When Streaming Fails: Dynamic Algorithms for Unconstrained Submodular Maximization

Kiarash Banihashem[*]
University of Maryland
College Park, MD, USA
kiarash@umd.edu

Samira Goudarzi[*]
University of Maryland
College Park, MD, USA
samirag@umd.edu

MohammadTaghi Hajiaghayi[*]
University of Maryland
College Park, MD, USA
hajiagha@umd.edu

Peyman Jabbarzade[*]
University of Maryland
College Park, MD, USA
peymanj@umd.edu

Morteza Monemizadeh[*]
TU Eindhoven
Eindhoven, The Netherlands
M.Monemizadeh@tue.nl

Abstract

Dynamic submodular optimization has attracted significant attention in recent years, with a growing body of work studying both monotone and non-monotone variants under various constraints. A key insight underlying much of this progress is that the dynamic setting is closely related to the streaming setting, both in terms of algorithms and lower bounds. This raises a natural and fundamental question: *can we obtain dynamic algorithms even for problems where no streaming analogue exists?* In this paper, we answer this question affirmatively for the fundamental problem of Unconstrained Submodular Maximization (USM). Unlike other variants of submodular maximization, no streaming algorithm with non-trivial guarantees is known for USM. In the dynamic setting, a trivial 0.25-approximation follows from random sampling, but obtaining any better approximation with sublinear update time had remained open. We present the first dynamic algorithms for USM that break the 0.25-approximation barrier while maintaining sublinear update time, across several natural dynamic models: incremental updates, decremental updates with known deletion order, and fully dynamic updates with known deletion times.

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 [Fuj84, TAC+20]. Given a ground set $\mathcal{V}$, a function $f : 2^{\mathcal{V}} \to \mathbb{R}_{\geq 0}$ over $\mathcal{V}$ is called *submodular* if all $B \subseteq A \subseteq \mathcal{V}$ and $e \in \mathcal{V} \setminus A$ satisfy $f(A \cup \{e\}) - f(A) \leq f(B \cup \{e\}) - f(B)$. The function is called *monotone* if it further satisfies $f(B) \leq f(A)$ for all $A \subseteq B$. Given the ubiquity of the diminishing returns property, submodular function optimization has found applications in core machine learning tasks such as data summarization [SSS07, TIWB14, SSSJ12], feature selection [KED+17, DK08, DK18], and personalized recommendation [EG11]. More broadly, submodular functions appear naturally in information-theoretic objectives [GKS05], graph-structured problems [BBG+23], and across economics, game theory, and operations research.

Motivated by the prevalence of large, continuously evolving real-world datasets, a recent body of work has studied *dynamic* submodular optimization, where the ground set undergoes a sequence of element insertions and deletions. This line of work was initiated by the independent works of Monemizadeh [Mon20] and Lattanzi et al. [LMN+20], who obtained $\frac{1}{2}$-approximation algorithms for monotone submodular maximization under a cardinality constraint. Subsequent works generalized

[*]Equal Contribution

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

these results to richer settings, including non-monotone functions[2] [BBG+23], and constraints such as matroids [BBG+24, DFL+23, PR23, GZZ26], matchings [BBG+25], and matchoids [BGH+25]. Lower bounds for dynamic submodular maximization have also been established [CP22].

A key observation underlying much of this progress is that the dynamic setting is intimately connected to the streaming setting. Concretely, the original algorithms of Monemizadeh [Mon20] and Lattanzi et al. [LMN+20] build directly on the classical thresholding streaming algorithm of Badanidiyuru et al. [BMKK14], which admits any element whose marginal gain exceeds $\mathrm{OPT}/2k$. This algorithmic connection is not isolated to the initial works: subsequent results for richer constraint families similarly adapt their streaming counterparts to the dynamic setting. The connection to streaming runs deeper still, extending to lower bounds: the hardness result of Chen and Peng [CP22], which rules out a $(\frac{1}{2}+\epsilon)$-approximation for monotone submodular maximization under a cardinality constraint in the dynamic setting, builds on the streaming lower bound of Feldman et al. [FNSZ20]. This raises a natural question: *can one obtain dynamic algorithms for submodular maximization problems where no streaming analogue exists?*

We answer this affirmatively for *Unconstrained Submodular Maximization* (USM), where given $f : 2^{\mathcal{V}} \to \mathbb{R}_{\geq 0}$, the goal is to find $S^* \subseteq \mathcal{V}$ maximizing $f(S^*)$ with no constraints on its structure or size. Unlike the constrained variants above, no streaming algorithm with non-trivial guarantees exists for USM[3], making the streaming-to-dynamic blueprint entirely unavailable. In the dynamic setting, a trivial 0.25-approximation follows from the random sampling algorithm of [FMV11], but obtaining any better approximation with sublinear update time had remained open. Our main result is the following.

