A. Proof of Theorem 1

Proof. **Approximation guarantee** The approximation ratio is proven very similar to the approximation guarantee of SIEVE-STREAMING (Badanidiyuru et al., 2014). Let's define $S^* = \arg\max_{A \subseteq V, |A| \leq k} f(A)$, $\texttt{OPT} = f(S^*)$ and $\tau^* = \frac{\texttt{OPT}}{2k}$. We further define $\Delta = \max_{e \in V} f(\{e\})$. It is easy to observe that $\max\{\Delta, \texttt{LB}\} \leq \texttt{OPT} \leq k\Delta$ and there is a threshold τ such that $(1-\varepsilon)\tau^* \leq \tau < \tau^*$. Now consider the set S_τ. SIEVE-STREAMING++ adds elements with a marginal gain at least τ to the set S_τ. We have two cases:

- $|S_\tau| = k$: We define $S_\tau = \{e_1, \cdots, e_k\}$ where e_i is the i-th picked element. Furthermore, we define $S_{\tau,i} = \{e_1, \cdots, e_i\}$. We have
$$f(S_\tau) = \sum_{i=1}^{k} f(\{e_i\} \mid S_{\tau,i-1}) \geq k\tau \geq (\frac{1}{2} - \varepsilon) \cdot \texttt{OPT}.$$
This is true because the marginal gain of each element at the time it has been added to the set S_τ is at least τ^*.
- $|S_\tau| < k$: We have
$$\texttt{OPT} \leq f(S^* \cup S_\tau) = f(S_\tau) + f(S^* \mid S_\tau) \overset{(a)}{\leq} f(S_\tau) + \sum_{e \in S^* \setminus S_\tau} f(\{e\} \mid S_\tau) \overset{(b)}{\leq} f(S_\tau) + k\tau^* = f(S_\tau) + \frac{\texttt{OPT}}{2},$$
where (a) is correct because f is a submodular function, and we have (b) because each element of S^* that is not picked by the algorithm has had a marginal gain of less than $\tau < \tau^*$.

Memory complexity Let S_τ be the set we maintain for threshold τ. We know that $\texttt{OPT}$ is at least $\texttt{LB} = \max_\tau f(S_\tau) \geq \max_\tau(|S_\tau| \times \tau)$ because the marginal gain of an element in set S_τ is at least τ. Note that $\texttt{LB}$ is the best solution found so far. Given this lower bound on $\texttt{OPT}$ (which is potentially better than Δ if we have picked enough items), we can dismiss all thresholds that are too small, i.e., remove all thresholds $\tau < \frac{\texttt{LB}}{2k} \leq \frac{\texttt{OPT}}{2k}$. For any remaining $\tau \geq \frac{\texttt{LB}}{2k}$, we know that $|S_\tau|$ is at most $\frac{\texttt{LB}}{\tau}$. We consider two sets of thresholds: (i) $\frac{\texttt{LB}}{2k} \leq \tau \leq \frac{\texttt{LB}}{k}$, and (ii) $\tau \geq \frac{\texttt{LB}}{k}$. For the first group of thresholds, the bound on $|S_\tau|$ is the trivial upper bound of k. Note that we have $\log_{1+\varepsilon}(2) \leq \lceil \log(2)/\varepsilon \rceil$ of such thresholds. For the second group of thresholds, as we increase τ, for a fixed value of $\texttt{LB}$ the upper bound on the size of S_τ gets smaller. Indeed, these upper bounds are geometrically decreasing values with the first term equal to k. And they reduce by a coefficient of $(1+\varepsilon)$ as thresholds increase by a factor of $(1+\varepsilon)$. Therefore, we can bound the memory complexity by

$$\text{Memory complexity} \leq \left\lceil \frac{k \log(2)}{\varepsilon} \right\rceil + \sum_{i=0}^{\log_{1+\varepsilon}(k)} \frac{k}{(1+\varepsilon)^i} = O\left(\frac{k}{\varepsilon}\right).$$

Therefore, the total memory complexity is $O(\frac{k}{\varepsilon})$.

Query complexity For every incoming element e, in the worst case, we should compute the marginal gain of e to all the existing sets S_τ. Because there is $O(\frac{\log k}{\varepsilon})$ of such sets (the number of different thresholds), therefore the query complexity per element is $O(\frac{\log k}{\varepsilon})$. □

B. Proof of Theorem 2

Proof. **Approximation Guarantee:** Assume $\mathcal{B}$ is the set of elements buffered from the stream V. Let's define $S^* = \arg\max_{A \subseteq V, |A| \leq k} f(A)$, $\texttt{OPT} = f(S^*)$ and $\tau^* = \frac{\texttt{OPT}}{2k}$. Similar to the proof of Theorem 1, we can show that BATCH-SIEVE-STREAMING++ considers a range of thresholds such that for one of them (say τ) we have $\frac{\texttt{OPT}(1-\varepsilon)}{2k} \leq \tau < \frac{\texttt{OPT}}{2k}$. In the rest of this proof, we focus on τ and its corresponding set of picked items S_τ. For set S_τ we have two cases:

