# Unconstrained Submodular Maximization in Dynamic Setting

**Anonymous Authors**[1]

## Abstract

The unconstrained submodular maximization (USM) problem is one of the most fundamental and challenging versions of submodular maximization problems, where the aim is to find the subset $S$ maximizing $f(S)$ for a given non-monotone submodular function $f$. Buchbinder, Feldman, Naor, and Schwartz, in their *FOCS Test of Time Award*-winning paper, introduced a randomized linear-time algorithm for unconstrained submodular maximization (in offline setting), achieving a tight approximation factor of $0.5$, outperforming all prior algorithms for this problem. Despite the significance of this breakthrough, their algorithm heavily relies on having access to the entire input, which is why it has not yet been extended to fundamental large-scale computational models such as streaming and dynamic settings, unlike other variants of submodular maximization problem, which have been extensively studied in such models. While straightforward extensions of previously known algorithms for the offline USM problem can achieve a $0.5$ approximation with linear update time or a $0.25$ approximation with no update time in dynamic settings, proposing any dynamic algorithm with an approximation factor better than $0.25$ and a sublinear update time had remained an open challenge to this day. In this paper, we present the first dynamic algorithms for USM that break the $0.25$-approximation barrier while maintaining sublinear update time. Our results apply to several natural dynamic models, including incremental updates, decremental updates with known deletion order, and fully dynamic updates with known deletion times.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

## 1. Introduction

Submodular functions provide a powerful mathematical framework for capturing the *diminishing returns* property that naturally arises in many machine learning and decision-making problems (Fujishige, 1984; Tohidi et al., 2020). This property makes submodular optimization particularly well-suited for core machine learning tasks such as data summarization (Simon et al., 2007; Tschiatschek et al., 2014; Sipos et al., 2012), feature selection (Khanna et al., 2017; Das & Kempe, 2008; 2018), and personalized recommendation (El-Arini & Guestrin, 2011). Beyond these applications, submodular maximization also plays a central role in information-theoretic objectives (e.g., mutual information maximization) (Guestrin et al., 2005), graph-structured problems (e.g., cut functions) (Banihashem et al., 2023), and a wide range of optimization tasks across economics, game theory, and operations research. As a result, designing efficient and scalable algorithms for submodular optimization directly impacts both theoretical foundations and practical advances in modern machine learning.

Formally, a nonnegative real-valued function $f : 2^{\mathcal{V}} \to \mathbb{R}_{\geq 0}$ defined over subsets of a ground set $\mathcal{V}$ is called *submodular* if it satisfies $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq \mathcal{V}$. Equivalently, $f$ is submodular if it shows the diminishing returns property: for all $A \subseteq B \subseteq \mathcal{V}$ and all $u \notin B$, $f(A \cup \{u\}) - f(A) \geq f(B \cup \{u\}) - f(B)$. A submodular function is said to be *monotone* if $f(A) \leq f(B)$ for all $A \subseteq B \subseteq \mathcal{V}$. When this property does not hold, the function is referred to as *non-monotone*. Non-monotone submodular maximization generalizes the monotone case and is consequently more challenging. This problem has been studied extensively in the literature (Buchbinder et al., 2015a; Balkanski et al., 2018), with applications including video summarization, movie recommendation (Mirzasoleiman et al., 2016), and revenue optimization in viral marketing campaigns (Hartline et al., 2008).

In this work, we study the problem of *Unconstrained (non-monotone) Submodular Maximization* (often abbreviated as USM), which is one of the most fundamental problems in submodular optimization. The objective of USM is to identify the subset of the ground set that maximizes the submodular function $f$, or formally, to find the subset $S^* \subseteq \mathcal{V}$, for which $f(S^*) = \max_{S \subseteq \mathcal{V}} f(S)$.



For ICML 2026 reviewers: When reviewing this paper, please follow the policy for the use of LLMs as specified on your Reviewer Console. Note that the assigned LLM policy might differ from the policy you indicated during reviewer registration.

In the USM problem, there are no restrictions or constraints on the subset $S^*$, such as limitations on its size or specific structure, allowing any subset of the ground set $\mathcal{V}$ to be a candidate solution. USM has broad applications and serves as a powerful framework for optimizing submodular functions in real-world settings. For example, USM captures many combinatorial optimization problems such as Max-Cut and Max Directed Cut (Feige & Goemans, 1995; Goemans & Williamson, 1995; Halperin & Zwick, 2001; Håstad, 2001; Kindler et al., 2004), Submodular Max-SAT, Submodular Welfare problems (e.g., resource allocation between two players), Maximum Facility Location (Ageev & Sviridenko, 1999; Cornuéjols et al., 1977), and Generalized Assignment (Chekuri & Khanna, 2005; Cohen et al., 2006; Feige & Vondrák, 2006).

USM is an NP-hard problem, which has been extensively studied over the years. However, Feige, Mirrokni, and Vondrák (Feige et al., 2011)(FOCS'07) were the first to introduce constant-factor approximation algorithms for this problem. Within the offline query access model, they demonstrated that selecting a subset $S$ uniformly at random yields a solution that approximates the optimal value by a factor of $0.25$ (or $0.5$ if $f$ is symmetric). Additionally, they developed two local search algorithms to improve over this approximation. Their first local search algorithm directly uses $f$ and achieves a $0.33$ approximation, while their second algorithm utilizes a "noisy" variant of $f$, which further improves the approximation to $0.4$. In their work, Feige, Mirrokni, and Vondrák also studied the inherent hardness of USM. They proved that any algorithm that seeks an approximation factor exceeding $0.5$ would require an exponential number of oracle queries. This hardness result even holds for symmetric submodular functions, where the bound is known to be tight. The subsequent works of (Oveis Gharan & Vondrák, 2011) (SODA'11) and (Feldman et al., 2011) (ICALP'11) further improved the approximation factor for USM in the classic offline setting to $0.41$ and $0.42$, respectively.

Subsequently, Buchbinder, Feldman, Naor, and Schwartz (Buchbinder et al., 2015a) (FOCS'12) published a groundbreaking work in which they proposed a deterministic linear-time algorithm achieving a $0.33$ approximation factor, and a randomized linear-time algorithm with a $0.5$-approximation for the offline USM, the latter of which matched the known lower bound established by (Feige et al., 2011). Their result fully addressed the approximability of the offline USM problem by polynomial time algorithms. Later, Buchbinder and Feldman (Buchbinder & Feldman, 2015) (SODA'16) showed that a $0.5$ approximation ratio for the offline USM can also be achieved by a deterministic algorithm that makes $O(n^2)$ oracle queries. Finally, (Li et al., 2020) (NeurIPS'20) proved that achieving an approximation ratio of $0.25 + \epsilon$ for the USM problem requires $\Omega(\frac{n}{\log n})$ oracle queries.

Despite all the progress made for the USM problem in the offline setting, extending USM to other computational models for large data while preserving efficiency and quality remains an open challenge. Different variants of submodular maximization with additional constraints (such as limits on subset size or structure) have been explored in many large-scale computational models such as streaming (Alaluf et al., 2020; Feldman et al., 2020), dynamic (Lattanzi et al., 2020; Banihashem et al., 2024), distributed, online settings. Nevertheless, to date (Buchbinder et al., 2015b) is the only work that studies USM in such a model beyond the classic offline setting. In their work, Buchbinder, Feldman, and Schwartz consider the online setting. They show that no algorithm can achieve a competitive ratio better than $0.25$ without preemption and propose a polynomial time algorithm with a $0.367$ approximation ratio for the online version of USM with preemption, where discarding elements is irrevocable while the selected elements can be discarded at later stages.

### 1.1. Our contribution

In this paper, we initiate the study of the Unconstrained Submodular Maximization Problem in dynamic settings, where the ground set is subject to updates in the form of insertions or deletions. These updates continuously change the optimal and valid solutions, and we seek to maintain a competitive solution while ensuring fast update times. In this work, we present the first generalizations of the offline algorithm proposed in (Buchbinder et al., 2015a) to dynamic algorithms, which address USM, achieve approximation ratios better than $0.25$, and use sublinear update time.

To highlight the significance and challenges of obtaining these results, we note that, unlike prior dynamic submodular maximization algorithms, which drew inspiration from their streaming counterparts and employed relatively similar mechanisms to adapt to dynamic settings, there is no streaming algorithm for the USM problem. As a result, our algorithms rely on entirely different techniques than their predecessors. Indeed, even for offline USM, the query complexity of algorithms that surpass the $0.25$ approximation ratio is very high. The hardness result of (Li et al., 2020) establishes a lower bound of $\Omega(\frac{n}{\log n})$ on the query complexity required to exceed this approximation ratio. The lowest query complexity among such algorithms belongs to the linear-time algorithms presented in (Buchbinder et al., 2015a), which rely heavily on full access to the entire ground set, making them difficult to adapt beyond the offline model.

We first establish our result for a *dynamic incremental model*, where given an adversarially ordered stream of elements from the underlying ground set $\mathcal{V}$, we show that we can maintain a $0.3$-approximate solution for USM with $O(\sqrt{n})$ amortized query time. This improves upon the guarantees of the following two trivial algorithms:



Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.

**1) Random sampling:** As mentioned above, Feige, Mirrokni, and Vondrák (Feige et al., 2011) showed that selecting a subset of elements uniformly at random yields a 0.25-approximation algorithm for USM. By maintaining a random sample of the elements seen so far, we obtain a dynamic algorithm in the dynamic incremental model that provides a 0.25-approximation to the optimal solution at any time, with $O(1)$ oracle calls per update.

**2) Rerunning the greedy algorithm:** The offline algorithm proposed in (Buchbinder et al., 2015a) is a greedy algorithm that runs in time $O(n)$, where $n = |\mathcal{V}|$, and returns a 0.5-approximate solution to USM. In the dynamic incremental model, we can rerun this algorithm after the arrival of every element, resulting in a trivial algorithm that requires $O(n)$ oracle calls per update.

**Overview of algorithm.** Our algorithm is based on a novel partitioning scheme that balances the trade-off between the approximation factor and update time for USM in the dynamic incremental model. Indeed, we maintain two sets during element insertion: *permanent storage* $\mathcal{N}_1$ and a buffer $\mathcal{N}_2$. Upon arrival of a new element $e$, we first add $e$ to the set $\mathcal{N}_2$. We also maintain a sample set $S_1$ of $\mathcal{N}_1$, where each element in $\mathcal{N}_1$ has a probability of $\frac{1}{2}$ of being in $S_1$. Next, we apply an extended version of the offline algorithm proposed in (Buchbinder et al., 2015a) to expand the sample set $S_1$ using elements of $\mathcal{N}_2$. This yields a solution set, denoted as $\text{Sol}_2$. We repeat this process as long as $|\mathcal{N}_2|$ remains below a threshold of $\sqrt{n}$. Observe that this approach requires one oracle call for $S_1$ and at most $\sqrt{n}$ oracle calls to expand $S_1$ with elements from $\mathcal{N}_2$.

When $|\mathcal{N}_2|$ reaches the threshold $\sqrt{n}$, we transfer its contents to $\mathcal{N}_1$, apply the offline algorithm from (Buchbinder et al., 2015a) to $\mathcal{N}_1$ to update the solution set $\text{Sol}_1$, and resample $S_1$. Since we invoke the linear-time algorithm (Buchbinder et al., 2015a) after every $\sqrt{n}$ insertions, the $O(n)$ oracle calls required by this operation are amortized.