Theorem 1 (Informal)**.** *There exist dynamic algorithms for USM with $O(\sqrt{n})$ amortized query time per update that achieve a 0.3-approximation in the incremental and decremental (with known deletion order) settings, and a 0.264-approximation in the fully dynamic setting with known deletion times.*

USM has broad applications, capturing combinatorial optimization problems such as Max-Cut and Max Directed Cut [FG95, GW95, HZ01, Hås01, KOKM04], Submodular Max-SAT, Submodular Welfare problems, Maximum Facility Location [AS99, CFN77], and Generalized Assignment [CK05, CKR06, FV06].

USM is an NP-hard problem, which has been extensively studied over the years. However, Feige, Mirrokni, and Vondrák [FMV11] 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 [OGV11] and [FNS11] 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 [BFNS15] (Test-of-Time award, FOCS'22) 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 [FMV11]. Their result fully addressed the approximability of the offline USM problem by polynomial time algorithms. Later, Buchbinder and Feldman [BF15] 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, [LFKK20] proved that achieving an approximation ratio of $0.25+\epsilon$ for the USM problem requires $\Omega(\frac{n}{\log n})$ oracle queries.

[2] A non-monotone submodular function is one that is not necessarily monotone. Since monotonicity provides additional structural properties that algorithms can exploit, the non-monotone setting is strictly harder. The term "non-monotone" is used to emphasize that no such properties are assumed, not that the function is guaranteed to violate monotonicity.

[3] Streaming algorithms must operate in sublinear memory. However, even for the simple monotone function $f(S) = |S|$, the optimal solution is the entire ground set $\mathcal{V}$, so any constant-approximate solution has size $\Omega(|\mathcal{V}|)$, requiring linear memory. Hence no meaningful streaming result is possible.

1.1 Our Techniques

We first establish our result for a *dynamic incremental model*, where given an arbitrary 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:

1) Random sampling: As mentioned above, Feige, Mirrokni, and Vondrák [FMV11] 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 [BFNS15] 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 [BFNS15] to expand the sample set S_1 using elements of $\mathcal{N}_2$. This yields a solution set, denoted as 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 [BFNS15] to $\mathcal{N}_1$ to update the solution set Sol_1, and resample S_1. Since we invoke the linear-time algorithm [BFNS15] 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, Sol_1 and 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 Sol_1 for all elements seen so far. As long as Sol_1 achieves at least a 0.3-approximation of the optimal value for $\mathcal{N}_1 \cup \mathcal{N}_2$, 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 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 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 A.

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 [HKN18], single-source reachability on directed graphs [HKN15], matching [ABD22], approximate min-cost flow [BGS21], and strongly-connected components [BPW19], 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 [MV22] and subsequently was studied for other problems, including online matrix vector [HSSY24], transitive closure, triangle detection, single-source reachability [vdBFNP24], and planar digraph all pairs shortest paths, k-edge connectivity, DFS tree [LS24], submodular maximization under cardinality constraint [LS24, AB23].

1.2 Related Work

The offline and dynamic settings 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 model.

We first review prior results on streaming algorithms for submodular maximization. For monotone submodular maximization, Badanidiyuru et al. [BMKK14] introduced an insertion-only streaming algorithm achieving a $(0.5-\epsilon)$-approximation under a cardinality constraint k. Chekuri, Gupta, and Quanrud [CGQ15] extended the streaming framework to both monotone and non-monotone submodular functions subject to p-matchoid constraints. Subsequent work by Mirzasoleiman et al. [MJK18] and Feldman et al. [FKK18] 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. [AEF$^+$20], who achieved a $(0.277+\epsilon)$-approximation, improving upon the previous 0.17-approximation of Feldman et al. [FKK18]. We also note that there exist streaming algorithms for monotone submodular maximization that support deletions [KZK18, MKK17]. 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.