- if $|S_\tau| < k$, we have:
$$\texttt{OPT} \leq f(S^* \cup S_\tau) = f(S_\tau) + f(S^* \mid S_\tau) \overset{(a)}{\leq} f(S_\tau) + \sum_{e \in S^* \setminus S_\tau} f(\{e\} \mid S_\tau) \overset{(b)}{\leq} f(S_\tau) + k\tau^* = f(S_\tau) + \frac{\texttt{OPT}}{2},$$

where inequality (a) is correct because f is a submodular function. And inequality (b) is correct because each element of the optimal set S^* that is not picked by the algorithm, i.e., it had discarded in the filtering process, has had a marginal gain of less than $\tau < \tau^*$.

- if $|S_\tau| = k$: Assume the set S_τ of size k is sampled in ℓ iterations of the while loop in Lines 2–18 of Algorithm 3, and T_i is the union of sampled batches in the i-th iteration of the while loop. Furthermore, let $T_{i,j}$ denote the j-th sampled batch in the i-th iteration of the while loop. We define $S_{\tau,i,j} = \bigcup_{i,j} T_{i,j}$, i.e, $S_{\tau,i,j}$ is the state of set S_τ after the j-th batch insertion in the i-th iteration of the while loop. We first prove that the average gain of each one of these sets T_i to the set S_τ is at least $(1-2\varepsilon)\cdot|T_i|$.

  To lower bound the average marginal gain of T_i, for each $T_{i,j}$ we consider three different cases:

  - the while loop breaks at Line 7 of Algorithm 3: We know that the size of all $T_{i,j}$ is one in this case. It is obvious that $f(T_{i,j} \mid S_{\tau,i,j-1}) \geq (1-\varepsilon)\cdot\tau$.
  - THRESHOLD-SAMPLING passes the first loop and does not break at Line 16, i.e., it continues to pick items till $S_\tau = k$ or the buffer memory is empty: again it is obvious that

    $$f(T_{i,j} \mid S_{\tau,i,j-1}) \geq (1-\varepsilon)\cdot\tau\cdot|T_{i,j}|$$

    This is true because when set $T_{i,j}$ is picked, it has passed the test at Line 15. Note that it is possible the algorithm breaks at Line 14 without passing the test at Line 15. If the average marginal gain of the sampled set $T_{i,j}$ is more than $(1-\varepsilon)\cdot\tau$ then the analysis would be exactly the same as the current case. Otherwise, we handle it similar to the next case where the sampled batch does not provide the required average marginal gain.
  - it passes the first loop and breaks at Line 16. We have the two following observations:
    1. in the current while loop, from the above-mentioned cases, we conclude that the average marginal gain of all the element picked before the last sampling is at least $(1-\varepsilon)\cdot\tau$, i.e.,

       $$\forall r, 1 \leq r < j : f(T_{i,r} \mid S_{\tau,i,r-1}) \geq (1-\varepsilon)\cdot\tau\cdot|T_{i,r}|.$$

    2. the number of elements which are picked at the latest iteration of the while loop is at most ε fraction of all the elements picked so far (in the current while loop), i.e., $|T_{i,j}| \leq \varepsilon\cdot|\bigcup_{1\leq r<j} T_{i,r}|$ and $|T_i| \leq (1+\varepsilon)\cdot|\bigcup_{1\leq r<j} T_{i,r}|$. Therefore, from the monotonicity of f, we have

       $$f(\bigcup_{1\leq r\leq j} T_{i,r} \mid S) \geq f(\bigcup_{1\leq r<j} T_{i,r} \mid S) \geq (1-\varepsilon)\cdot\tau\cdot|\bigcup_{1\leq r<j} T_{i,r}| \geq \frac{|T_i|\cdot\tau\cdot(1-\varepsilon)}{(1+\varepsilon)} \geq (1-2\varepsilon)\cdot\tau\cdot|T_i|.$$

  To sum-up, we have

  $$f(S_\tau) = \sum_{i=1}^{\ell} f(T_i \mid S_{\tau,i-1}) \geq \sum_{i=1}^{\ell}(1-2\varepsilon)\cdot\tau\cdot|T_i| = (1-2\varepsilon)\cdot\tau\cdot k \geq \left(\frac{1}{2}-\frac{3\varepsilon}{2}\right)\cdot\texttt{OPT}.$$

Memory complexity In a way similar to analyzing the memory complexity of SIEVE-STREAMING++, we conclude that the required memory of BATCH-SIEVE-STREAMING++ in order to store solutions for different thresholds is also $O(\frac{k}{\varepsilon})$. Since we buffer at most $\texttt{B}$ items, the total memory complexity is $O(\texttt{B}+\frac{k}{\varepsilon})$.

