is the number of times node i has been observed;
  end
  Call oracle: $S^* = \text{Top-k}(\{LCB_1, LCB_2, ..., LCB_m\})$;
  Return: Optimal seed set S^*;
end

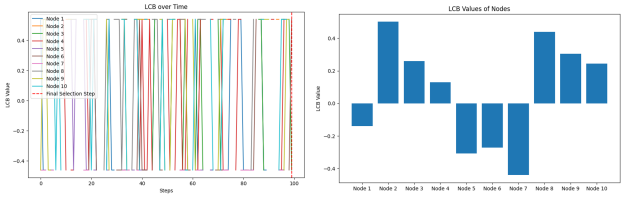

Figure 5: simulation: Social Influence Maximization

5 Experimments Form Paper

The following are some results form the paper.

Application	Smoothness	Data Coverage	Suboptimality Gap	Improvements
Learning to Rank	$B_1 = 1$	$C_1^* = \frac{\mu_1 \cdot m}{\mu_k}$	$\tilde{O}\left(\sqrt{\frac{k}{n} \cdot \frac{m\mu_1}{\mu_k}}\right)^*$	–
LLM Cache	$B_1 = 1$	$C_1^* = m$	$\tilde{O}\left(\sqrt{\frac{m}{n}}\right)$	$\tilde{O}\left(\sqrt{\frac{k^2}{C_1}}\right)^\dagger$
Social Influence Maximization	$B_1 = V$	C_1^*	$\tilde{O}\left(\sqrt{\frac{V^2 d_{\max}^2 \sigma^2(S^*,G)}{\eta \cdot \gamma^3 \cdot n}}\right)$	$\tilde{O}\left(\sqrt{\frac{V^4}{k^2 d_{\max}^2 \eta}}\right)^\ddagger$

Figure 6: Summary of the results of applying the Off-CMAB framework to various applications

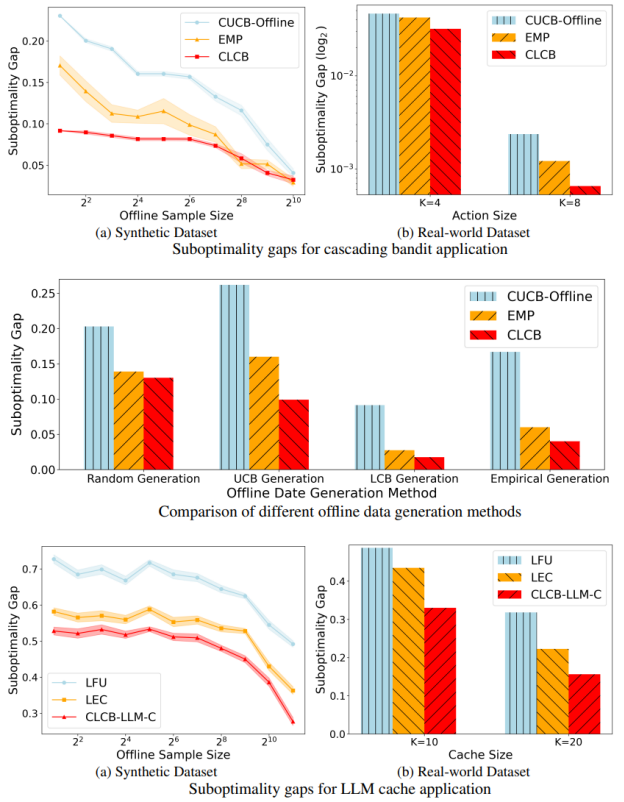

Figure 7: figures about experiments

6 Improvements

6.1 Dynamic Dataset

The distribution of rewards evolves over time, we can adapt the algorithm to respond to this dynamic nature by using dynamic modeling techniques.

1. **Tracking Environmental Changes**: Use exponential moving averages (EMA) or recency-weighted averages to estimate reward distributions that account for recent observations. This allows the model to adapt quickly to shifts in the environment. Implement decay factors for older data to ensure the model doesn't overfit to outdated information. This could be especially important in scenarios where the reward distribution changes frequently.
2. **Dynamic Reward Estimation**: If the rewards change over time, we need to model this non-stationary process. This can be achieved using non-parametric models or regression-based approaches that are updated as new data is observed. For instance, one could use adaptive regression techniques like Ridge regression or Lasso to estimate the reward distribution dynamically.
3. **Modeling Non-Stationarity**: Consider introducing a bandit variant with context-switching or regret bounds under non-stationarity. This allows the model to handle periods where the reward distribution follows different regimes over time. Alternatively, Change-point detection algorithms (e.g., CUSUM) can be integrated to detect significant shifts in the environment, prompting a reset of the model parameters.