At any point, our reported solution is the maximum of the two maintained sets, $\text{Sol}_1$ and $\text{Sol}_2$. The essence of our analysis lies in the fact that each time we recompute $\mathcal{N}_1$, we obtain a 0.5-approximate solution $\text{Sol}_1$ for all elements seen so far. As long as $\text{Sol}_1$ achieves at least a 0.3-approximation of the optimal value for $\mathcal{N}_1 \cup \mathcal{N}_2$, $\text{Sol}_1$ continues to represent a high-quality solution. However, if at some point $f(\text{Sol}_1)$ falls below 0.3 of the optimal value for $\mathcal{N}_1 \cup \mathcal{N}_2$, we can show that expanding the random sample set $S_1$ with elements from $\mathcal{N}_2$ yields a solution $\text{Sol}_2$ with a submodular value exceeding 0.3 of the optimal value for $\mathcal{N}_1 \cup \mathcal{N}_2$. Specifically, this outcome arises because $f(S_1)$ provides a 0.25-approximation of the optimal submodular value for $\mathcal{N}_1$, and the contribution of $\mathcal{N}_2$ to $\text{Sol}_2$ boosts its value to a 0.3-approximation of the optimal value for $\mathcal{N}_1 \cup \mathcal{N}_2$.

Next, we show that our partitioning scheme extends effectively to two additional dynamic settings. For both models, we can sustain a 0.3-approximate solution for USM with an amortized cost of $O(\sqrt{n})$ oracle calls. Below, we briefly outline these two models; a more detailed explanation and analysis are provided in Appendix B.

**Decremental setting with known-order deletions.** In this setting, we assume that the initial elements provided to the algorithm are sorted according to their deletion order. Specifically, the algorithm receives an initial list of elements $[e_1, e_2, \ldots, e_n]$, and after $t$ deletions, it must return an approximate solution to the problem of finding $\text{OPT}_t = \arg\max_{A \subseteq V_t} f(A)$, where $V_t := \{e_{t+1}, \ldots, e_n\}$. Many problems have been studied in the decremental setting, including single-source shortest paths on undirected graphs (Henzinger et al., 2018), single-source reachability on directed graphs (Henzinger et al., 2015), matching (Assadi et al., 2022), approximate min-cost flow (Bernstein et al., 2021), and strongly-connected components (Bernstein et al., 2019), to name a few.

**Fully dynamic setting with deletion times.** In this setting, we assume that a stream $\mathcal{S}$ of insertion and deletion updates to the problem, determined by an adversary and the deletion time of each element is revealed to the algorithm at the time of its insertion. This model was first proposed in (Mitzenmacher & Vassilvitskii, 2022) and subsequently was studied for other problems, including online matrix vector (Henzinger et al., 2024), transitive closure, triangle detection, single-source reachability (van den Brand et al., 2024), and planar digraph all pairs shortest paths, k-edge connectivity, DFS tree (Liu & Srinivas, 2024), submodular maximization under cardinality constraint (Liu & Srinivas, 2024; Agarwal & Balkanski, 2023).

### 1.2. Related Work

The offline setting of non-monotone submodular maximization has been discussed in detail in the introduction, including the foundational approximation algorithms and matching hardness results. Next, we give an overview of related work on (non-monotone) submodular maximization in streaming and dynamic models.

**Streaming algorithms.** We first review prior results on streaming algorithms for submodular maximization. For monotone submodular maximization, Badanidiyuru et al. (Badanidiyuru et al., 2014) introduced an insertion-only streaming algorithm achieving a $(0.5 - \epsilon)$-approximation under a cardinality constraint $k$. Chekuri, Gupta, and Quanrud (Chekuri et al., 2015) extended the streaming framework to both monotone and non-monotone submodular functions subject to $p$-matchoid constraints.



Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.

Subsequent work by Mirzasoleiman et al. (Mirzasoleiman et al., 2018) and Feldman et al. (Feldman et al., 2018) developed improved streaming algorithms for non-monotone submodular maximization under $p$-matchoid constraints.

For the cardinality-constrained non-monotone setting, the current state of the art is due to Alaluf et al. (Alaluf et al., 2020), who achieved a $(0.277 + \epsilon)$-approximation, improving upon the previous 0.17-approximation of Feldman et al. (Feldman et al., 2018). We also note that there exist streaming algorithms for monotone submodular maximization that support deletions (Kazemi et al., 2018; Mirzasoleiman et al., 2017). However, in these dynamic settings, the space and update time typically scale with the number of deletions, which in the worst case can be as large as $\Omega(n)$, where $n$ denotes the size of the ground set.

**Dynamic algorithms.** In the dynamic setting, monotone submodular maximization with a cardinality constraint $k$ was first investigated by Lattanzi et al. (Lattanzi et al., 2020) and Monemizadeh (Monemizadeh, 2020) at NeurIPS'20. Both approaches developed dynamic algorithms that maintain $(0.5 - \epsilon)$-approximate solutions with efficient query times. Specifically, the algorithm from Lattanzi et al. (Lattanzi et al., 2020) has an expected amortized query complexity of $O(\epsilon^{-11} \log^6(k) \log^2(n))$, while Monemizadeh's algorithm (Monemizadeh, 2020) achieves an amortized expected query complexity of $O(\epsilon^{-3} k^2 \log^5(n))$. Chen and Peng (Chen & Peng, 2022) later proved that developing a $c$-approximation dynamic algorithm for $c > 0.5$ would require a polynomial number of oracle queries.

More recently, Duetting et al. (Duetting et al., 2023) examined monotone submodular maximization under a matroid constraint in the dynamic setting, proposing a $(0.25 - \epsilon)$-approximation algorithm with an amortized expected query complexity of $\mathcal{O}(\frac{k^2}{\epsilon} \log(k) \log^2(n) \log^3(\frac{k}{\epsilon}))$. Simultaneously, Banihashem et al. (Banihashem et al., 2024) achieved similar approximation guarantees with improved query complexities, specifically providing a worst-case expected query complexity of $\mathcal{O}(k \log(k) \log^3(\frac{k}{\epsilon}))$. They also improved the query complexity of Monemizadeh's dynamic algorithm for cardinality constraints to an expected $\mathcal{O}(k\epsilon^{-1} \log^2(k))$.

For non-monotone submodular maximization in dynamic settings, Banihashem et al. (Banihashem et al., 2023) proposed a dynamic algorithm that maintains a $(0.125 - \epsilon)$-approximate solution under cardinality constraints with expected amortized query complexity $O(\epsilon^{-2} k^2 \log^3(k))$.

### 1.3. Preliminaries

**Definition 1** (Submodular function)**.** *Let $\mathcal{V}$ be a ground set consisting of elements. We define a function $f : 2^{\mathcal{V}} \to \mathbb{R}_{\geq 0}$ as submodular if it satisfies the following inequality for any subsets $A, B \subseteq \mathcal{V}$: $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$.*

This condition implies that adding an element to a smaller subset provides at least as much marginal gain as adding it to a larger subset. Equivalently, for any subsets $A \subseteq B \subseteq \mathcal{V}$ and an element $e \in \mathcal{V} \setminus B$, submodularity requires: $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$. We define the *marginal gain* of adding element $e$ to set $A$ as: $\Delta(e \mid A) := f(A \cup \{e\}) - f(A)$. Similarly, for any sets $A, B \subseteq \mathcal{V}$, we define: $\Delta(B \mid A) := f(A \cup B) - f(A)$. A function $f$ is called *monotone* if $f(A) \leq f(B)$ for any $A \subseteq B \subseteq \mathcal{V}$. In contrast, a function is *non-monotone* if this property does not necessarily hold.

**Definition 2** (Unconstrained (non-monotone) submodular maximization (USM))**.** *In USM, the goal is to identify a subset $S^* \subseteq \mathcal{V}$ that maximizes the submodular function $f$, or formally, to find $f(S^*) = \max_{S \subseteq \mathcal{V}} f(S)$.*

**Query model.** For dynamic analysis, as explored in recent works (Lattanzi et al., 2020; Monemizadeh, 2020; Chen & Peng, 2022; Duetting et al., 2023), we assume access to $f$ through an *oracle*. This oracle provides the ability to perform *set queries*, where for any subset $A \subseteq \mathcal{V}$, one can retrieve the value $f(A)$. The marginal gain $\Delta_f(e \mid A) \doteq f(A \cup \{e\}) - f(A)$ can be computed using two set queries: first querying $f(A \cup \{e\})$, and then querying $f(A)$.

Consider a sequence of operations, $\mathcal{S}$, consisting of insertions and deletions applied to the input of our problem. We assume $f : 2^{\mathcal{V}} \to \mathbb{R}_{\geq 0}$ is a (potentially non-monotone) submodular function defined on subsets of $V$. The time $t$ corresponds to the $t^{\text{th}}$ update (either an insertion or deletion) in the sequence. Let $\mathcal{S}_t$ denote the subsequence of updates from the beginning of $\mathcal{S}$ up to time $t$, and let $V_t \subseteq \mathcal{V}$ represent the set of elements that have been inserted but not deleted up to time $t$. Thus, $V_t$ constitutes the current ground set of elements. We denote the optimal value at time $t$ by $OPT_t = \max_{S \subseteq V_t : |S| \leq k} f(S)$.

**Definition 3** (Query complexity)**.** *The query complexity of a dynamic $\alpha$-approximate algorithm refers to the number of oracle queries required by the algorithm to compute a solution $S_t$ with respect to the ground set $V_t$, such that $f(S_t) \geq \alpha \cdot OPT_t$.*

It is important to note that dynamic algorithms maintain a record of all previous queries, which can potentially be utilized to find $S_t$ at the current time $t$. The dynamic algorithms that we develop operate under the *oblivious adversarial model*, a common framework for analyzing randomized data structures like universal hashing (Carter & Wegman, 1977). In this model, the adversary is aware of the submodular function $f$ and the algorithm being used, allowing it to determine the sequence of insertions and deletions. However, the adversary cannot adapt its strategy based on the algorithm's random choices and thus cannot change the updates dynamically in response to these choices.



Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.

## 2. The Dynamic Unconstrained Algorithm

In this section, we will introduce the first dynamic algorithm for the unconstrained submodular maximization problem, marking a significant advancement in the field as the first algorithm to address this problem in a setting beyond the classic offline framework. However, before delving into the original algorithm, we will briefly discuss a subroutine that we have repeatedly utilized in implementing our subsequent algorithms.

### 2.1. Generalized Offline Unconstrained Submodular Maximization

In their groundbreaking study, Buchbinder, Feldman, Naor, and Schwartz (Buchbinder et al., 2012) introduced a tight randomized algorithm for the *Unconstrained Submodular Maximization* (USM) problem in the offline setting, which they referred to as RANDOMIZEDUSM. This algorithm produces a solution whose submodular value achieves a $1/2$ approximation of the optimal solution and has an execution time that is linear in the size of its input.

Before moving on to our main algorithms, we present the procedure EXTEND as our first building block, which is a slightly modified version of the RANDOMIZEDUSM algorithm. By using a proof that closely follows that of RANDOMIZEDUSM, it can be shown that EXTEND solves a generalized form of the offline unconstrained submodular maximization problem. This generalized version of the algorithm will later serve as a subroutine in developing our dynamic algorithms throughout the paper.