1.3 Preliminaries

Definition 2 (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: $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$. 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 3 (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 [LMN$^+$20, Mon20, CP22, DFL$^+$23], 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 $f(A)$.

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 $\mathcal{V}$. The time t corresponds to the t^{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 $OPT_t = \max_{S \subseteq V_t} f(S)$.

Definition 4 (Query complexity)**.** *The query complexity of a dynamic α-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 [CW77]. 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.

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 [BFNS12] 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.

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

Definition 5 (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*

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

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 6. *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:*

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

Proof. The proof is exactly similar to the proof of RANDOMIZEDUSM [BFNS12]. 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 $\mathcal{V} \supseteq \mathcal{N}$. □

Algorithm 1 Generalized USM

function EXTEND($S, \mathcal{N}$)
  $X_0 \leftarrow S,\ Y_0 \leftarrow S \cup \mathcal{N}$
  Let $\{s_1, s_2, \ldots, s_{|\mathcal{N}|}\} \leftarrow \mathcal{N}$
  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 $Y_{|\mathcal{N}|}$)

*If $a'_i + b'_i = 0$, we assume that $a'_i/(a'_i + b'_i) = 1$.

Algorithm 2 DYNAMICINSERTION

function MAIN()
  $\mathcal{N}_1, \mathcal{N}_2, S_1, \mathrm{Sol}_1, \mathrm{Sol}_2 \leftarrow \emptyset$
  for each element e in the stream **do**
    yield INSERT(e)
function INSERT(e)
  $\mathcal{N}_2 \leftarrow \mathcal{N}_2 \cup \{e\}$
  if $|\mathcal{N}_2| = \lfloor \sqrt{n} \rfloor$ **then**
    $\mathcal{N}_1 \leftarrow \mathcal{N}_1 \cup \mathcal{N}_2, \mathcal{N}_2 \leftarrow \emptyset$
    $\mathrm{Sol}_1 \leftarrow$ EXTEND($\emptyset, \mathcal{N}_1$)
    $S_1 \leftarrow \mathcal{N}_1(\frac{1}{2})$
  $\mathrm{Sol}_2 \leftarrow$ EXTEND($S_1, \mathcal{N}_2$)
  return $\arg\max\{f(\mathrm{Sol}_1), f(\mathrm{Sol}_2)\}$

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 Sol_1 and 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 element 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 Sol_1 accordingly, such that the following 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 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 Sol_1, we update Sol_2 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 Sol_1 and Sol_2.

2.2.3 Analysis

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

Theorem 7. *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 8, and we also prove that its amortized query complexity is $O(\sqrt{n})$ per update in Lemma 21. This completes the proof. □

Lemma 8 (Approximation Guarantee)**.** *Consider any fixed time t. Let V_t denote the set of elements inserted during the first t updates, and let OPT_t be the optimal subset of V_t. The output of Algorithm 2 after update t, denoted by Sol_t, provides a 0.3-approximation of OPT_t. Specifically, we have:*

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

Proof. We begin by fixing the time t and will establish a series of lemmas that culminate in Lemma 20.

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

According to Lemma 20, we have: $\max\left(\mathbb{E}\left[f(\text{Sol}_1)\right], \mathbb{E}\left[f(\text{Sol}_2)\right]\right) \geq \frac{3}{10} f(\text{OPT})$, which implies, by Jensen's inequality, that $\mathbb{E}\left[\max\left(f(\text{Sol}_1), f(\text{Sol}_2)\right)\right] \geq \frac{3}{10} f(\text{OPT})$. This completes the proof. □

Lemma 10. *For a fixed set S_1, the following inequality holds:*

$$f(OPT(S_1, \mathcal{N}_2)) + \frac{1}{2}\left(f(S_1) + f(S_1 \cup \mathcal{N}_2)\right) \leq 2\mathbb{E}\left[f(Sol_2)\right].$$

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 Sol_2 right before returning its solution.

Also, by Lemma 6, 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 $\mathbb{E}\left[f(\text{Sol}_2)\right]$ only accounts for the randomness of the function EXTEND, not the randomness of the set S_1.