Adaptivity Complexity of THRESHOLD-SAMPLING To guarantee the adaptive complexity of our algorithm, we first upper bound the expected number of iterations of the while loop in Lines 2–18 of Algorithm 3.

Lemma 1. *For any constant $\varepsilon > 0$, the expected number of iterations in the while loop of Lines 2–18 of* THRESHOLD-SAMPLING *is $O(\log(|\mathcal{B}|))$ where $\mathcal{B}$ is the set of buffered elements passed to* THRESHOLD-SAMPLING.

Before, proving Lemma 1, we discuss how this lemma translates to the total expected adaptivity of BATCH-SIEVE-STREAMING++.

There are at most $\lceil\frac{1}{\varepsilon}\rceil + \log_{1+\varepsilon} k = O(\frac{\log k}{\varepsilon})$ adaptive rounds in each iteration of the while loop of Algorithm 3. So, from Lemma 1 we conclude that the expected adaptive complexity of each call to THRESHOLD-SAMPLING is $O(\frac{\log(|\mathcal{B}|)\log(k)}{\varepsilon})$. To sum up, the general adaptivity of the algorithm takes its maximum value when the number of times a buffer gets full is the most, i.e., when $|\mathcal{B}| = \texttt{Threshold}\times\texttt{B}$ for $\frac{N}{\texttt{Threshold}\times\texttt{B}}$ times. We assume $\texttt{Threshold}$ is constant. Therefore, the expected adaptive complexity is $O(\frac{N\log(\texttt{B})\log(k)}{\texttt{B}\varepsilon})$. □

Proof of Lemma 1

Proof. Since we are only adding elements to S, using submodularity the marginal value of any element x to S, i.e. $f(x \mid S)$, is decreasing. Therefore, once an element is removed from the buffer $\mathcal{B}$, it never comes back. As a result, the set $\mathcal{B}$ is shrinking over time. When $\mathcal{B}$ becomes empty, the algorithm terminates. Therefore it suffices to show that in every iteration of the while loop, a constant fraction of elements will be removed from $\mathcal{B}$ in expectation. The rest of the analysis follows by analyzing the expected size of $\mathcal{B}$ over time and applying Markov's inequality.

We note that to avoid confusion, we call one round of the while loop in Lines 3–18 of THRESHOLD-SAMPLING an iteration. There are two other internal for loops at Lines 4–10 and Lines 11–18. Later in the proof, we call each run of these for loops a step. There are $\lceil \frac{1}{\varepsilon} \rceil$ steps in the first for loop and $O(\log(k))$ steps in the second.

If an iteration ends with growing S into a size k set, that is going to be the final iteration as the algorithm THRESHOLD-SAMPLING because the algorithm terminates once k elements are selected. So we focus on the other case. An iteration breaks (finishes) either in the first for the loop at Lines 4–10 or in the second for loop of Lines 11–18. We say an iteration fails if after termination less than $\varepsilon/2$ fraction of elements in $\mathcal{B}$ is removed. For iteration ℓ, let $\mathcal{B}^\ell$ be the set $\mathcal{B}$ at Line 3 at the beginning of this iteration. So the first set $\mathcal{B}^1$ consists of all the input elements passed to THRESHOLD-SAMPLING. So we can say that an iteration ℓ fails if $|\mathcal{B}^{\ell+1}|$ is greater than $(1-\varepsilon/2) \cdot |\mathcal{B}^\ell|$.

Failure of an iteration can happen in any of the $\lceil \frac{1}{\varepsilon} \rceil + O(\log(k))$ steps of the two for loops. For each step $1 \leq z \leq \lceil \frac{1}{\varepsilon} \rceil + O(\log(k))$, we denote the probability that the current iteration is terminated at step z at a failed state with P_z. The probability that an iteration will not fail can then be written as

$$\prod_z (1 - P_z).$$

In the rest of the proof, we show that this quantity is at least a constant for any constant $\varepsilon > 0$.

First, we show that at any of the $\lceil \frac{1}{\varepsilon} \rceil$ steps of the first for loop, the probability of failing is less than $\varepsilon/2$. Let us consider step $1 \leq z \leq \lceil \frac{1}{\varepsilon} \rceil$. We focus on the beginning of step z and upper bound P_z for any possible outcome of the previous steps $1, \cdots, z-1$. Let S be the set of selected elements in all the first $z-1$ steps. If at least $\varepsilon/2$ fraction of elements in $\mathcal{B}^\ell$ has a marginal value less than τ to S, we can say that the iteration will not fail for the rest of the steps for sure (with probability 1). We note that as S grows the marginal values of elements to it will not increase, so at least $\varepsilon/2$ fraction of elements will be filtered out independent of which step the process terminates.