**Algorithm 1** Generalized USM

**function** EXTEND($S, \mathcal{N}$)
  $X_0 \leftarrow S,\ Y_0 \leftarrow S \cup \mathcal{N}$
  **for** $i = 1$ to $|\mathcal{N}|$ **do**
    $a_i \leftarrow f(X_{i-1} \cup \{s_i\}) - f(X_{i-1})$
    $b_i \leftarrow f(Y_{i-1} \setminus \{s_i\}) - f(Y_{i-1})$
    $a'_i \leftarrow \max(a_i, 0),\ b'_i \leftarrow \max(b_i, 0)$
    **with probability** $a'_i/(a'_i + b'_i)$ **do**
      $X_i \leftarrow X_{i-1} \cup \{s_i\},\ Y_i \leftarrow Y_{i-1}$
    **else do**
      $X_i \leftarrow X_{i-1},\ Y_i \leftarrow Y_{i-1} \setminus \{s_i\}$
  **return** $X_{|\mathcal{N}|}$ (or equivalently $Y_{|\mathcal{N}|}$)

Below, we formally introduce the Generalized Unconstrained Submodular Maximization problem, solved by the procedure Extend presented in Algorithm 1.

**Definition 4** (Optimal Extension)**.** *Let $f$ be a non-negative submodular function defined over a ground set $\mathcal{V}$. For any pair $(S, \mathcal{N})$ of disjoint subsets of $\mathcal{V}$ (i.e., $S \cap \mathcal{N} = \emptyset$), we define the optimal extension of $S$ with respect to $\mathcal{N}$ as*

$$OPT(S, \mathcal{N}) = \arg \max_{S \subseteq A \subseteq S \cup \mathcal{N}} f(A),$$

*and refer to the* optimal extension problem *as the task of finding a set $A$ that is a superset of $S$, contained within $S \cup \mathcal{N}$, which maximizes $f(A)$.*

**Lemma 5.** *Given the input sets $S$ and $\mathcal{N}$, Algorithm 1 makes $O(|\mathcal{N}|)$ query calls and returns the set $X_{|\mathcal{N}|}$, for which the following inequality holds:*

$$f(OPT(S, \mathcal{N})) + \frac{1}{2} f(S) + \frac{1}{2} f(S \cup \mathcal{N}) \leq 2\mathbb{E}\left[ f(X_{|\mathcal{N}|}) \right].$$

*Proof.* The proof is exactly similar to the proof of RANDOMIZEDUSM (Buchbinder et al., 2012). Alternatively, it can be derived directly from their result by considering the function $g(A) := f(S \cup A)$ and noting that $g$ is also a non-negative submodular function defined over $\mathcal{V} \supseteq \mathcal{N}$. □

### 2.2. Incremental Dynamic Algorithm for USM

We now proceed to present our dynamic algorithm for the Unconstrained Submodular Maximization Problem (USM) in the incremental setting.

#### 2.2.1. SETTING

Recall that the incremental dynamic setting refers to a framework where the input of the problem undergoes incremental changes over time. This means the algorithm receives a stream of updates in the form of element insertions. The goal is to efficiently maintain an approximately optimal solution for the updated input set. The performance of such algorithms is evaluated based on the quality of the solution they maintain and the computational complexity of the tasks performed after each update.

Specifically, at each time $t$, our algorithm receives an element $e_t \in \mathcal{V}$. Its objective is to return an approximate solution to the problem of finding $\text{OPT}_t = \arg\max_{A \subseteq V_t} f(A)$, where $V_t$ is the set of elements inserted up to time $t$, i.e., $\{e_1, \ldots, e_t\}$. We measure the performance of our algorithm by the number of queries it makes after each update and the approximation ratio of the solution it maintains.

Note that for simplicity, we assume that we know the number of updates in the input stream, i.e., the size of the final input for the USM problem, which we denote by $n$. However, we will later indicate that this assumption is not necessary.

#### 2.2.2. ALGORITHM DESCRIPTION

In this algorithm (Algorithm 2), we maintain two sets $\mathcal{N}_1$ and $\mathcal{N}_2$ to store the inserted elements, two solution sets $\text{Sol}_1$ and $\text{Sol}_2$, and an additional set $S_1$ to store a random subset of $\mathcal{N}_1$. Initially, all the sets mentioned above are initialized as empty. We then process each incoming element insertion as follows:

After an element $e$ gets inserted, we first add this new ele-



Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.

ment to the set $\mathcal{N}_2$. Throughout the algorithm, $\mathcal{N}_2$ acts as our buffer, while $\mathcal{N}_1$ acts as our permanent storage. After adding each element to $\mathcal{N}_2$, we check the size of the set $\mathcal{N}_2$, and in the case that the size of $\mathcal{N}_2$ had reached the buffer threshold $\lfloor \sqrt{n} \rfloor$, we transfer its content to $\mathcal{N}_1$. Note that during the execution of our algorithm, the union of these two sets is always equal to the set of all elements inserted from the beginning until the current time frame $t$, denoted as $V_t$. Additionally, whenever we modify the set $\mathcal{N}_1$, that is whenever we transfer the elements of the buffer $\mathcal{N}_2$ into it, we also update $S_1$ and $\text{Sol}_1$ accordingly, such that the followings hold:

- The set $S_1$ is always a subset of $\mathcal{N}_1$, and it includes each element $e \in \mathcal{N}_1$ with a probability of $\frac{1}{2}$.
- The set $\text{Sol}_1$ always maintains an approximate for the optimal solution restricted to the elements of $\mathcal{N}_1$.

After modifying $\mathcal{N}_2$, and applying any necessary changes to sets $\mathcal{N}_1$, $S_1$, and $\text{Sol}_1$, we update Sol2 by invoking the function EXTEND with the updated pair $(S_1, \mathcal{N}_2)$ as its input. Finally, we conclude this process by returning the best solution among $\text{Sol}_1$ and $\text{Sol}_2$.

**Algorithm 2** DYNAMICINSERTION

```
function MAIN()
    N1, N2, S1, Sol1, Sol2 ← ∅
    for each element e in the stream do
        yield INSERT(e)
function INSERT(e)
    N2 ← N2 ∪ {e}
    if |N2| = ⌊√n⌋ then
        N1 ← N1 ∪ N2, N2 ← ∅
        Sol1 ← Extend(∅, N1)
        S1 ← N1(1/2)
    Sol2 ← Extend(S1, N2)
    return arg max{f(Sol1), f(Sol2)}
```

### 2.2.3. ANALYSIS

In this section, we prove the following theorem by establishing the approximation guarantee and query complexity of Algorithm 2.

**Theorem 6.** *There exists an algorithm for the USM problem in the incremental dynamic setting described in Section 2.2.1 that achieves a 0.3 approximation with an amortized query complexity of $O(\sqrt{n})$ per update.*

*Proof.* We prove that Algorithm 2 maintains a 0.3-approximate solution for the unconstrained submodular maximization problem in Lemma 7, and we also prove that its amortized query complexity is $O(\sqrt{n})$ per update in Lemma 20. This completes the proof. □

**Lemma 7** (Approximation Guarantee)**.** *Consider any fixed time $t$. Let $V_t$ denote the set of elements inserted during the first $t$ updates, and let $\text{OPT}_t$ be the optimal subset of $V_t$. The output of Algorithm 2 after update $t$, denoted by $\text{Sol}_t$, provides a 0.3-approximation of $\text{OPT}_t$. Specifically, we have:*

$$\mathbb{E}[\, f(\text{Sol}_t)\,] = \mathbb{E}[\, \max(f(\text{Sol}_1), f(\text{Sol}_2))\,] \geq 0.3 f(\text{OPT}_t).$$

*Proof.* We begin by fixing the time $t$ and will establish a series of lemmas that culminate in Lemma 19.

At any point during the execution of this algorithm, the elements of $V_t$ are partitioned between the two sets $\mathcal{N}_1$ and $\mathcal{N}_2$. Consequently, we introduce an alternative definition for $\text{OPT}_t$, which we will denote simply as OPT from this point onward. Additionally, we define the term $\overline{\text{OPT}}$, which will be used later in our proof.

**Definition 8.** *Let $\text{OPT} := \arg\max_{A \subseteq \mathcal{N}_1 \cup \mathcal{N}_2} f(A)$, and define $\overline{\text{OPT}} := (\mathcal{N}_1 \cup \mathcal{N}_2) \setminus \text{OPT}$.*

According to Lemma 19, we have:

$$\max\left(\mathbb{E}[\, f(\text{Sol}_1)\,], \mathbb{E}[\, f(\text{Sol}_2)\,]\right) \geq \frac{3}{10} f(\text{OPT}),$$

which implies, by Jensen's inequality, that $\mathbb{E}[\max(f(\text{Sol}_1), f(\text{Sol}_2))] \geq \frac{3}{10} f(\text{OPT})$. This completes the proof. □

**Lemma 9.** *For a fixed set $S_1$, the following inequality holds:*

$$f(\text{OPT}(S_1, \mathcal{N}_2)) + \frac{1}{2}\left(f(S_1) + f(S_1 \cup \mathcal{N}_2)\right) \leq 2\mathbb{E}[\, f(\text{Sol}_2)\,].$$

*Proof.* Recall that after each update, the algorithm executes the EXTEND function on input sets $S_1$ and $\mathcal{N}_2$, storing its output in $\text{Sol}_2$ right before returning its solution.

Also, by Lemma 5, we know that for any input sets $S$ and $\mathcal{N}$, the function EXTEND ensures the following inequality:

$$f(\text{OPT}(S, \mathcal{N})) + \frac{1}{2} f(S) + \frac{1}{2} f(S \cup \mathcal{N}) \leq 2\mathbb{E}\left[\, f(X_{|\mathcal{N}|})\,\right],$$

where $X_{|\mathcal{N}|}$ denotes its output.

Thus, by applying this result to our specific sets $S_1$ and $\mathcal{N}_2$, we obtain the inequality in the lemma statement.

It's important to note that in this lemma, we consider a fixed (determined) value for the set $S_1$; thus, $f(\text{OPT}(S_1, \mathcal{N}_2))$, $f(S_1)$, and $f(S_1 \cup \mathcal{N}_2)$ are all constants, and $\mathbb{E}[\, f(\text{Sol}_2)\,]$ only accounts for the randomness of the function EXTEND, not the randomness of the set $S_1$.

□



Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.

**Lemma 10.** *For a fixed set $S_1$, the following inequality holds:* $2\mathbb{E}[f(Sol_2)] \geq f(S_1 \cup (OPT \cap \mathcal{N}_2)) + \frac{1}{2}(f(S_1) + f(S_1 \cup \mathcal{N}_2))$

*Proof.* Definition 4, defines $\mathrm{OPT}(S_1, \mathcal{N}_2)$ or the optimal extension of $S_1$ with respect to $\mathcal{N}_2$ as $\arg\max_{S_1 \subseteq A \subseteq S_1 \cup \mathcal{N}_2} f(A)$. This implies that for any set $A$ with $S_1 \subseteq A$ and $A \subseteq S_1 \cup \mathcal{N}_2$, we have: $f(\mathrm{OPT}(S_1, \mathcal{N}_2)) \geq f(A)$. Since $S_1 \subseteq S_1 \cup (\mathrm{OPT} \cap \mathcal{N}_2)$ and $S_1 \cup (\mathrm{OPT} \cap \mathcal{N}_2) \subseteq S_1 \cup \mathcal{N}_2$, we get: $f(\mathrm{OPT}(S_1, \mathcal{N}_2)) \geq f(S_1 \cup (\mathrm{OPT} \cap \mathcal{N}_2))$. Additionally, by Lemma 9, we have: $2\mathbb{E}[f(\mathrm{Sol}_2)] \geq f(\mathrm{OPT}(S_1, \mathcal{N}_2)) + \frac{1}{2}(f(S_1) + f(S_1 \cup \mathcal{N}_2))$. Combining the last two inequalities yields the desired inequality and completes the proof. □

As noted, the earlier lemmas assumed a fixed value for $S_1$, disregarding the randomness associated with $S_1$. We will now analyze how this inherent randomness of $S_1$ affects the quality of our solution. To do so, we leverage Lemma 2.3 from the work of Feige, Mirrokni, and Vondrák (Feige et al., 2011), which we restate below for convenience.