□

Lemma 11. *For a fixed set S_1, the following inequality holds:* $2\mathbb{E}\left[f(Sol_2)\right] \geq f(S_1 \cup (OPT \cap \mathcal{N}_2)) + \frac{1}{2}(f(S_1) + f(S_1 \cup \mathcal{N}_2))$

Proof. Definition 5, defines $\text{OPT}(S_1, \mathcal{N}_2)$ 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: $f(\text{OPT}(S_1, \mathcal{N}_2)) \geq f(A)$. Since $S_1 \subseteq S_1 \cup (\text{OPT} \cap \mathcal{N}_2)$ and $S_1 \cup (\text{OPT} \cap \mathcal{N}_2) \subseteq S_1 \cup \mathcal{N}_2$, we get: $f(\text{OPT}(S_1, \mathcal{N}_2)) \geq f(S_1 \cup (\text{OPT} \cap \mathcal{N}_2))$. Additionally, by Lemma 10, we have: $2\mathbb{E}\left[f(\text{Sol}_2)\right] \geq f(\text{OPT}(S_1, \mathcal{N}_2)) + \frac{1}{2}\left(f(S_1) + f(S_1 \cup \mathcal{N}_2)\right)$. 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 [FMV11], which we restate below for convenience.

Lemma 12 (Lemma 2.3 of [FMV11])**.** *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* $\mathbb{E}\left[f(A(p) \cup B(q))\right] \geq (1-p)(1-q)f(\emptyset) + p(1-q)f(A) + (1-p)qf(B) + pqf(A \cup B)$.

Corollary 13. *We have:* $2\mathbb{E}\left[f(Sol_2)\right] \geq \mathbb{E}\left[f(\mathcal{N}_1(\frac{1}{2}) \cup (OPT \cap \mathcal{N}_2))\right] + \frac{1}{2}\mathbb{E}\left[f(\mathcal{N}_1(\frac{1}{2}))\right] + \frac{1}{2}\mathbb{E}\left[f(\mathcal{N}_1(\frac{1}{2}) \cup \mathcal{N}_2)\right]$.

Proof. This corollary follows directly from Lemma 11 by replacing the fixed set S_1 with the random subset $\mathcal{N}_1\left(\frac{1}{2}\right)$ and then taking the expectation over this randomness. □

Lemma 14. *We have:* $\mathbb{E}\left[f(\mathcal{N}_1(\frac{1}{2}))\right] \geq \frac{1}{4}\left(f(\emptyset) + f(OPT \cap \mathcal{N}_1) + f(\overline{OPT} \cap \mathcal{N}_1) + f(\mathcal{N}_1)\right)$.

Proof. Firstly note that $\mathcal{N}_1 = (\text{OPT} \cap \mathcal{N}_1) \cup (\overline{\text{OPT}} \cap \mathcal{N}_1)$. Hence, we have: $\mathcal{N}_1(\frac{1}{2}) = (\text{OPT} \cap \mathcal{N}_1)(\frac{1}{2}) \cup (\overline{\text{OPT}} \cap \mathcal{N}_1)(\frac{1}{2})$. By setting $p = q = \frac{1}{2}$, $A = \text{OPT} \cap \mathcal{N}_1$, and $B = \overline{\text{OPT}} \cap \mathcal{N}_1$ in Lemma 12,

we obtain:

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

□

Fact 15. *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 16. $\mathbb{E}\left[f(\mathcal{N}_1(\frac{1}{2}) \cup (\mathit{OPT} \cap \mathcal{N}_2)) \right]$ *is lower-bounded by*

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

Proof. We first apply Lemma 12, with $p = q = \frac{1}{2}$, $A = \text{OPT} \cap \mathcal{N}_1$, and $B = \overline{\text{OPT}} \cap \mathcal{N}_1$ and function $g(A) := f(A \cup (\text{OPT} \cap \mathcal{N}_2))$, which is also a non-negative submodular function based on Fact 15. Therefore, $\mathbb{E}\left[g(\mathcal{N}_1(\frac{1}{2})) \right] \geq \frac{1}{4}\left(g(\emptyset) + g(\text{OPT} \cap \mathcal{N}_1) + g(\overline{\text{OPT}} \cap \mathcal{N}_1) + g(\mathcal{N}_1) \right)$.
This implies that:

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

which establishes the lower bound as required. □

Lemma 17. $\mathbb{E}\left[f(\mathcal{N}_1(\frac{1}{2}) \cup \mathcal{N}_2) \right] \geq \frac{1}{4}\left(f(\mathcal{N}_2) + f(\mathit{OPT} \cup \mathcal{N}_2) + f(\overline{\mathit{OPT}} \cup \mathcal{N}_2) + f(\mathcal{N}_1 \cup \mathcal{N}_2) \right)$.