So we focus on the case that less than $\varepsilon/2$ fraction of elements in $\mathcal{B}^\ell$ have marginal value less than τ to S. Since, in the first loop, we pick one of them randomly and look at its marginal value as a test to whether terminate the iteration or not, the probability of termination at this step z is not more than $\varepsilon/2$ and therefore P_z is also at most $\varepsilon/2$.

In the second for loop, at Lines 11–18, we have a logarithmic number of steps and we can upper bound the probability of terminating the iteration in a failed state at any of these steps in a similar way. The main difference is that instead of sampling one random element from $\mathcal{B}$, we pick t random elements and look at their average marginal value together as a test to whether terminate the current iteration or not.

We want to upper bound the probability of terminating the iteration in a step $z > \lceil \frac{1}{\varepsilon} \rceil$ at a failed state. This will happen if at the step z the THRESHOLD-SAMPLING algorithm picks a random subset T with

- $\dfrac{f(T \mid S)}{|T|} \leq (1-\varepsilon)\tau$, and
- also less than $\varepsilon/2$ fraction of elements in $\mathcal{B}^\ell$ has a marginal value less than τ to $T \cup S$.

We look at the process of sampling T as a sequential process in which we pick t random elements one by one. We can call each of these t parts a small random experiment. We note that the first property above holds only if in at least εt of these smaller random experiments the marginal value of the selected element to the current set S is below τ. We also assume that we add the selected elements to S as we move on. We simulate this random process with a binomial process of tossing t independent coins. If the marginal value of the i-th sampled element to S is at least τ, we say that the associated coin toss is a head. Otherwise, we call it a tail. The probability of a tail depends on the fraction of elements in $\mathcal{B}^\ell$ with marginal value

less than τ to S. If this fraction at any point is at least $\varepsilon/2$, we know that the second necessary property for a failed iteration does not hold anymore and will not hold for the rest of the steps. Therefore the failure happens only if we face at least εt tails each with probability at most $\varepsilon/2$. The rest of the analysis is applying simple concentration bounds for different values of t.

So we have a binomial distribution with t trials each with head probability at least $1-\varepsilon/2$, and we want to upper bound the probability that we get at least εt tails. The expected number of tails is not more than $\varepsilon t/2$ so using Markov's inequality, the probability of seeing at least εt tails is at most 0.5. Furthermore, for larger values of t we can have much better concentration bounds.

Using Chernoff type bounds in Lemma 2, we know the probability of observing at least εt tails is not more than:

$$P_z \leq \Pr(\text{tails} - \varepsilon t/2 \geq \varepsilon t/2) \leq e^{-\varepsilon t/10}.$$

As we proceed in steps, the number of samples t grows geometrically. Consequently, the failure probability declines exponentially (double exponential in the limit).

So the number of steps it takes to reach the failure probability declining phase is a function of ε and therefore it is a constant number. We conclude that for any constant $\varepsilon > 0$, the probability of not failing in an iteration, i.e., $\prod_z (1-P_z)$, is lower bounded by a constant $\zeta_\varepsilon > 0$. Since any iteration will terminate eventually, we can say that for any iteration with constant probability an $\varepsilon/2$ fraction of elements will be filtered out of $\mathcal{B}$. So the expected size of $\mathcal{B}$ after X iterations will be at most $2^{-\Omega(X)}n$ where n is the number of input elements at the beginning of THRESHOLD-SAMPLING. So the probability of having more than $C\log(|\mathcal{B}|)$ iterations decreases exponentially with C for any coefficient C using Markov's inequality which means the expected number of iterations is $O(\log(|\mathcal{B}|))$. □

Lemma 2 (Chernoff bounds, (Bansal & Sviridenko, 2006)). *Suppose $X_1, \ldots, X_n$ are binary random variables such that $\Pr(X_i = 1) = p_i$. Let $\mu = \sum_{i=1}^n p_i$ and $X = \sum_{i=1}^n X_i$. Then for any $a > 0$, we have*

$$\Pr(X - \mu \geq a) \leq e^{-a\min\left(\frac{1}{5}, \frac{a}{4\mu}\right)}.$$

Moreover, for any $a > 0$, we have

$$\Pr(X - \mu \leq -a) \leq e^{-\frac{a^2}{2\mu}}.$$

C. Proof of Theorem 3

For m different data streams, assume $\mathcal{B}_i$ is the set of elements buffered from the i-th stream. We define $\mathcal{B}$ to be the union of all elements from all streams. The communication cost of BATCH-SIEVE-STREAMING++ in the multi-source setting is the total number of elements sampled (in a distributed way) from all sets $\{\mathcal{B}_i\}_{1\leq i\leq m}$ in Lines 5 and 13 of Algorithm 3.