**Lemma 11** (Lemma 2.3 of (Feige et al., 2011))**.** *Let $f : 2^X \to R$ be submodular, $A, B \subseteq X$ two (not necessarily disjoint) sets and $A(p)$ and $B(q)$ their independently sampled subsets, where each element of $A$ appears in $A(p)$ with probability $p$ and each element of $B$ appears in $B(q)$ with probability $q$. Then* $\mathbb{E}[f(A(p) \cup B(q))] \geq (1-p)(1-q)f(\emptyset) + p(1-q)f(A) + (1-p)qf(B) + pqf(A \cup B)$ .

**Corollary 12.** *We have:* $2\mathbb{E}[f(Sol_2)] \geq \mathbb{E}[f(\mathcal{N}_1(\frac{1}{2}) \cup (OPT \cap \mathcal{N}_2))] + \frac{1}{2}\mathbb{E}[f(\mathcal{N}_1(\frac{1}{2}))] + \frac{1}{2}\mathbb{E}[f(\mathcal{N}_1(\frac{1}{2}) \cup \mathcal{N}_2)]$ .

*Proof.* This corollary follows directly from Lemma 10 by replacing the fixed set $S_1$ with the random subset $\mathcal{N}_1(\frac{1}{2})$ and then taking the expectation over this randomness. □

**Lemma 13.** *We have:* $\mathbb{E}[f(\mathcal{N}_1(\frac{1}{2}))] \geq \frac{1}{4}\big(f(\emptyset) + f(OPT \cap \mathcal{N}_1) + f(\overline{OPT} \cap \mathcal{N}_1) + f(\mathcal{N}_1)\big)$.

*Proof.* Firstly note that $\mathcal{N}_1 = (\mathrm{OPT} \cap \mathcal{N}_1) \cup (\overline{\mathrm{OPT}} \cap \mathcal{N}_1)$. Hence, we have: $\mathcal{N}_1(\frac{1}{2}) = (\mathrm{OPT} \cap \mathcal{N}_1)(\frac{1}{2}) \cup (\overline{\mathrm{OPT}} \cap \mathcal{N}_1)(\frac{1}{2})$. By setting $p = q = \frac{1}{2}$, $A = \mathrm{OPT} \cap \mathcal{N}_1$, and $B = \overline{\mathrm{OPT}} \cap \mathcal{N}_1$ in Lemma 11, we obtain:

$$\begin{aligned}\mathbb{E}\left[f(\mathcal{N}_1(\frac{1}{2}))\right] &\geq \frac{1}{4}(f(\emptyset) + f(\mathrm{OPT} \cap \mathcal{N}_1) \\ &+ f(\overline{\mathrm{OPT}} \cap \mathcal{N}_1) + f((\mathrm{OPT} \cap \mathcal{N}_1) \cup (\overline{\mathrm{OPT}} \cap \mathcal{N}_1))) \\ &= \frac{1}{4}\left(f(\emptyset) + f(\mathrm{OPT} \cap \mathcal{N}_1) + f(\overline{\mathrm{OPT}} \cap \mathcal{N}_1) + f(\mathcal{N}_1)\right).\end{aligned}$$

□

**Fact 14.** *Let $f$ be a non-negative submodular function, and let $C$ and $A$ be subsets of its ground set. Then, the function $g(A) := f(A \cup C)$ is also a non-negative submodular function.*

**Lemma 15.** $\mathbb{E}[f(\mathcal{N}_1(\frac{1}{2}) \cup (OPT \cap \mathcal{N}_2))]$ *is lower-bounded by*

$$\begin{aligned}&\frac{1}{4}(f(OPT \cap \mathcal{N}_2) + f(OPT) \\ &+ f((\overline{OPT} \cap \mathcal{N}_1) \cup (OPT \cap \mathcal{N}_2)) + f(\mathcal{N}_1 \cup OPT)).\end{aligned}$$

*Proof.* We first apply Lemma 11, with $p = q = \frac{1}{2}$, $A = \mathrm{OPT} \cap \mathcal{N}_1$, and $B = \overline{\mathrm{OPT}} \cap \mathcal{N}_1$ and function $g(A) := f(A \cup (\mathrm{OPT} \cap \mathcal{N}_2))$, which is also a non-negative submodular function based on Fact 14. $\mathbb{E}[g(\mathcal{N}_1(\frac{1}{2}))] \geq \frac{1}{4}\big(g(\emptyset) + g(\mathrm{OPT} \cap \mathcal{N}_1) + g(\overline{\mathrm{OPT}} \cap \mathcal{N}_1) + g(\mathcal{N}_1)\big)$ ..

This implies that:

$$\begin{aligned}&\mathbb{E}\left[f(\mathcal{N}_1(\frac{1}{2}) \cup (\mathrm{OPT} \cap \mathcal{N}_2))\right] \geq \frac{1}{4}f(\emptyset \cup (\mathrm{OPT} \cap \mathcal{N}_2)) \\ &+ \frac{1}{4}f((\mathrm{OPT} \cap \mathcal{N}_1) \cup (\mathrm{OPT} \cap \mathcal{N}_2)) \\ &+ \frac{1}{4}f((\overline{\mathrm{OPT}} \cap \mathcal{N}_1) \cup (\mathrm{OPT} \cap \mathcal{N}_2)) \\ &+ \frac{1}{4}f(\mathcal{N}_1 \cup (\mathrm{OPT} \cap \mathcal{N}_2)) \\ &= \frac{1}{4}(f(\mathrm{OPT} \cap \mathcal{N}_2) + f(\mathrm{OPT}) \\ &+ f((\overline{\mathrm{OPT}} \cap \mathcal{N}_1) \cup (\mathrm{OPT} \cap \mathcal{N}_2)) + f(\mathcal{N}_1 \cup \mathrm{OPT})),\end{aligned}$$

which establishes the lower bound as required. □

**Lemma 16.** *We have:* $\mathbb{E}[f(\mathcal{N}_1(\frac{1}{2}) \cup \mathcal{N}_2)] \geq \frac{1}{4}\big(f(\mathcal{N}_2) + f(OPT \cup \mathcal{N}_2) + f(\overline{OPT} \cup \mathcal{N}_2) + f(\mathcal{N}_1 \cup \mathcal{N}_2)\big)$

*Proof.* Similar to Lemma 16, we apply Lemma 11 with $p = q = 1/2$, $A = \mathrm{OPT} \cap \mathcal{N}_1$, and $B = \overline{\mathrm{OPT}} \cap \mathcal{N}_1$, and function $g(A) := f(A \cup \mathcal{N}_2)$, and derive the following inequality:

$$\begin{aligned}&\mathbb{E}\left[f(\mathcal{N}_1(\frac{1}{2}))\right] \geq \frac{1}{4}(f(\emptyset \cup \mathcal{N}_2) + f((\mathrm{OPT} \cap \mathcal{N}_1) \cup \mathcal{N}_2) \\ &+ f((\overline{\mathrm{OPT}} \cap \mathcal{N}_1) \cup \mathcal{N}_2) + f(\mathcal{N}_1 \cup \mathcal{N}_2)).\end{aligned}$$

Next, we utilize the facts that: $\emptyset \cup \mathcal{N}_2 = \mathcal{N}_2$, $(\mathrm{OPT} \cap \mathcal{N}_1) \cup \mathcal{N}_2 = \mathrm{OPT} \cup \mathcal{N}_2$, and $(\overline{\mathrm{OPT}} \cap \mathcal{N}_1) \cup \mathcal{N}_2 = \overline{\mathrm{OPT}} \cup \mathcal{N}_2$. Substituting these into our earlier expression gives us the desired inequality, thereby completing the proof. □

The proof of the following lemma is given in Appendix A



Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.

**Lemma 17.** *The following inequality holds:*

$$2\mathbb{E}[f(Sol_2)] \geq \frac{1}{2}f(OPT) + \frac{1}{4}f(OPT \cap \mathcal{N}_2) + \frac{1}{8}f((\overline{OPT} \cap \mathcal{N}_1) \cup (OPT \cap \mathcal{N}_2)) + \frac{1}{8}f(\mathcal{N}_1 \cup OPT) + \frac{1}{4}f(\emptyset) + \frac{1}{8}f(\overline{OPT} \cap \mathcal{N}_1) + \frac{1}{8}f(\mathcal{N}_1) + \frac{1}{4}f(\overline{OPT} \cup \mathcal{N}_2) + \frac{1}{4}f(\mathcal{N}_1 \cup \mathcal{N}_2).$$

The bound established in Lemma 17 immediately results in the following Corollary, isolating the components we will utilize in our final proof.

**Corollary 18.** *The following inequality holds:*

$$2\mathbb{E}[f(Sol_2)] \geq \frac{1}{2}f(OPT) + \frac{1}{4}f(OPT \cap \mathcal{N}_2).$$

**Lemma 19.** *The maximum expected submodular value of $Sol_1$ and $Sol_2$ provides a $\frac{3}{10}$-approximation of $f(OPT)$. Specifically,*

$$\max(\mathbb{E}[f(Sol_1)], \mathbb{E}[f(Sol_2)]) \geq \frac{3}{10}f(OPT).$$

*Proof.* Having established a lower bound for the submodular value of $\text{Sol}_2$, we argue that $\text{Sol}_2$ provides a good approximation of OPT, unless our buffer $\mathcal{N}_2$ does not contribute to the optimal solution; in that case, the focus should shift to $\mathcal{N}_1$, which we have already addressed. The formal proof follows.

We proceed by analyzing two complementary cases.

**Case 1:** $f(\text{OPT} \cap \mathcal{N}_1) \geq \frac{6}{10}f(\text{OPT})$.

In this case, we show that $\mathbb{E}[f(\text{Sol}_1)] \geq \frac{3}{10}f(\text{OPT})$. Recall that in Algorithm DYNAMICINSERTION, $\text{Sol}_1$ is the output of the subroutine Extend$(\emptyset, \mathcal{N}_1)$. Applying Lemma 5, we have:

$$2\mathbb{E}[f(\text{Sol}_1)] \geq f(\text{OPT}(\emptyset, \mathcal{N}_1)) \geq f(\text{OPT} \cap \mathcal{N}_1) \geq \frac{6}{10}f(\text{OPT}).$$

This implies: $\mathbb{E}[f(\text{Sol}_1)] \geq \frac{3}{10}f(\text{OPT})$.