Proof. Similar to Lemma 16, we apply Lemma 12 with $p = q = 1/2$, $A = \text{OPT} \cap \mathcal{N}_1$, and $B = \overline{\text{OPT}} \cap \mathcal{N}_1$, and function $g(A) := f(A \cup \mathcal{N}_2)$, and derive the following inequality:

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

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

Lemma 18. *The following inequality holds:*

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

Proof. Recall the Corollary 13, which states:

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

We replace each term in the right-hand side of this inequality with their respective lower bound that we provided Lemmas 14, 16, and 17. We obtain the following:

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

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

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

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

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

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

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

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

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

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

The bound established in Lemma 18 immediately results in the following Corollary, isolating the components we will utilize in our final proof.

Corollary 19. *The following inequality holds:*

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

Lemma 20. *The maximum expected submodular value of Sol_1 and Sol_2 provides a $\frac{3}{10}$-approximation of $f(\mathit{OPT})$. Specifically,*

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

Proof. Having established a lower bound for the submodular value of Sol_2, we argue that 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, Sol_1 is the output of the subroutine $\text{EXTEND}(\emptyset, \mathcal{N}_1)$. Applying Lemma 6, 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 19, 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: $\max\left(\mathbb{E}[\, f(\text{Sol}_1)\,], \mathbb{E}[\, f(\text{Sol}_2)\,]\right) \geq \frac{3}{10} f(\text{OPT})$. □

Lemma 21. *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 $\text{EXTEND}(S, \mathcal{N})$ is $O(|\mathcal{N}|)$. After each update, our algorithm executes the EXTEND function on the input sets S_1 and $\mathcal{N}_2$, which makes $O(|\mathcal{N}_2|) = O(\sqrt{n})$ query calls since $|\mathcal{N}_2|$ is always less than the buffer threshold of $\sqrt{n}$.

Thus, we only need to focus on the number of queries made to compute Sol_1. We invoke the EXTEND function to compute 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 Sol_2. Thus, we only focus on the computations of Sol_1. Note that invocations related to 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 bounded by $O(|\,\mathcal{N}_1\,|) \leq O(b^2)$, meaning each of these b elements gets charged for no more than $O(b)$ 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. □

Acknowledgments and Disclosure of Funding

The work is partially supported by DARPA expMath, ONR MURI 2024 award on Algorithms, Learning, and Game Theory, Army-Research Laboratory (ARL) grant W911NF2410052, NSF AF:Small grants 2218678, 2114269, 2347322. The authors declare no competing interests.

References

[AB23] Arpit Agarwal and Eric Balkanski. Learning-augmented dynamic submodular maximization. *CoRR*, abs/2311.13006, 2023.

[ABD22] Sepehr Assadi, Aaron Bernstein, and Aditi Dudeja. Decremental matching in general graphs. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 11:1–11:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[AEF+20] Naor Alaluf, Alina Ene, Moran Feldman, Huy L. Nguyen, and Andrew Suh. 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*, pages 6:1–6:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[AS99] Alexander A. Ageev and Maxim Sviridenko. An 0.828-approximation algorithm for the uncapacitated facility location problem. *Discret. Appl. Math.*, 93:149–156, 1999.

[BBG+23] Kiarash Banihashem, Leyla Biabani, Samira Goudarzi, MohammadTaghi Hajiaghayi, Peyman Jabbarzade, and Morteza Monemizadeh. Dynamic non-monotone submodular maximization. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *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.

[BBG+24] Kiarash Banihashem, Leyla Biabani, Samira Goudarzi, MohammadTaghi Hajiaghayi, Peyman Jabbarzade, and Morteza Monemizadeh. Dynamic algorithms for matroid submodular maximization. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 3485–3533. SIAM, 2024.