As a result, we can conclude that the communication cost is at most twice the number of elements has been in a set S_τ at a time during the run of the algorithm. To see the reason for this argument, note that because the filtering step happens just before the for loop of Lines 4–10, the first picked sample in this for loop always passes the test and is added to S_τ. Furthermore, all the items sampled at Line 13, irrespective of their marginal gain, are added to S_τ. So, in the worst case scenario, the communication complexity is maximum when the for loop breaks always at the second instance of the sampling process of Line 5 (after one successful try). Therefore, we only need to upper bound the total number of elements which at some point has been in a set S_τ at one of the calls to THRESHOLD-SAMPLING.

The first group of thresholds the BATCH-SIEVE-STREAMING++ algorithm considers the interval $[\Delta_0/(2k), \Delta_0]$, where in the beginning we have $\texttt{LB} = \Delta_0$. Following the same arguments as the proof of Theorem 1, we can show that if neither $\texttt{LB}$ nor Δ changes, the total number of elements in sets $\{S_\tau\}$ is $O(\frac{k}{\varepsilon})$. We define $\Delta_{\max}$ to be the largest singleton element in the whole data streams. The number of times the interval of thresholds changes because of the change in Δ is $\log_{1+\varepsilon}(\Delta_{\max}/\Delta_0)$. Furthermore, by changes in $\texttt{LB}$ some thresholds and their corresponding sets are deleted and new elements might be added. The number of times $\texttt{LB}$ changes is upper bounded by $\log_{1+\varepsilon}(\texttt{OPT}/\Delta_0)$. Note that we have $\Delta_{\max} \leq \texttt{OPT}$. From the fact that the number of changes in the set of thresholds is upper bounded by $\log_{1+\varepsilon}(\Delta_{\max}/\Delta_0) = O(\frac{\log C}{\varepsilon})$ and the number of elements in $\{S_\tau\}$ at every step of the algorithm is $O(\frac{k}{e})$, we conclude the total communication cost of BATCH-SIEVE-STREAMING++ is $O(\frac{k\log C}{\varepsilon^2})$.

D. Implications of SIEVE-STREAMING++

Recently, there has been several successful instances of using the idea of (Badanidiyuru et al., 2014) for designing streaming algorithms for a wide range of submodular maximization problems. In Section 3, we showed SIEVE-STREAMING++ (see Algorithm 1 and Theorem 1) reduces the memory complexity of streaming submodular maximization to $O(k)$. In this section, we discuss how the approach of SIEVE-STREAMING++ significantly improves the memory complexity for several important problems.

Random Order Streams Norouzi-Fard et al. (2018) studied streaming submodular maximization under the assumption that elements of a stream arrive in random order. They introduced a streaming algorithm called SALSA with an approximation guarantee better than $1/2$. This algorithm uses $O(k\log(k))$ memory. In a very straightforward way, similarly to the idea of SIEVE-STREAMING for lower bounding the optimum value, we are able to improve the memory complexity of this algorithm to $O(k)$. Furthermore, (Norouzi-Fard et al., 2018) introduced a p-pass algorithm ($p \geq 2$) for submodular maximization subject to a cardinality constraint k. We can also reduce the memory of this p-pass algorithm by a factor $\log(k)$.

Deletion-Robust Mirzasoleiman et al. (2017) have introduced a streaming algorithm for the deletion-robust submodular maximization. Their algorithm provides a summary S of size $O(kd\log(k)/\varepsilon)$ where it is robust to deletion of any set D of at most d items. Kazemi et al. (2018) were able to reduce the size of the deletion-robust summary to $O(k\log(k)/\varepsilon + d\log^2(k)/\varepsilon^3)$. The idea of SIEVE-STREAMING++ for estimating the value of OPT reduces the memory complexities of these two algorithms to $O(kd/\varepsilon)$ and $O(k/\varepsilon + d\log^2(k)/\varepsilon^3)$, respectively. It is also possible to reduce the memory complexity of STAR-T-GREEDY (Mitrovic et al., 2017b) by at least a factor of $\log(k)$.

Two-Stage Mitrovic et al. (2018) introduced a streaming algorithm called REPLACEMENT-STREAMING for the two-stage submodular maximization problem which is originally proposed by (Balkanski et al., 2016; Stan et al., 2017). The memory complexity of REPLACEMENT-STREAMING is $O(\frac{\ell\log\ell}{\varepsilon})$, where ℓ is the size of the produced summary. Again, by applying the idea of SIEVE-STREAMING++ for guessing the value of OPT and analysis similar to the proof of Theorem 1, we can reduce the memory complexity of the streaming two-stage submodular maximization to $O(\frac{\ell}{\varepsilon})$.

Streaming Weak Submodularity Weak submodular functions generalize the diminishing returns property.