**Case 2:** $f(\text{OPT} \cap \mathcal{N}_1) < \frac{6}{10}f(\text{OPT})$.

By the submodularity property of the function $f$, we have $f(\text{OPT}) + f(\emptyset) \leq f(\text{OPT} \cap \mathcal{N}_1) + f(\text{OPT} \cap \mathcal{N}_2) < \frac{6}{10}f(\text{OPT}) + f(\text{OPT} \cap \mathcal{N}_2)$. Using the non-negativity of $f$ implies: $f(\text{OPT} \cap \mathcal{N}_2) > \frac{4}{10}f(\text{OPT}) + f(\emptyset) \geq \frac{4}{10}f(\text{OPT})$.

Next, by Corollary 18, we have that:

$$2\mathbb{E}[f(\text{Sol}_2)] \geq \frac{1}{2}f(\text{OPT}) + \frac{1}{4}f(\text{OPT} \cap \mathcal{N}_2).$$

Substituting the lower bound on $f(\text{OPT} \cap \mathcal{N}_2)$ yields:

$$2\mathbb{E}[f(\text{Sol}_2)] > \left(\frac{1}{2} + \frac{1}{4} \times \frac{4}{10}\right) f(\text{OPT}),$$

which simplifies to: $\mathbb{E}[f(\text{Sol}_2)] > \frac{3}{10}f(\text{OPT})$. Thus, in both cases, we have:

$$\max(\mathbb{E}[f(\text{Sol}_1)], \mathbb{E}[f(\text{Sol}_2)]) \geq \frac{3}{10}f(\text{OPT}),$$

□

**Lemma 20.** *The amortized time complexity of our algorithm is $O(\sqrt{n})$.*

*Proof.* As explained in Section 2.2.1, for simplicity, we assume that the size of our final input (i.e., $n$) is known from the beginning, and we use $\sqrt{n}$ as our buffer threshold.

First, note that the time complexity of executing EXTEND$(S, \mathcal{N})$ is $O(|\mathcal{N}|)$. After each update, our algorithm executes the EXTEND function on the input sets $S_1$ and $\mathcal{N}_1$, which makes $O(|\mathcal{N}_1|) = O(\sqrt{n})$ query calls since $|\mathcal{N}_1|$ is always less than the buffer threshold of $\sqrt{n}$.

Thus, we only need to focus on the number of queries made to compute $\text{Sol}_1$. We invoke the EXTEND function to compute $\text{Sol}_1$ $O(\sqrt{n})$ times, with one invocation occurring after every $\sqrt{n}$ insertions without any such invocation. The query complexity of none of these invocations exceeds $O(n)$, so the amortized query complexity of these invocations is also $O(\sqrt{n})$ per element. Hence, the proof is complete.

We also address the problem without the assumption of knowing $n$ from the beginning. Consider the following fix:

Let $b$ denote the buffer threshold. Start with a constant buffer threshold, for example, $b = 2$, and double it when the number of insertions exceeds $b^2$.

First, observe that $b$ never exceeds $2\sqrt{n}$. Similar to the previous case, we do not need to consider the queries made to compute $\text{Sol}_2$. Thus, we only focus on the computations of $\text{Sol}_1$. Note that invocations related to $\text{Sol}_1$ occur only when an update causes the transfer of elements from the buffer to $\mathcal{N}_1$, which happens when the buffer reaches $b$ elements. Consider such an update $t$. We charge the cost of the queries made after this update to the $b$ elements transferred. The total number of queries made for this update is less than $b^2$, meaning each of these $b$ elements gets charged for no more than $b$ queries.

Since $b = O(\sqrt{n})$, and each element is charged only once, the amortized query complexity of the algorithm remains $O(\sqrt{n})$ per element.

□



Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.

## Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. The nature of our work is mainly theoretical and focus on submodular maximization, and thus its insights might find use in various areas.

## References


Agarwal, A. and Balkanski, E. Learning-augmented dynamic submodular maximization. *CoRR*, abs/2311.13006, 2023. doi: 10.48550/ARXIV.2311.13006. URL https://doi.org/10.48550/arXiv.2311.13006.

Ageev, A. A. and Sviridenko, M. An 0.828-approximation algorithm for the uncapacitated facility location problem. *Discret. Appl. Math.*, 93:149–156, 1999. URL https://api.semanticscholar.org/CorpusID:14585942.

Alaluf, N., Ene, A., Feldman, M., Nguyen, H. L., and Suh, A. Optimal streaming algorithms for submodular maximization with cardinality constraints. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pp. 6:1–6:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi: 10.4230/LIPIcs.ICALP.2020.6. URL https://doi.org/10.4230/LIPIcs.ICALP.2020.6.

Assadi, S., Bernstein, A., and Dudeja, A. Decremental matching in general graphs. In Bojanczyk, M., Merelli, E., and Woodruff, D. P. (eds.), *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pp. 11:1–11:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi: 10.4230/LIPICS.ICALP.2022.11. URL https://doi.org/10.4230/LIPIcs.ICALP.2022.11.

Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., and Krause, A. Streaming submodular maximization: massive data summarization on the fly. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014.

Balkanski, E., Breuer, A., and Singer, Y. Non-monotone submodular maximization in exponentially fewer iterations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 2359–2370, Red Hook, NY, USA, 2018. Curran Associates Inc.

Banihashem, K., Biabani, L., Goudarzi, S., Hajiaghayi, M., Jabbarzade, P., and Monemizadeh, M. Dynamic non-monotone submodular maximization. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/387982dbf23d9975c7fc45813dd3dabc\protect\discretionary{\char\hyphenchar\font}{}{}Abstract-Conference.html.

Banihashem, K., Biabani, L., Goudarzi, S., Hajiaghayi, M., Jabbarzade, P., and Monemizadeh, M. Dynamic algorithms for matroid submodular maximization. In Woodruff, D. P. (ed.), *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pp. 3485–3533. SIAM, 2024. doi: 10.1137/1.9781611977912.125. URL https://doi.org/10.1137/1.9781611977912.125.

Bernstein, A., Probst, M., and Wulff-Nilsen, C. Decremental strongly-connected components and single-source reachability in near-linear time. In Charikar, M. and Cohen, E. (eds.), *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pp. 365–376. ACM, 2019. doi: 10.1145/3313276.3316335. URL https://doi.org/10.1145/3313276.3316335.

Bernstein, A., Gutenberg, M. P., and Saranurak, T. Deterministic decremental SSSP and approximate min-cost flow in almost-linear time. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pp. 1000–1008. IEEE, 2021. doi: 10.1109/FOCS52979.2021.00100. URL https://doi.org/10.1109/FOCS52979.2021.00100.

Buchbinder, N. and Feldman, M. Deterministic algorithms for submodular maximization problems. *ACM Transactions on Algorithms (TALG)*, 14:1 – 20, 2015. URL https://api.semanticscholar.org/CorpusID:10181845.

Buchbinder, N., Feldman, M., Naor, J., and Schwartz, R. A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pp. 649–658. IEEE Computer Society, 2012. doi: 10.1109/FOCS.2012.73. URL https://doi.org/10.1109/FOCS.2012.73.

Buchbinder, N., Feldman, M., Naor, J., and Schwartz, R. A tight linear time (1/2)-approximation for unconstrained




Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.

submodular maximization. *SIAM J. Comput.*, 44(5):1384–1402, 2015a. doi: 10.1137/130929205. URL https://doi.org/10.1137/130929205.

Buchbinder, N., Feldman, M., and Schwartz, R. Online submodular maximization with preemption. *ACM Transactions on Algorithms (TALG)*, 15:1 – 31, 2015b. URL https://api.semanticscholar.org/CorpusID:10303316.

Carter, L. and Wegman, M. N. Universal classes of hash functions (extended abstract). In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pp. 106–112, 1977. doi: 10.1145/800105.803400. URL https://doi.org/10.1145/800105.803400.

Chekuri, C. and Khanna, S. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 35:713–728, 2005. URL https://api.semanticscholar.org/CorpusID:14609890.

Chekuri, C., Gupta, S., and Quanrud, K. Streaming algorithms for submodular function maximization. In *Automata, Languages, and Programming*, pp. 318–330, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

Chen, X. and Peng, B. On the complexity of dynamic submodular maximization. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pp. 1685–1698. ACM, 2022. doi: 10.1145/3519935.3519951. URL https://doi.org/10.1145/3519935.3519951.

Cohen, R., Katzir, L., and Raz, D. An efficient approximation for the generalized assignment problem. *Inf. Process. Lett.*, 100:162–166, 2006. URL https://api.semanticscholar.org/CorpusID:14897015.

Cornuéjols, G., Fisher, M. L., and Nemhauser, G. L. On the uncapacitated location problem. *Annals of discrete mathematics*, 1:163–177, 1977. URL https://api.semanticscholar.org/CorpusID:115375975.

Das, A. and Kempe, D. Algorithms for subset selection in linear regression. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pp. 45–54. ACM, 2008. doi: 10.1145/1374376.1374384. URL https://doi.org/10.1145/1374376.1374384.

Das, A. and Kempe, D. Approximate submodularity and its applications: Subset selection, sparse approximation and dictionary selection. *J. Mach. Learn. Res.*, 19:3:1–3:34, 2018. URL http://jmlr.org/papers/v19/16-534.html.

Duetting, P., Fusco, F., Lattanzi, S., Norouzi-Fard, A., and Zadimoghaddam, M. Fully dynamic submodular maximization over matroids. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 8821–8835. PMLR, 2023. URL https://proceedings.mlr.press/v202/duetting23a.html.

El-Arini, K. and Guestrin, C. Beyond keyword search: discovering relevant scientific literature. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pp. 439–447. ACM, 2011. doi: 10.1145/2020408.2020479. URL https://doi.org/10.1145/2020408.2020479.

Feige, U. and Goemans, M. X. Approximating the value of two power proof systems, with applications to max 2sat and max dicut. *Proceedings Third Israel Symposium on the Theory of Computing and Systems*, pp. 182–189, 1995. URL https://api.semanticscholar.org/CorpusID:2553935.

Feige, U. and Vondrák, J. Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e. *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pp. 667–676, 2006. URL https://api.semanticscholar.org/CorpusID:6006402.

Feige, U., Mirrokni, V. S., and Vondrák, J. Maximizing non-monotone submodular functions. *SIAM J. Comput.*, 40(4):1133–1153, 2011. doi: 10.1137/090779346. URL https://doi.org/10.1137/090779346.

Feldman, M., Naor, J., and Schwartz, R. Nonmonotone submodular maximization via a structural continuous greedy algorithm - (extended abstract). In *International Colloquium on Automata, Languages and Programming*, 2011. URL https://api.semanticscholar.org/CorpusID:1865603.

Feldman, M., Karbasi, A., and Kazemi, E. Do less, get more: Streaming submodular maximization with subsampling. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 730–740, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/d1f255a373a3cef72e03aa9d980c7eca-Abstract.html.

Feldman, M., Norouzi-Fard, A., Svensson, O., and Zenklusen, R. The one-way communication complexity of



Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.

submodular maximization with applications to streaming and robustness. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pp. 1363–1374. ACM, 2020. doi: 10.1145/3357713.3384286. URL https://doi.org/10.1145/3357713.3384286.

Fujishige, S. Theory of submodular programs: A fenchel-type min-max theorem and subgradients of submodular functions. *Math. Program.*, 29(2):142–155, 1984. doi: 10.1007/BF02592218. URL https://doi.org/10.1007/BF02592218.

Goemans, M. X. and Williamson, D. P. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42:1115–1145, 1995. URL https://api.semanticscholar.org/CorpusID:15794408.