[BBG+25] Kiarash Banihashem, Leyla Biabani, Samira Goudarzi, MohammadTaghi Hajiaghayi, Peyman Jabbarzade, and Morteza Monemizadeh. Dynamic algorithms for submodular matching. In Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis, editors, *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025, Aarhus, Denmark, July 8-11, 2025*, LIPIcs, pages 19:1–19:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[BF15] Niv Buchbinder and Moran Feldman. Deterministic algorithms for submodular maximization problems. *ACM Transactions on Algorithms (TALG)*, 14:1 – 20, 2015.

[BFNS12] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. 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*, pages 649–658. IEEE Computer Society, 2012.

[BFNS15] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. *SIAM J. Comput.*, 44(5):1384–1402, 2015.

[BGH+25] Kiarash Banihashem, Samira Goudarzi, MohammadTaghi Hajiaghayi, Peyman Jabbarzade, and Morteza Monemizadeh. Non-monotone submodular optimization: p-matchoid constraints and fully dynamic setting. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[BGS21] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. 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*, pages 1000–1008. IEEE, 2021.

[BMKK14] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: massive data summarization on the fly. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 671–680. ACM, 2014.

[BPW19] Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 365–376. ACM, 2019.

[CFN77] Gérard Cornuéjols, Marshall L. Fisher, and George L. Nemhauser. On the uncapacitated location problem. *Annals of discrete mathematics*, 1:163–177, 1977.

[CGQ15] Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In *Automata, Languages, and Programming*, pages 318–330, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[CK05] Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 35:713–728, 2005.

[CKR06] Reuven Cohen, Liran Katzir, and Danny Raz. An efficient approximation for the generalized assignment problem. *Inf. Process. Lett.*, 100:162–166, 2006.

[CP22] Xi Chen and Binghui Peng. 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*, pages 1685–1698. ACM, 2022.

[CW77] Larry Carter and Mark N. Wegman. 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*, pages 106–112, 1977.

[DFL+23] Paul Duetting, Federico Fusco, Silvio Lattanzi, Ashkan Norouzi-Fard, and Morteza Zadimoghaddam. Fully dynamic submodular maximization over matroids. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 8821–8835. PMLR, 2023.

[DK08] Abhimanyu Das and David Kempe. 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*, pages 45–54. ACM, 2008.

[DK18] Abhimanyu Das and David Kempe. Approximate submodularity and its applications: Subset selection, sparse approximation and dictionary selection. *J. Mach. Learn. Res.*, 19:3:1–3:34, 2018.

[EG11] Khalid El-Arini and Carlos Guestrin. 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*, pages 439–447. ACM, 2011.

[FG95] Uriel Feige and Michel X. Goemans. 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*, pages 182–189, 1995.

[FKK18] Moran Feldman, Amin Karbasi, and Ehsan Kazemi. 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*, pages 730–740, 2018.

[FMV11] Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM J. Comput.*, 40(4):1133–1153, 2011.

[FNS11] Moran Feldman, Joseph Naor, and Roy Schwartz. Nonmonotone submodular maximization via a structural continuous greedy algorithm - (extended abstract). In *International Colloquium on Automata, Languages and Programming*, 2011.

[FNSZ20] Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1363–1374. ACM, 2020.

[Fuj84] Satoru Fujishige. Theory of submodular programs: A fenchel-type min-max theorem and subgradients of submodular functions. *Math. Program.*, 29(2):142–155, 1984.

[FV06] Uriel Feige and Jan Vondrák. Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e. *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 667–676, 2006.

[GKS05] Carlos Guestrin, Andreas Krause, and Ajit Paul Singh. Near-optimal sensor placements in gaussian processes. In Luc De Raedt and Stefan Wrobel, editors, *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*, pages 265–272. ACM, 2005.

[GW95] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42:1115–1145, 1995.

[GZZ26] Yiwei Gao, Jialin Zhang, and Zhijie Zhang. Improved fully dynamic submodular maximization under matroid constraints. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor, editors, *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pages 36928–36936. AAAI Press, 2026.

[Hås01] Johan Håstad. Some optimal inapproximability results. *Electron. Colloquium Comput. Complex.*, TR97, 2001.

[HKN15] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Improved algorithms for decremental single-source reachability on directed graphs. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *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*, pages 725–736. Springer, 2015.

[HKN18] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. *J. ACM*, 65(6):36:1–36:40, 2018.