Definition 1 (Weakly Submodular (Das & Kempe, 2011))**.** A monotone and non-negative set function $f : 2^V \to \mathbb{R}_{\geq 0}$ is γ–weakly submodular if for each sets $A, B \subset V$, we have

$$\gamma \leq \frac{\sum_{e\in A\setminus B} f(\{e\} \mid B)}{f(A \mid B)},$$

where the ratio is considered to be equal to 1 when its numerator and denominator are 0.

It is straightforward to show that f is submodular if and only if $\gamma = 1$. In the streaming context subject to a cardinality constraint k, Elenberg et al. (2017) designed an algorithm with a constant factor approximation for γ–weakly submodular functions. The memory complexity of their algorithm is $O(\frac{k\log k}{\varepsilon})$. By adopting the idea of SIEVE-STREAMING++, we could reduce the memory complexity of their algorithm to $O(\frac{k}{\varepsilon})$.

Table 2 provides a summary of algorithms that we could significantly improve their memory complexity, while their approximation factors are maintained.

E. Twitter Dataset Details

E.1. Intuition

To clean the data, we removed punctuation and common English words (known as stop words, thus leaving each individual tweet as a list of keywords with a particular timestamp. To give additional value to more popular posts, we also saved the number of retweets each post received.

Therefore, any individual tweet t consists of a set of keywords K_t and a value v_t that is the number of retweets divided by the number of words in the post.

Table 2. Streaming algorithms for several other submodular maximization problems.

Problem	Algorithm	Memory	Improved Memory	Ref.
Stream of random order	SALSA	$O(k \log(k))$	$O(k)$	Norouzi-Fard et al. (2018)
Stream of random order	P-PASS	$O(k \log(k)/\varepsilon)$	$O(k/\varepsilon)$	Norouzi-Fard et al. (2018)
Weakly submodular	STREAK	$O(k \log(k)/\varepsilon)$	$O(k/\varepsilon)$	Elenberg et al. (2017)
Deletion-Robust	ROBUST	$O(kd \log(k)/\varepsilon)$	$O(kd/\varepsilon)$	Mirzasoleiman et al. (2017)
Deletion-Robust	ROBUST-STREAMING	$O(k \log(k)/\varepsilon + d \log^2(k)/\varepsilon^3)$	$O(k/\varepsilon + d \log^2(k)/\varepsilon^3)$	Kazemi et al. (2018)
Two-Stage	REPLACEMENT-STREAMING	$O(\ell \log(\ell)/\varepsilon)$	$O(\ell/\varepsilon)$	Mitrovic et al. (2018)

A set of tweets T can be thought of as a list of $(keyword, score)$ pairs. The keywords K_T in a set T is simply a union of the keywords of the tweets in T:

$$K_T = \bigcup_{t \in T} K_t$$

The score s_k of each keyword $k \in K_T$ is simply the sum of the values of posts containing that keyword. That is, if $T_k \subseteq T$ is the subset of tweets in T containing the keyword k, then:

$$s_k = \sum_{t \in T_k} v_t$$

Therefore, we define our submodular function f as follows:

$$f(T) = \sum_{k \in K_T} \sqrt{s_k}$$

Intuitively, we sum over all the keyword scores because we want our set of tweets to cover as many high-value keywords as possible. However, we also use the square root to introduce a notion of diminishing returns because once a keyword already has a high score, we would prefer to diversify instead of further picking similar tweets.

E.2. General Formalization

In this section, we first rigorously define the function used for Twitter stream summarization in Section 5.1. We then prove this function is non-negative and monotone submodular.

Function Definition Consider a function f defined over a ground set V of items. Each item $e \in V$ consists of a positive value val_e and a set of ℓ_e keywords $W_e = \{w_{e,1}, \cdots, w_{e,\ell_e}\}$ from a general set of keywords $\mathcal{W}$. The score of a word $w \in W_e$ for an item e is defined by $\text{score}(w, e) = \text{val}_e$. If $w \notin W_e$, we define $\text{score}(w, e) = 0$. For a set $S \subseteq V$ the function f is defined as follows:

$$f(S) = \sum_{w \in \mathcal{W}} \sqrt{\sum_{e \in S} \text{score}(w, e)}. \tag{2}$$

Lemma 3. *The function f defined in Eq. (2) is non-negative and monotone submodular.*

Proof. The not-negativity and monotonicity of f are trivial. For two sets $A \subset B$ and $e \in V \setminus B$ we show that

$$f(\{e\} \cup A) - f(A) \geq f(\{e\} \cup B) - f(B).$$

To prove the above inequality, assume $W_e = \{w_{e,1}, \cdots, w_{e,\ell_e}\}$ is the set of keywords of e. For a keyword $w_{e,i}$ define $a_{w_{e,i}} = \sum_{u \in A} \text{score}(w_{e,i}, u)$ and $b_{w_{e,i}} = \sum_{u \in B} \text{score}(w_{e,i}, u)$. It is obvious that $a_{w_{e,i}} \leq b_{w_{e,i}}$. It is straightforward to show that

$$\sqrt{a_{w_{e,i}} + \text{score}(w_{e,i}, e)} - \sqrt{a_{w_{e,i}}} \geq \sqrt{b_{w_{e,i}} + \text{score}(w_{e,i}, u)} - \sqrt{b_{w_{e,i}}}.$$