Guestrin, C., Krause, A., and Singh, A. P. Near-optimal sensor placements in gaussian processes. In Raedt, L. D. and Wrobel, S. (eds.), *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, volume 119 of *ACM International Conference Proceeding Series*, pp. 265–272. ACM, 2005. doi: 10.1145/1102351.1102385. URL https://doi.org/10.1145/1102351.1102385.

Halperin, E. and Zwick, U. Combinatorial approximation algorithms for the maximum directed cut problem. In *ACM-SIAM Symposium on Discrete Algorithms*, 2001. URL https://api.semanticscholar.org/CorpusID:18822824.

Hartline, J., Mirrokni, V., and Sundararajan, M. Optimal marketing strategies over social networks. In *Proceedings of the 17th international conference on World Wide Web*, pp. 189–198, 2008.

Håstad, J. Some optimal inapproximability results. *Electron. Colloquium Comput. Complex.*, TR97, 2001. URL https://api.semanticscholar.org/CorpusID:5120748.

Henzinger, M., Krinninger, S., and Nanongkai, D. Improved algorithms for decremental single-source reachability on directed graphs. In Halldórsson, M. M., Iwama, K., Kobayashi, N., and Speckmann, B. (eds.), *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pp. 725–736. Springer, 2015. doi: 10.1007/978-3-662-47672-7_59. URL https://doi.org/10.1007/978-3-662-47672-7_59.

Henzinger, M., Krinninger, S., and Nanongkai, D. Decremental single-source shortest paths on undirected graphs in near-linear total update time. *J. ACM*, 65(6):36:1–36:40, 2018. doi: 10.1145/3218657. URL https://doi.org/10.1145/3218657.

Henzinger, M., Saha, B., Seybold, M. P., and Ye, C. On the complexity of algorithms with predictions for dynamic graph problems. In Guruswami, V. (ed.), *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA*, volume 287 of *LIPIcs*, pp. 62:1–62:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi: 10.4230/LIPICS.ITCS.2024.62. URL https://doi.org/10.4230/LIPIcs.ITCS.2024.62.

Kazemi, E., Zadimoghaddam, M., and Karbasi, A. Scalable deletion-robust submodular maximization: Data summarization with privacy and fairness constraints. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2549–2558. PMLR, 2018. URL http://proceedings.mlr.press/v80/kazemi18a.html.

Khanna, R., Elenberg, E. R., Dimakis, A. G., Negahban, S. N., and Ghosh, J. Scalable greedy feature selection via weak submodularity. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*, pp. 1560–1568. PMLR, 2017. URL http://proceedings.mlr.press/v54/khanna17b.html.

Kindler, G., O'Donnell, R., Khot, S., and Mossel, E. Optimal inapproximability results for max-cut and other 2-variable csps? *45th Annual IEEE Symposium on Foundations of Computer Science*, pp. 146–154, 2004. URL https://api.semanticscholar.org/CorpusID:2090495.

Lattanzi, S., Mitrovic, S., Norouzi-Fard, A., Tarnawski, J., and Zadimoghaddam, M. Fully dynamic algorithm for constrained submodular optimization. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/9715d04413f296eaf3c30c47cec3daa6-Abstract.html.

Li, W., Feldman, M., Kazemi, E., and Karbasi, A. Submodular maximization in clean linear



Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.

time. In *Neural Information Processing Systems*, 2020. URL https://api.semanticscholar.org/CorpusID:247084420.

Liu, Q. C. and Srinivas, V. The predicted-updates dynamic model: Offline, incremental, and decremental to fully dynamic transformations. In Agrawal, S. and Roth, A. (eds.), *The Thirty Seventh Annual Conference on Learning Theory, June 30 - July 3, 2023, Edmonton, Canada*, volume 247 of *Proceedings of Machine Learning Research*, pp. 3582–3641. PMLR, 2024. URL https://proceedings.mlr.press/v247/liu24c.html.

Mirzasoleiman, B., Badanidiyuru, A., and Karbasi, A. Fast constrained submodular maximization: Personalized data summarization. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1358–1367, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/mirzasoleiman16.html.

Mirzasoleiman, B., Karbasi, A., and Krause, A. Deletion-robust submodular maximization: Data summarization with "the right to be forgotten". In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2449–2458. PMLR, 2017. URL http://proceedings.mlr.press/v70/mirzasoleiman17a.html.

Mirzasoleiman, B., Jegelka, S., and Krause, A. Streaming non-monotone submodular maximization: Personalized video summarization on the fly. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 1379–1386. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17014.

Mitzenmacher, M. and Vassilvitskii, S. Algorithms with predictions. *Commun. ACM*, 65(7):33–35, 2022. doi: 10.1145/3528087. URL https://doi.org/10.1145/3528087.

Monemizadeh, M. Dynamic submodular maximization. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6fbd841e2e4b2938351a4f9b68f12e6b-Abstract.html.

Oveis Gharan, S. and Vondrák, J. Submodular maximization by simulated annealing. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pp. 1098–1116. SIAM, 2011. doi: 10.1137/1.9781611973082.83. URL https://doi.org/10.1137/1.9781611973082.83.

Simon, I., Snavely, N., and Seitz, S. M. Scene summarization for online image collections. In *IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007*, pp. 1–8. IEEE Computer Society, 2007. doi: 10.1109/ICCV.2007.4408863. URL https://doi.org/10.1109/ICCV.2007.4408863.

Sipos, R., Swaminathan, A., Shivaswamy, P., and Joachims, T. Temporal corpus summarization using submodular word coverage. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pp. 754–763. ACM, 2012. doi: 10.1145/2396761.2396857. URL https://doi.org/10.1145/2396761.2396857.

Tohidi, E., Amiri, R., Coutino, M., Gesbert, D., Leus, G., and Karbasi, A. Submodularity in action: From machine learning to signal processing applications. *IEEE Signal Process. Mag.*, 37(5):120–133, 2020. doi: 10.1109/MSP.2020.3003836. URL https://doi.org/10.1109/MSP.2020.3003836.

Tschiatschek, S., Iyer, R. K., Wei, H., and Bilmes, J. A. Learning mixtures of submodular functions for image collection summarization. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/a8e864d04c95572d1aece099af852d0a-Paper.pdf.

van den Brand, J., Forster, S., Nazari, Y., and Polak, A. On dynamic graph algorithms with predictions. In Woodruff, D. P. (ed.), *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pp. 3534–3557. SIAM, 2024. doi: 10.1137/1.9781611977912.126. URL https://doi.org/10.1137/1.9781611977912.126.



Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.

## A. Proof of Lemma 17

*Proof.* Recall the Corollary 12, which states:

$$2\mathbb{E}\left[\, f(\mathrm{Sol}_2)\,\right] \geq \mathbb{E}\left[\, f(\mathcal{N}_1(\frac{1}{2}) \cup (\mathrm{OPT} \cap \mathcal{N}_2))\,\right] + \frac{1}{2}\mathbb{E}\left[\, f(\mathcal{N}_1(\frac{1}{2}))\,\right] + \frac{1}{2}\mathbb{E}\left[\, f(\mathcal{N}_1(\frac{1}{2}) \cup \mathcal{N}_2)\,\right] .$$

We replace each term in the right-hand side of this inequality with their repspective lower bound that we provided Lemmas 13, 15, and 16. We obtain the following:

$$\begin{aligned}
2\mathbb{E}\left[\, f(\mathrm{Sol}_2)\,\right] &\geq \mathbb{E}\left[\, f(\mathcal{N}_1(\frac{1}{2}) \cup (\mathrm{OPT} \cap \mathcal{N}_2))\,\right] + \frac{1}{2}\left(\, \mathbb{E}\left[\, (f(\mathcal{N}_1(\frac{1}{2}))\,\right] + \mathbb{E}\left[\, f(\mathcal{N}_1(\frac{1}{2}) \cup \mathcal{N}_2)\,\right]\,\right) \\
&\geq \frac{1}{4}\left(\, f(\mathrm{OPT} \cap \mathcal{N}_2) + f(\mathrm{OPT}) + f((\overline{\mathrm{OPT}} \cap \mathcal{N}_1) \cup (\mathrm{OPT} \cap \mathcal{N}_2)) + f(\mathcal{N}_1 \cup \mathrm{OPT})\,\right) \\
&+ \frac{1}{8}\left(\, f(\emptyset) + f(\mathrm{OPT} \cap \mathcal{N}_1) + f(\overline{\mathrm{OPT}} \cap \mathcal{N}_1) + f(\mathcal{N}_1)\,\right) \\
&+ \frac{1}{8}\left(\, f(\mathcal{N}_2) + f(\mathrm{OPT} \cup \mathcal{N}_2) + f(\overline{\mathrm{OPT}} \cup \mathcal{N}_2) + f(\mathcal{N}_1 \cup \mathcal{N}_2)\,\right) .
\end{aligned}$$

Next, we rearrange the terms and group them as follows:

$$\begin{aligned}
2\mathbb{E}\left[\, f(\mathrm{Sol}_2)\,\right] &\geq \frac{1}{4} f(\mathrm{OPT}) \\
&+ \frac{1}{8}\left(\, f(\mathrm{OPT} \cap \mathcal{N}_2) + f(\mathrm{OPT} \cap \mathcal{N}_1)\,\right) \\
&+ \frac{1}{8}\left(\, f(\mathcal{N}_1 \cup \mathrm{OPT}) + f(\mathrm{OPT} \cup \mathcal{N}_2)\,\right) \\
&+ \frac{1}{8}\left(\, f((\overline{\mathrm{OPT}} \cap \mathcal{N}_1) \cup (\mathrm{OPT} \cap \mathcal{N}_2)) + f(\mathcal{N}_2)\,\right) \\
&+ \frac{1}{8}\left(\, f(\mathrm{OPT} \cap \mathcal{N}_2) + f((\overline{\mathrm{OPT}} \cap \mathcal{N}_1) \cup (\mathrm{OPT} \cap \mathcal{N}_2)) + f(\mathcal{N}_1 \cup \mathrm{OPT})\,\right) \\
&+ \frac{1}{8}\left(\, f(\emptyset) + f(\overline{\mathrm{OPT}} \cap \mathcal{N}_1) + f(\mathcal{N}_1) + f(\overline{\mathrm{OPT}} \cup \mathcal{N}_2) + f(\mathcal{N}_1 \cup \mathcal{N}_2)\,\right) .
\end{aligned}$$

The following inequalities hold because of the submodularity of the function $f$:

$$\begin{aligned}
f(\mathrm{OPT} \cap \mathcal{N}_2) + f(\mathrm{OPT} \cap \mathcal{N}_1) &\geq f((\mathrm{OPT} \cap \mathcal{N}_2) \cup (\mathrm{OPT} \cap \mathcal{N}_1)) + f((\mathrm{OPT} \cap \mathcal{N}_2) \cap (\mathrm{OPT} \cap \mathcal{N}_1)) \\
&= f(\mathrm{OPT}) + f(\emptyset).
\end{aligned}$$

$$\begin{aligned}
f(\mathcal{N}_1 \cup \mathrm{OPT}) + f(\mathrm{OPT} \cup \mathcal{N}_2) &\geq f((\mathcal{N}_1 \cup \mathrm{OPT}) \cup (\mathrm{OPT} \cup \mathcal{N}_2)) + f((\mathcal{N}_1 \cup \mathrm{OPT}) \cap (\mathrm{OPT} \cup \mathcal{N}_2)) \\
&= f(\mathcal{N}_1 \cup \mathcal{N}_2) + f(\mathrm{OPT}).
\end{aligned}$$

$$\begin{aligned}
f((\overline{\mathrm{OPT}} \cap \mathcal{N}_1) \cup (\mathrm{OPT} \cap \mathcal{N}_2)) + f(\mathcal{N}_2) &\geq f(((\overline{\mathrm{OPT}} \cap \mathcal{N}_1) \cup (\mathrm{OPT} \cap \mathcal{N}_2)) \cup f(\mathcal{N}_2)) \\
&+ f(((\overline{\mathrm{OPT}} \cap \mathcal{N}_1) \cup (\mathrm{OPT} \cap \mathcal{N}_2)) \cap f(\mathcal{N}_2)) \\
&= f(\overline{\mathrm{OPT}} \cup \mathcal{N}_2) + f(\mathrm{OPT} \cap \mathcal{N}_2).
\end{aligned}$$



Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.