[HSSY24] Monika Henzinger, Barna Saha, Martin P. Seybold, and Christopher Ye. On the complexity of algorithms with predictions for dynamic graph problems. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA*, volume 287 of *LIPIcs*, pages 62:1–62:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[HZ01] Eran Halperin and Uri Zwick. Combinatorial approximation algorithms for the maximum directed cut problem. In *ACM-SIAM Symposium on Discrete Algorithms*, 2001.

[KED+17] Rajiv Khanna, Ethan R. Elenberg, Alexandros G. Dimakis, Sahand N. Negahban, and Joydeep Ghosh. 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*, pages 1560–1568. PMLR, 2017.

[KOKM04] Guy Kindler, Ryan O'Donnell, Subhash Khot, and Elchanan Mossel. Optimal inapproximability results for max-cut and other 2-variable csps? *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 146–154, 2004.

[KZK18] Ehsan Kazemi, Morteza Zadimoghaddam, and Amin Karbasi. 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*, pages 2549–2558. PMLR, 2018.

[LFKK20] Wenxin Li, Moran Feldman, Ehsan Kazemi, and Amin Karbasi. Submodular maximization in clean linear time. In *Neural Information Processing Systems*, 2020.

[LMN+20] Silvio Lattanzi, Slobodan Mitrovic, Ashkan Norouzi-Fard, Jakub Tarnawski, and Morteza Zadimoghaddam. 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.

[LS24] Quanquan C. Liu and Vaidehi Srinivas. The predicted-updates dynamic model: Offline, incremental, and decremental to fully dynamic transformations. In Shipra Agrawal and Aaron Roth, editors, *The Thirty Seventh Annual Conference on Learning Theory, June 30 - July 3, 2024, Edmonton, Canada*, volume 247 of *Proceedings of Machine Learning Research*, pages 3582–3641. PMLR, 2024.

[MJK18] Baharan Mirzasoleiman, Stefanie Jegelka, and Andreas Krause. 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*, pages 1379–1386. AAAI Press, 2018.

[MKK17] Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. 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*, pages 2449–2458. PMLR, 2017.

[Mon20] Morteza Monemizadeh. 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.

[MV22] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *Commun. ACM*, 65(7):33–35, 2022.

[OGV11] Shayan Oveis Gharan and Jan Vondrák. 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*, pages 1098–1116. SIAM, 2011.

[PR23] Binghui Peng and Aviad Rubinstein. Fully-dynamic-to-incremental reductions with known deletion order (e.g. sliding window). In Telikepalli Kavitha and Kurt Mehlhorn, editors, *2023 Symposium on Simplicity in Algorithms, SOSA 2023, Florence, Italy, January 23-25, 2023*, pages 261–271. SIAM, 2023.

[SSS07] Ian Simon, Noah Snavely, and Steven M. Seitz. Scene summarization for online image collections. In *IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007*, pages 1–8. IEEE Computer Society, 2007.

[SSSJ12] Ruben Sipos, Adith Swaminathan, Pannaga Shivaswamy, and Thorsten Joachims. 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*, pages 754–763. ACM, 2012.

[TAC+20] Ehsan Tohidi, Rouhollah Amiri, Mario Coutino, David Gesbert, Geert Leus, and Amin Karbasi. Submodularity in action: From machine learning to signal processing applications. *IEEE Signal Process. Mag.*, 37(5):120–133, 2020.

[TIWB14] Sebastian Tschiatschek, Rishabh K Iyer, Haochen Wei, and Jeff A Bilmes. Learning mixtures of submodular functions for image collection summarization. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[vdBFNP24] Jan van den Brand, Sebastian Forster, Yasamin Nazari, and Adam Polak. On dynamic graph algorithms with predictions. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 3534–3557. SIAM, 2024.

A 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.

A.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.

A.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 $\text{OPT}_t = \arg\max_{A \subseteq V_t} f(A)$, where $V_t := \{e_{t+1}, \ldots, e_n\}$.

A.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 Sol'_1 and 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 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 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 Sol'_1 and 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 Sol'_1 and Sol'_2 provide 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.