If sum over all keywords in W_e then the submodularity of f is proven. □

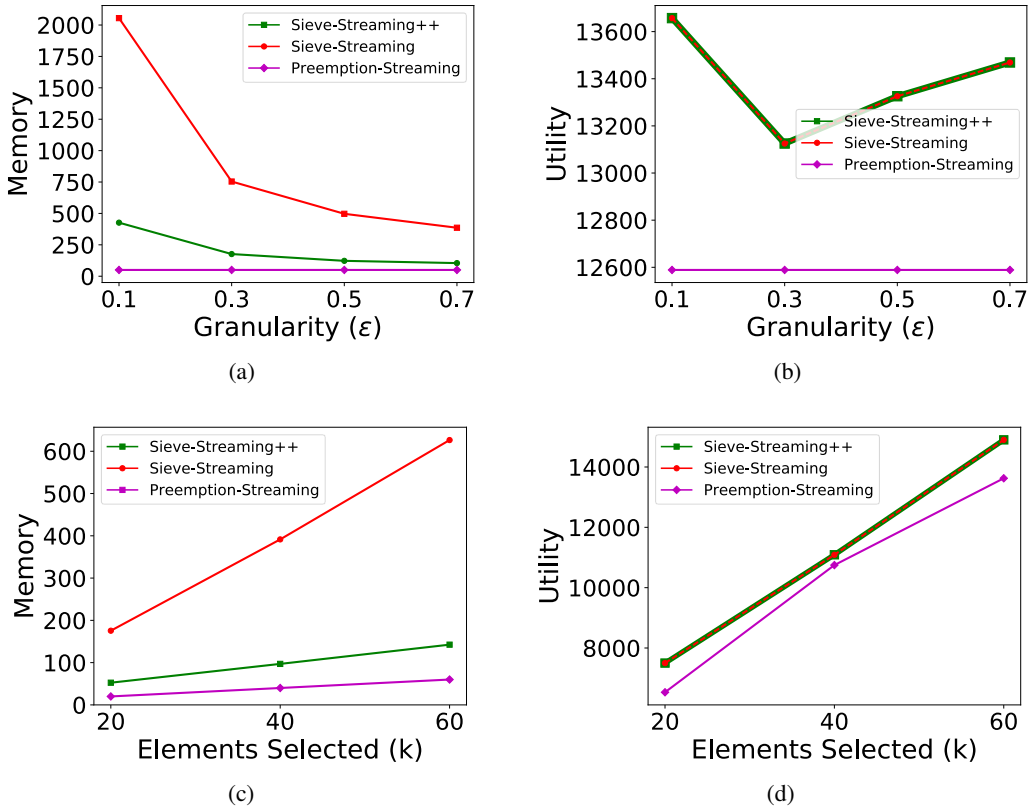

Figure 6. Single-source streaming results on the Twitter summarizaton task. In (a) and (b), $\varepsilon = 0.5$. In (c) and (d), $k = 50$.

F. More Experimental Results

In this section, we will present a few more graphs that we didn't have space for in the main paper.

F.1. Single-Source Experiments

Here we present the set of graphs that we displayed in Figures 4a through 4d, except here they are run on the Twitter dataset instead. For the most part, they are showing the same trends we saw before. SIEVE-STREAMING++ has the exact same utility as SIEVE-STREAMING, which is better than PREEMPTION-STREAMING. We also see the memory requirement of SIEVE-STREAMING++ is much lower than that of SIEVE-STREAMING, as we had hoped.

The only real difference is in the shape of the utility curve as ε varies. In Figure 4b, the utility was decreasing as ε increased, which is not necessarily the case here. However, this is relatively standard because changing ε completely changes the set of thresholds kept by SIEVE-STREAMING++, so although it usually helps the utility, it is not necessarily guaranteed to do so.

Also, note that we only went up to $k = 60$ in this experiment because PREEMPTION-STREAMING was prohibitively slow for larger k.

F.2. Multi-Source Experiments

In Figure 4e, SIEVE-STREAMING++ had the best utility performance. In Figure 7a, we set $k = 50$ and $\varepsilon = 0.6$ and now we see that BATCH-SIEVE-STREAMING++ and SAMPLE-ONE-STREAMING have higher utility, and that this utility increases as the buffer size increases. However, in this case too, the main message is that the utilities of the three algorithms are comparable, but BATCH-SIEVE-STREAMING++ uses the fewest adaptive rounds (Figure 4f).