Incorporating these three inequalities in our previous bound, gives us:

$$
\begin{aligned}
2\mathbb{E}\left[\, f(\mathrm{Sol}_2)\,\right] & && (1)\\
&\geq \frac{1}{4} f(\mathrm{OPT}) && (2)\\
&+ \frac{1}{8}\left(\, f(\mathrm{OPT}) + f(\emptyset)\,\right) && (3)\\
&+ \frac{1}{8}\left(\, f(\mathcal{N}_1 \cup \mathcal{N}_2) + f(\mathrm{OPT})\,\right) && (4)\\
&+ \frac{1}{8}\left(\, f(\overline{\mathrm{OPT}} \cup \mathcal{N}_2) + f(\mathrm{OPT} \cap \mathcal{N}_2)\,\right) && (5)\\
&+ \frac{1}{8}\left(\, f(\mathrm{OPT} \cap \mathcal{N}_2) + f((\overline{\mathrm{OPT}} \cap \mathcal{N}_1) \cup (\mathrm{OPT} \cap \mathcal{N}_2)) + f(\mathcal{N}_1 \cup \mathrm{OPT})\,\right) && (6)\\
&+ \frac{1}{8}\left(\, f(\emptyset) + f(\overline{\mathrm{OPT}} \cap \mathcal{N}_1) + f(\mathcal{N}_1) + f(\overline{\mathrm{OPT}} \cup \mathcal{N}_2) + f(\mathcal{N}_1 \cup \mathcal{N}_2)\,\right), && (7)
\end{aligned}
$$

which is equivalent to the inequality stated in the Lemma, and completes the proof. □

## B. Unconstrained Dynamic Algorithm with Insertions and Deletions

In this section, we develop a dynamic algorithm for unconstrained submodular maximization in a setting that includes both element insertions and deletions. We first present a deletion-only dynamic algorithm, building on concepts from our incremental approach, which serves as a foundation for our subsequent algorithm.

### B.1. Decremental Dynamic Algorithm for USM with Known-Order Deletions

We design a dynamic algorithm that addresses the problem of maximizing a submodular function without any constraints, in a decremental setting where the order of deletions is known in advance. This algorithm operates similarly to our incremental algorithm, but in reverse order.

#### B.1.1. Setting

The decremental dynamic setting refers to a framework in which the problem's input is subject to decremental updates. Specifically, the algorithm starts with an initial input and processes a stream of updates consisting solely of element deletions. The objective is to maintain a near-optimal solution while minimizing the computational cost of handling each deletion, rather than recomputing the solution from scratch after every change. We also assume that the initial elements provided to the algorithm are sorted according to their deletion order. Specifically, the algorithm receives an initial list of elements $[e_1, e_2, \ldots, e_n]$, and after $t$ deletions, it must return an approximate solution to the problem of finding $\mathrm{OPT}_t = \arg\max_{A \subseteq V_t} f(A)$, where $V_t := \{e_{t+1}, \ldots, e_n\}$.

#### B.1.2. Algorithm Description

In this algorithm (Algorithm 3), we maintain two sets $\mathcal{M}_1$ and $\mathcal{M}_2$ to store all the remaining elements subject to deletion, two solution sets $\mathrm{Sol}'_1$ and $\mathrm{Sol}'_2$, and the set $S_2$ that would always store a random subset of $\mathcal{M}_2$. First, we initialize $\mathcal{M}_2$ with the input set $V$, while initializing all other sets as empty. To process each deletion, we proceed according to the following structure:

We check, and if the set $\mathcal{M}_1$ is empty, we fill it with the first $\lfloor \sqrt{n} \rfloor$ elements of $\mathcal{M}_2$ and then update $\mathcal{M}_2$ by removing those elements. At this point, we also recompute $\mathrm{Sol}'_2$ using Extend on the updated set $\mathcal{M}_2$ and update $S_2$ by randomly sampling elements from the updated $\mathcal{M}_2$, where each element of $\mathcal{M}_2$ is in $S_2$ with a probability of $\frac{1}{2}$.

Once ensured $\mathcal{M}_1$ is not empty, we remove its first element. We then update the solution $\mathrm{Sol}'_1$ by executing Extend given the current (and possibly modified) set $S_2$ and the modified set $\mathcal{M}_1$.

Finally, we return the best solution between $\mathrm{Sol}'_1$ and $\mathrm{Sol}'_2$, maximizing the submodular function value.

In this algorithm, $\mathcal{M}_1$ holds the elements ready for deletion, while $\mathcal{M}_2$ contains the elements yet to be processed. At any given time, the union of the sets $\mathcal{M}_1$ and $\mathcal{M}_2$ represents all remaining elements. The solution sets $\mathrm{Sol}'_1$ and $\mathrm{Sol}'_2$ provide



Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.

approximate solutions based on elements from their respective sets and $\mathcal{M}_2$, ensuring an approximation guarantee for the algorithm based on the current partitioning of the optimal solution across these two sets.

**Algorithm 3** dynamic known order deletion only

**function** INIT($V$)
  $\mathcal{M}_2 = V$
  $\mathcal{M}_1, S_2, \text{Sol}'_1, \text{Sol}'_2 \leftarrow \emptyset$
  **return** Extend($\emptyset, \mathcal{M}_2$)
**function** DELETE()
  **if** $|\mathcal{M}_1| = 0$ **then**
    $\mathcal{M}_1 \leftarrow \mathcal{M}_2[:\sqrt{n}]$
    $\mathcal{M}_2 \leftarrow \mathcal{M}_2[\sqrt{n}:]$
    $\text{Sol}'_2 \leftarrow \text{Extend}(\emptyset, \mathcal{M}_2)$
    $S_2 \leftarrow \mathcal{M}_2(\frac{1}{2})$
  $\mathcal{M}_1.pop(0)$
  $\text{Sol}'_1 \leftarrow \text{Extend}(S_2, \mathcal{M}_1)$
  **return** $\arg\max\{f(\text{Sol}'_1), f(\text{Sol}'_2)\}$

### B.1.3. ANALYSIS

In this section, we establish the following theorem by demonstrating the approximation guarantee and query complexity of Algorithm 3.

**Theorem 21.** *There exists an algorithm for the USM problem in the decremental dynamic setting described in Section B.1.1 that achieves a* $0.3$ *approximation with an amortized query complexity of* $O(\sqrt{n})$ *per update.*

*Proof.* We show that Algorithm 3 maintains a $0.3$-approximate solution for the unconstrained submodular maximization problem in Lemma 22. The amortized query complexity is $O(\sqrt{n})$ per update which can be proven similar to Lemma 20. This concludes the proof. □

**Lemma 22** (Approximation Guarantee)**.** *Consider a fixed time* $t$*. Let* $V_t$ *represent the set of elements remaining after the first* $t$ *updates, and let* $OPT_t$ *denote the optimal subset of* $V_t$*. The output of Algorithm 3 following update* $t$*, denoted as* $Sol_t$*, provides a* $0.3$*-approximation of* $OPT_t$*. More specifically, we have:*

$$\mathbb{E}\left[\, f(Sol_t) \,\right] = \mathbb{E}\left[\, \max(f(Sol_1), f(Sol_2)) \,\right] \geq 0.3 f(OPT_t).$$

*Proof.* The proof of the correctness of Algorithm 3 closely follows the arguments presented for Algorithm 2. To avoid excessive repetition, we will only outline its structure instead. □

As before, at any point during the execution of our algorithm, the elements of $V_t$ are divided between the two sets $\mathcal{N}_1$ and $\mathcal{N}_2$. Consequently, $\text{OPT}_t$ and $\overline{\text{OPT}}$ will maintain the same definitions as previously established.

**Lemma 23.** *For a fixed set* $S_2$*, the following inequality holds:*

$$f(OPT(S_2, \mathcal{M}_1)) + \frac{1}{2}\left(f(S_2) + f(S_2 \cup \mathcal{M}_1)\right) \leq 2\mathbb{E}\left[\, f(Sol'_1) \,\right].$$

**Lemma 24.** *For a fixed set* $S_2$*, the following inequality holds:*

$$2\mathbb{E}\left[\, f(Sol'_1) \,\right] \geq f(S_2 \cup (OPT \cap \mathcal{M}_1)) + \frac{1}{2}(f(S_2) + f(S_2 \cup \mathcal{M}_1)).$$

**Corollary 25.** *We can assert:*

$$2\mathbb{E}\left[\, f(Sol'_1) \,\right] \geq \mathbb{E}\left[ f(\mathcal{M}_2(\frac{1}{2}) \cup (OPT \cap \mathcal{M}_1)) \right] + \frac{1}{2}\mathbb{E}\left[ f(\mathcal{M}_2(\frac{1}{2})) \right] + \frac{1}{2}\mathbb{E}\left[ f(\mathcal{M}_2(\frac{1}{2}) \cup \mathcal{M}_1) \right].$$



Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.