A.1.3 Analysis

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

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**
    $\mathcal{M}_1 \leftarrow \mathcal{M}_2[: \lfloor \sqrt{n} \rfloor]$
    $\mathcal{M}_2 \leftarrow \mathcal{M}_2[\lfloor \sqrt{n} \rfloor :]$
    $\text{Sol}'_2 \leftarrow$ EXTEND($\emptyset, \mathcal{M}_2$)
    $S_2 \leftarrow \mathcal{M}_2(\frac{1}{2})$
  $\mathcal{M}_1.pop(0)$
  $\text{Sol}'_1 \leftarrow$ EXTEND($S_2, \mathcal{M}_1$)
  return $\arg\max\{f(\text{Sol}'_1), f(\text{Sol}'_2)\}$

Theorem 22. *There exists an algorithm for the USM problem in the decremental dynamic setting described in Section A.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 23. The amortized query complexity is $O(\sqrt{n})$ per update which can be proven similar to Lemma 21. This concludes the proof. □

Lemma 23 (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, OPT_t and $\overline{\text{OPT}}$ will maintain the same definitions as previously established.

Lemma 24. *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 25. *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 26. *We can assert:*

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

Lemma 27. $\mathbb{E}\left[\, f(\mathcal{M}_2(\frac{1}{2})) \,\right] \geq \frac{1}{4}\left(f(\emptyset) + f(OPT \cap \mathcal{M}_2) + f(\overline{OPT} \cap \mathcal{M}_2) + f(\mathcal{M}_2)\right).$

Lemma 28. $\mathbb{E}\left[\, f(\mathcal{M}_2(\frac{1}{2}) \cup (OPT \cap \mathcal{M}_1)) \,\right]$ *is lower-bounded by*

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

Lemma 29.

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

Lemma 30. *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 31. *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 32. *The maximum expected submodular value of* Sol'_1 *and* Sol'_2 *provides a* $\frac{3}{10}$*-approximation of* $f(\mathit{OPT})$. *Specifically,* $\max\left(\mathbb{E}\left[f(\mathit{Sol}'_1) \right], \mathbb{E}\left[f(\mathit{Sol}'_2) \right]\right) \geq \frac{3}{10} f(\mathit{OPT})$.

A.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.

A.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.

A.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 Sol_2. The solution set 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:

- **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 Sol_1 and 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 $\text{Update}_t = \text{INSERT}(e)$ **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 $\text{Update}_t = \text{DELETE}(e)$ **then**
      if $e \in \mathcal{N}_2$ **then**
        $\mathcal{N}_2 \leftarrow \mathcal{N}_2 \backslash \{e\}$
      else
        $\text{Sol}_1 \leftarrow \mathcal{I}.\text{DELETE}()$
        $\mathcal{N}_1 \leftarrow \mathcal{N}_1 \backslash \{e\}$
        $S_1 \leftarrow S_1 \backslash \{e\}$
      $\text{Sol}_2 \leftarrow \text{EXTEND}(S_1, \mathcal{N}_2)$
      yield $\arg\max\{f(\text{Sol}_1), f(\text{Sol}_2)\}$

A.2.3 Analysis

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

Theorem 33. *There exists an algorithm for the USM problem in a fully dynamic setting described in Section A.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 34. The amortized query complexity is $O(\sqrt{n})$ per update, which can be proven similarly to Lemma 21. This concludes the proof. □

Lemma 34 (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:*

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

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 Sol_2. Hence, Lemmas 10 through Lemma 18 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 20. □

Lemma 35. *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, 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 19, 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. □

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly and accurately state the claims.

   Guidelines:

- The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The assumptions made in the paper as well as the results are clearly stated and therefore the limitations are clear.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: A complete proof of theoretical results is provided.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.

- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [N/A]

   Justification: The paper makes a significant theoretical contribution and does not include experimental results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [N/A]

   Justification: The paper makes a significant theoretical contribution and does not include experimental results.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.

- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [N/A]

   Justification: The paper makes a significant theoretical contribution and does not include experimental results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [N/A]

   Justification: The paper makes a significant theoretical contribution and does not include experimental results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).

- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [N/A]

   Justification: The paper makes a significant theoretical contribution and does not include experimental results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Conforms to the guidelines.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: The paper is theoretical in nature and does not have societal impact.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the

technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [N/A]

    Justification: The paper does not use existing assets.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: No new assets are released.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.

- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not use crowdsourcing.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: No LLMs were used in conducting the research or generating the results.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.