In Figures 7c through 7f, we display the same set of graphs as Figures 4e through 4h, but for the YouTube experiment. In the YouTube experiment, it is more difficult to select a set of items that is significantly better than random, so we

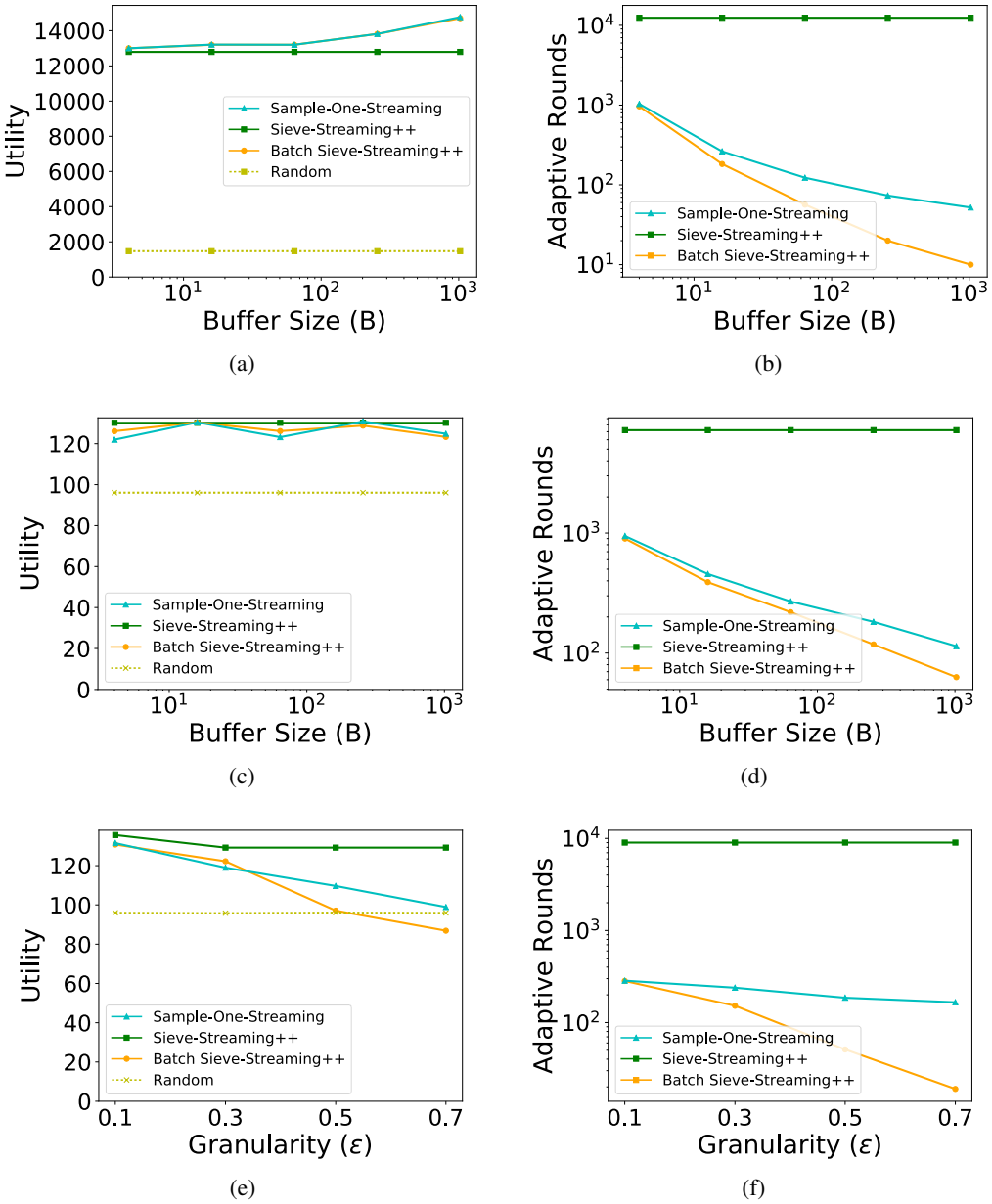

Figure 7. Additional multi-source graphs. (a) and (b) are additional graphs for the Twitter dataset, this time with $\varepsilon = 0.6$ and $k = 50$. (c) through (f) are the equivalent of Figures 4e through 4h, but for the YouTube dataset. Unless they are being varied on the x-axis, we set $\varepsilon = 0.25$, $\mathrm{B} = 100$, and $k = 100$.

need to use a smaller value of ε. We see that for smaller ε, the difference in adaptive rounds between BATCH-SIEVE-STREAMING++ and SAMPLE-ONE-STREAMING is smaller. This is consistent with our results because the number of adaptive rounds required by SAMPLE-ONE-STREAMING does not change much with ε, while the number of adaptive rounds of BATCH-SIEVE-STREAMING++ increases as ε gets smaller.