**Lemma 26.** $\mathbb{E}\left[\, f(\mathcal{M}_2(\frac{1}{2}))\,\right] \geq \frac{1}{4}\left(f(\emptyset) + f(\mathit{OPT} \cap \mathcal{M}_2) + f(\overline{\mathit{OPT}} \cap \mathcal{M}_2) + f(\mathcal{M}_2)\right).$

**Lemma 27.** $\mathbb{E}\left[\, f(\mathcal{M}_2(\frac{1}{2}) \cup (\mathit{OPT} \cap \mathcal{M}_1))\,\right]$ *is lower-bounded by*

$$\frac{1}{4}\left(\, f(\mathit{OPT} \cap \mathcal{M}_1) + f(\mathit{OPT}) + f((\overline{\mathit{OPT}} \cap \mathcal{M}_2) \cup (\mathit{OPT} \cap \mathcal{M}_1)) + f(\mathcal{M}_2 \cup \mathit{OPT})\,\right).$$

**Lemma 28.** $\mathbb{E}\left[\, f(\mathcal{M}_2(\frac{1}{2}) \cup \mathcal{M}_1)\,\right] \geq \frac{1}{4}\left(\, f(\mathcal{M}_1) + f(\mathit{OPT} \cup \mathcal{M}_1) + f(\overline{\mathit{OPT}} \cup \mathcal{M}_1) + f(\mathcal{M}_2 \cup \mathcal{M}_1)\,\right)$

**Lemma 29.** *The following inequality holds:*

$$\begin{aligned} 2\mathbb{E}\left[\, f(\mathit{Sol}'_1)\,\right] \geq\ & \frac{1}{2}f(\mathit{OPT}) + \frac{1}{4}f(\mathit{OPT} \cap \mathcal{M}_1) + \frac{1}{8}f((\overline{\mathit{OPT}} \cap \mathcal{M}_2) \cup (\mathit{OPT} \cap \mathcal{M}_1)) + \frac{1}{8}f(\mathcal{M}_2 \cup \mathit{OPT}) \\ & + \frac{1}{4}f(\emptyset) + \frac{1}{8}f(\overline{\mathit{OPT}} \cap \mathcal{M}_2) + \frac{1}{8}f(\mathcal{M}_2) + \frac{1}{4}f(\overline{\mathit{OPT}} \cup \mathcal{M}_1) + \frac{1}{4}f(\mathcal{M}_2 \cup \mathcal{M}_1). \end{aligned}$$

**Corollary 30.** *The following inequality holds:*

$$2\mathbb{E}\left[\, f(\mathit{Sol}'_1)\,\right] \geq \frac{1}{2}f(\mathit{OPT}) + \frac{1}{4}f(\mathit{OPT} \cap \mathcal{M}_1).$$

**Lemma 31.** *The maximum expected submodular value of $\mathit{Sol}'_1$ and $\mathit{Sol}'_2$ provides a $\frac{3}{10}$-approximation of $f(\mathit{OPT})$. Specifically,*

$$\max\left(\mathbb{E}\left[\, f(\mathit{Sol}'_1)\,\right], \mathbb{E}\left[\, f(\mathit{Sol}'_2)\,\right]\right) \geq \frac{3}{10}f(\mathit{OPT}).$$

### B.2. Fully Dynamic Algorithm for USM with Deletion Times

We proceed to present an unconstrained dynamic in a setting that accommodates both insertions and deletions.

#### B.2.1. Setting

The term "fully dynamic algorithm" denotes frameworks where the problem's input undergoes both types of updates: element insertions and element deletions. Let $\mathcal{S}$ be the stream of insertion and deletion updates to the problem's input, determined by an adversary. In this model, we assume that the deletion time of each element is revealed to the algorithm at the time of its insertion.

Specifically, at each time step, our algorithm receives an update in the form of either an element insertion or an element deletion. If an element $e$ is inserted into the set, the algorithm also immediately learns the time at which $e$ will be deleted.

Note, however, that the algorithm uses the deletion times solely to determine the order of element removals. It can also support various other settings, such as those where elements have a predetermined lifespan, First-in-First-out (FIFO) or Last-in-First-out (LIFO) orders, or settings where removals are based on a form of ranking. Additionally, with further adjustments, the algorithm could be adapted to handle settings with error tolerance boundaries or where statistical accuracy is sufficient.

Similar to the previous settings, The objective is to maintain an approximate solution to the problem of finding $\text{OPT}_t = \arg\max_{A \subseteq V_t} f(A)$, where $V_t$ is the set of elements currently in the input after processing all updates up to time $t$. The performance of our fully dynamic algorithm is evaluated based on the approximation ratio of the maintained solution and the computational complexity of each update operation.

#### B.2.2. Algorithm Description

Algorithm 4 maintains two sets, $\mathcal{N}_1$ and $\mathcal{N}_2$, to store elements. The set $\mathcal{N}_2$ acts as a temporary buffer for recently inserted elements, while $\mathcal{N}_1$ serves as a more permanent storage for older elements. The algorithm also keeps an instance of our decremental algorithm, which handles deletions from the elements stored in $\mathcal{N}_1$.

Similar to our incremental algorithm, the algorithm maintains a set $S_1$, which is always a subset of $\mathcal{N}_1$, with each element included with a probability of $\frac{1}{2}$. Also similar to our incremental algorithm, after each update (both insertions and deletions) we execute the Extend algorithm on input sets $S_1$ and $\mathcal{N}_2$, and store its output in $\text{Sol}_2$. The solution set $\text{Sol}_1$, however, always holds the latest output generated by the decremental algorithm instance $\mathcal{I}$.

The algorithm processes a series of updates over $T$ time steps, where each update is either an element insertion or deletion:



Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.

- **Element Insertions**: These are handled similarly to the incremental algorithm, with one key difference: when the buffer $\mathcal{N}_2$ becomes full and its elements are transferred to $\mathcal{N}_1$, the elements in $\mathcal{N}_1$ are sorted by their predicted removal times, and a new instance of the decremental algorithm is created.

- **Element Deletions**: These are managed differently:
  - If the element being deleted is in the buffer $\mathcal{N}_2$, it is simply removed from $\mathcal{N}_2$.
  - If the element is in $\mathcal{N}_1$, it is removed from both $\mathcal{N}_1$ and $S_1$ (if applicable), and the decremental algorithm instance $\mathcal{I}$ is invoked to handle the deletion.

At the end of each update step, the algorithm returns the best solution between $\text{Sol}_1$ and $\text{Sol}_2$.

**Algorithm 4** Fully dynamic algorithm with predictions

**function** INIT
  $\mathcal{N}_1, \mathcal{N}_2, S_1, \text{Sol}_1, \text{Sol}_2 \leftarrow \emptyset$
  Let $\mathcal{I}$ be an instance of our decremental algorithm
  **for** $t = 1$ to $T$ **do**
    **if** $\text{Update}_i = \text{INSERT}(e)$ **then**
      $\mathcal{N}_2 \leftarrow \mathcal{N}_2 \cup \{e\}$
      **if** $|\mathcal{N}_2| = \lfloor \sqrt{n} \rfloor$ **then**
        $\mathcal{N}_1 \leftarrow \mathcal{N}_1 \cup \mathcal{N}_2, \mathcal{N}_2 \leftarrow \emptyset$
        Sort the elements in $\mathcal{N}_1$ based on their removal time
        $\text{Sol}_1 \leftarrow \mathcal{I}.\text{INIT}(\mathcal{N}_1)$
        $S_1 \leftarrow \mathcal{N}_1(\frac{1}{2})$
      $\text{Sol}_2 \leftarrow \text{Extend}(S_1, \mathcal{N}_2)$
      **yield** $\arg\max\{f(\text{Sol}_1), f(\text{Sol}_2)\}$
    **if** $\text{Update}_i = \text{DELETE}(e)$ **then**
      **if** $e \in \mathcal{N}_2$ **then**
        $\mathcal{N}_2 \leftarrow \mathcal{N}_2 \backslash \{e\}$
      **else**
        $\text{Sol}_2 \leftarrow \mathcal{I}.\text{DELETE}()$
        $\mathcal{N}_1 \leftarrow \mathcal{N}_1 \backslash \{e\}$
        $S_1 \leftarrow S_1 \backslash \{e\}$
      $\text{Sol}_2 \leftarrow \text{Extend}(S_1, \mathcal{N}_2)$
      **yield** $\arg\max\{f(\text{Sol}_1), f(\text{Sol}_2)\}$

### B.2.3. ANALYSIS

In this section, we establish the following theorem by demonstrating the approximation guarantee and query complexity of Algorithm 4.

**Theorem 32.** *There exists an algorithm for the USM problem in a fully dynamic setting described in Section B.2.1 that achieves a* $0.264$ *approximation with an amortized query complexity of* $O(\sqrt{n})$ *per update.*

*Proof.* We show that Algorithm 4 maintains a $0.264$-approximate solution for the unconstrained submodular maximization problem in Lemma 22. The amortized query complexity is $O(\sqrt{n})$ per update, which can be proven similarly to Lemma 20. This concludes the proof. □

**Lemma 33** (Approximation Guarantee)**.** *Consider a fixed time* $t$*. Let* $V_t$ *represent the set of all present elements after the first* $t$ *updates, and let* $OPT_t$ *denote the optimal subset of* $V_t$*. The output of Algorithm 4 following update* $t$*, denoted as* $Sol_t$*, provides a* $0.264$*-approximation of* $OPT_t$*. More specifically, we have:*

$$\mathbb{E}[\, f(Sol_t)\,] = \mathbb{E}[\, \max(f(Sol_1), f(Sol_2))\,] \geq 0.264 f(OPT_t).$$



Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.

*Proof.* As explained in the description of the algorithm, exactly similar to the decremental algorithm, Algorithm 4 maintains a set $S_1$, which is always a subset of $\mathcal{N}_1$, with each element included with a probability of $\frac{1}{2}$. After any kind of update, it executes the EXTEND algorithm on input sets $S_1$ and $\mathcal{N}_2$, storing its output in $\text{Sol}_2$. Hence, Lemmas 9 through Lemma 17 all hold exactly the same for this algorithm as well. Thus, to prove the approximation guarantee of this algorithm, we only provide the modified version of Lemma 19. □

**Lemma 34.** *The maximum expected submodular value of $Sol_1$ and $Sol_2$ provides a $\frac{9}{34}$-approximation of $f(OPT)$. Specifically,*

$$\max\left(\mathbb{E}\left[\, f(Sol_1)\,\right], \mathbb{E}\left[\, f(Sol_2)\,\right]\right) \geq \frac{9}{34} f(OPT).$$

*Proof.* We proceed by analyzing two complementary cases.

**Case 1:** $f(\text{OPT} \cap \mathcal{N}_1) \geq \frac{30}{34} f(\text{OPT})$.

In this case, we show that $\mathbb{E}\left[\, f(\text{Sol}_1)\,\right] \geq \frac{90}{340} f(\text{OPT})$. Recall that in Algorithm 4, $\text{Sol}_1$ is the output of our decremental algorithm with an approximation ratio of $\frac{3}{10}$, and the optimal solution in $\mathcal{N}_1$ is at least as good as $\text{OPT} \cap \mathcal{N}_1$.

**Case 2:** $f(\text{OPT} \cap \mathcal{N}_1) < \frac{30}{34} f(\text{OPT})$.

By the submodularity property of the function $f$, we have:

$$f(\text{OPT}) + f(\emptyset) \leq f(\text{OPT} \cap \mathcal{N}_1) + f(\text{OPT} \cap \mathcal{N}_2) < \frac{30}{34} f(\text{OPT}) + f(\text{OPT} \cap \mathcal{N}_2).$$

Non-negativity of $f$ implies:

$$f(\text{OPT} \cap \mathcal{N}_2) > \frac{4}{34} f(\text{OPT}) + f(\emptyset) \geq \frac{4}{34} f(\text{OPT}).$$

Next, by Corollary 18, we have:

$$2\mathbb{E}\left[\, f(\text{Sol}_2)\,\right] \geq \frac{1}{2} f(\text{OPT}) + \frac{1}{4} f(\text{OPT} \cap \mathcal{N}_2).$$

Substituting the lower bound on $f(\text{OPT} \cap \mathcal{N}_2)$ yields:

$$2\mathbb{E}\left[\, f(\text{Sol}_2)\,\right] > \left(\frac{1}{2} + \frac{1}{4} \times \frac{4}{34}\right) f(\text{OPT}),$$

which simplifies to:

$$\mathbb{E}\left[\, f(\text{Sol}_2)\,\right] > \frac{9}{34} f(\text{OPT}).$$

Thus, in both cases, we have:

$$\max\left(\mathbb{E}\left[\, f(\text{Sol}_1)\,\right], \mathbb{E}\left[\, f(\text{Sol}_2)\,\right]\right) \geq \frac{9}{34} f(\text{OPT}),$$

completing the proof. □



Confidential reviewer copy. This manuscript is under double-blind review by ICML 2026. Unauthorized sharing, redistribution, or disclosure is strictly prohibited.