Algorithm 7: Dynamic Combinatorial Multi-Armed Bandit with Non-Stationary Rewards

Input: Offline dataset $D = \{(S_t, \tau_t, (X_{t,i})_{i\in\tau_t})\}_{t=1}^{n}$, Learning rate α, Decay factor β, Confidence level δ
Output: Optimal combinatorial action S^*

Function `DynamicCMAB(`D, α, β, δ`):`
  for $i \in [m]$ **do**
    Initialize: $N_i \leftarrow 0$, $\hat{\mu}_i \leftarrow 0$, $\hat{\sigma}_i \leftarrow 0$;
    for $t \in [1, n]$ **do**
      Decay previous observations: $\hat{\mu}_i \leftarrow (1-\beta)\hat{\mu}_i + \beta X_{t,i}$;
      Update the confidence bounds: $LC_i \leftarrow \hat{\mu}_i - \sqrt{\frac{\log(4mn/\delta)}{2N_i}}$;
      if *arm* i *is triggered* **then**
        $N_i \leftarrow N_i + 1$;
        $\hat{\mu}_i \leftarrow \hat{\mu}_i + \frac{X_{t,i}-\hat{\mu}_i}{N_i}$;
        $\hat{\sigma}_i \leftarrow \sqrt{\frac{1}{N_i}\sum_{t=1}^{n}(X_{t,i}-\hat{\mu}_i)^2}$
      end
    end
  end
  Selection of combinatorial action: ;
  $S^* \leftarrow \text{oracle}(LC_1, LC_2, ..., LC_m)$;
  return S^*

6.2 Incorporating Multi-objective Optimization

To incorporate multi-objective optimization into the Combinatorial Lower Confidence Bound (CLCB) algorithm, we would need to adapt the existing framework to optimize for more than one objective simultaneously. This is particularly relevant for real-world applications where decisions often require balancing multiple objectives, such as minimizing cost while maximizing performance.

Theoretical analysis for multi-objective problems typically provides guarantees for **Pareto optimality** or bounded regret for each objective. Adapt the theoretical analysis of CLCB to provide multi-objective guarantees, such as proving that the chosen action is **Pareto optimal** or achieves a desired trade-off between objectives under certain conditions.

1. **Multi-objective Objective Function**: The original CLCB framework is designed to optimize a single reward function. To extend it for multiple objectives, you would need to define a multi-objective reward function that combines all objectives of interest. One common approach is to **normalize** the individual objectives and then

aggregate them using a weighted sum:

$$R(S) = \sum_{i=1}^{M} w_i \cdot R_i(S)$$

where $R_i(S)$ is the reward associated with the i-th objective for action S, and w_i are the weights assigned to each objective. You would adjust the weights according to how much importance each objective has in the given problem.

2. **Multi-objective Action Selection**: In the original algorithm, the decision process involves selecting the action S that maximizes the reward (or minimizes suboptimality gap). In a multi-objective scenario, you could use the **Pareto dominance** approach to identify a set of actions that dominate other actions in terms of all objectives. An action S_1 is said to **dominate** another action S_2 if S_1 is better than S_2 in at least one objective and not worse in the others.Alternatively, you could define an **aggregate score** that combines the objectives and use it to select the action that minimizes the weighted sum of regret or suboptimality.
3. **Modified Suboptimality Gap**: The theoretical analysis of CLCB defines a suboptimality gap for a single objective. For multi-objective optimization, the suboptimality gap can be extended to reflect the performance across all objectives:

$$\text{SubOpt}(S) = \sum_{i=1}^{M} w_i \cdot \text{SubOpt}_i(S)$$

where $\text{SubOpt}_i(S)$ is the suboptimality gap for the i-th objective. This way, the multi-objective suboptimality gap combines the gaps from each individual objective into a single measure.

Algorithm 8: Multi-objective Combinatorial Lower Confidence Bound (CLCB) Algorithm

Input : Dataset $D = \{(S_t, \tau_t, (X_{t,i})_{i\in\tau_t})\}_{t=1}^{n}$, set of objectives $\{O_1, O_2, \ldots, O_M\}$, weights $w_1, w_2, \ldots, w_M$, oracle $ORACLE$, failure probability δ

Output : The selected combinatorial action S^*

for *each objective* $i \in \{1, 2, \ldots, M\}$ **do**

  for *each arm* $i \in [m]$ **do**

    Calculate the number of times arm i is triggered: $N_i = \sum_{t=1}^{n} I_{i\in\tau_t}$;

    Calculate the empirical mean for objective O_i:

$$\hat{\mu}_i = \frac{\sum_{t=1}^{n} I_{i\in\tau_t} X_{t,i}}{N_i}$$

    Compute the lower confidence bound (LCB) for objective O_i:

$$\text{LCB}_i = \hat{\mu}_i - \sqrt{\frac{\log(4mn/\delta)}{2N_i}}$$

  end

  Select action S^* **using the oracle:** Use the oracle $ORACLE$ to select S^* that maximizes the weighted sum of LCBs:

$$\text{Maximize} \quad \sum_{i=1}^{M} w_i \cdot \text{LCB}_i$$

  Store the selected action and its suboptimality gap for each objective O_i:

$$\text{SubOpt}(S^*) = \sum_{i=1}^{M} w_i \cdot \text{SubOpt}_i(S^*)$$

end

return S^*